BetonQuest - 1.12.0-DEV-170

Table of contents

User Documentation

Objectives List

Developer Documentation

Home

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/index.md)

Welcome!

Docs rework in progress!

There may be some inconsistencies.

BetonQuest is an advanced and powerful quest plugin. It offers RPG-style conversations with NPCs and a very flexible quest system. The *multi-path conversations* can be displayed in many ways ranging from basic clickable chat output, to all-new technologies. BetonQuest is not limited to simple, repetitive quests but can also power highly flexible quests that have never been seen before. This is made possible by *support for more than 20 other plugins* such as Citizens, Heroes, JobsReborn, Magic, McMMO and MythicMobs. You can also interact with other quest plugins such as "Quests" to maintain compatibility with existing Quests on your server.

However, *BetonQuest is not made for intense scripting*! You can write player based scripts with BetonQuest but *we support hooking into Denizen and Skript for a reason*! These plugins allow for far more comfortable (server) scripting!

For more information please visit the page that fits your needs the most:

Im a server owner!

Im a quester!

I want to contribute!

Im a developer and want to add support for my plugin!

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/For-Admins.md)

Information for Administrators🔗

BetonQuest has quite a lot of features and use cases. This page will give you an overview on where BetonQuest works best.

Advantages:

- BetonQuest empowers your staff to write truly unique quests that will make your server special. It offers extensive customizability options which - again - empowers you to also make your servers quests looks special. convIos

- This website provides in-depth learning material for your staff which teaches them all the important BetonQuest basics. It also has guides on related topics such as the setup of a local test server, so your main server will not get flooded with bugs while your staff writes quests.

- BetonQuests community also provides active support on the community discord for you and your staff.

Disadvantages:

- The creation of quests may take longer then with other plugins due to BetonQuests enormous flexibility. This is also true for the learning curve, your staff will need some time to learn BetonQuest. However, one of the goals of the further development of BetonQuest is to reduce the overhead that is needed for simple quests.

Feature[1]	BetonQuest	Quests	BeautyQuests	QuestCreator
Free	✅	✅	✅	❌ (20$)
OpenSource	✅	✅	✅	❌

Feature[1]	BetonQuest	Quests	BeautyQuests	QuestCreator (Private)[2]
API	✓	✓	✓	😩
Version Support	1.13.2-1.15.2	1.7-1.15.2	1.11-1.15.2	1.7-1.15.2
Database Support	SQL Lite & MySQL	✗	✗	MySQL
Bungee Support	✗	✗	✗	✗
Per Player Language	✓	?	✗	✗
Multi-Path Conversations	✓	?	?	✓

Feature [1]	BetonQuest	Quests	BeautyQuests	QuestCreator
Client Side NPCs	✔	?	?	✔
Ingame Editor	✘	✔	?	✔
External Editor	✔	?	?	✘
Organized File Structure	✔	✘	?	✔
Integrated Plugins	27	12	15	23

ProQuest, QuestWorld (2) and Quest by LMBishop have been classified as way to limited or unknown to appear in this table. They are simple task plugins, not plugins that are able to create story driven quests.

1. This table has been last updated on the 19th of June 2020. If there are any mistakes let us know!

2. This means that Open Source plugins cannot add support which is considered an unfriendly policy.

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/For-Questers.md)

For Questers

Using BetonQuest you can create nearly any quest you can imagine.

Being able to write quests with BetonQues is a valuable skill since you can make most admins dreams come true. BetonQuest is not only limited to quests but can also be used in combination with the Denizen or Skript scripting plugins that allow you to create things that would only be possible by coding your own plugin.

Learning how to use BetonQuest is not super easy since it is quite complex due to its enormous flexibility. This website provides great learning material that will help you on your journey to become a good quester. If you struggle to understand something just join the community discord and ask! Support is usually really fast.

After learning BetonQuest you can provide quests for one of over 250 servers that run BetonQuest!

Last update:

Feature List🔗

This page will give you an overview of nearly all of BetonQuests features.

Multi-threaded conversations🔗

Players can have immersive conversation with NPC's by utilizing BetonQuests Multi-Path-Conversation System. Questers can freely define multi-threaded stories, narrated with NPC conversations, and with multiple endings that affect a player's gameplay differently. Questers can also choose between five different styles to display their conversations:

Beautiful Conversations🔗

https://mkpdfs.comwes.eufile:///home/runner/work/BetonQuest/BetonQuest/build/documentation/media/content/Home/MenuConvIO.mp4 Sorry, your browser doesn't support embedded videos.

Custom Notification System🔗

Questers can make use of BetonQuests notifyIO system that provides access to all of Minecafts GUI elements to display notifications. Therefore you can provide information to your players like so:

[Video with all the NotifyIO types]

BetonQuest makes it possible to provide NPC conversations in any language. Each player can set his own language which means that - if translated - all conversations will be in the players native language.

You can give players information about where they are in a quest using the "Journal". The Journal is a book in which you can write content based on the players actions.

[Upcoming] First Class Item System🔗

BetonQuest allows for item customization far beyond what other plugins can do. You can use ANY item from ANY plugin in your quests as rewards or conditions. BetonQuest also features a set of powerful item flags that will add non-vanilla behaviour to your items. For example, you could make a proper "Quest Item" that cannot be dropped and won't be dropped when the player dies. This system is *really* powerful, read the final concept *(https://github.com/BetonQuest/BetonQuest/issues/1037#issuecomment-590513472)* to get an idea of what will be possible.

Countless Integrations🔗

BetonQuest supports dozens of plugins, ranging from standard plugins like WorldGuard/ WorldEdit and Citizens to more advanced ones such as EffectLib.

It also offers support for other quest plugins, so you can just keep your old quests and create further additions to your quest lines based upon the progress in your old plugin.

If just Beton isn't enough you can always just hook into Skript or Denizen too.

Full list of integrations here.

Last update:

Learn Beton

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Learn-Beton/Welcome.md)

Welcome!🔗

Welcome to BetonQuest! Thank you for trying it out. This plugin can do some really cool stuff and therefore requires some learning.

If you ever struggle with anything please don't hesitate to ask in the discord *(https:// discordapp.com/invite/rK6mfHq)* !
Its best to join it now before you forget we have one!

Start your journey by installing BetonQuest.

Last update:

Getting Started

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Learn-Beton/Getting-Started/Setting-up-a-local-test-server.md)

Setting up a local test server🔗

Why do I need a local server?!🔗

You might think that a local test server is useless because you already have a remote server. There are multiple reasons why you really need one:

- Working on a live server could lead to crashes and bugs that your players will have to deal with.
- Making quests can, especially for new questers, lead to unexpected behavior. This can be anything from spawning hundreds of mobs to endlessly giving out items to a player. Exactly the stuff you don't want to happen.
- Working with a test server is usually faster and therefore more productive. You can restart it all the time, change plugin configurations as you wish etc.

Setup of your local server🔗

- Step 1: You have propably heard of Spigot, the biggest server software for Minecraft. We are going to install Paper (an improved version of Spigot) on your computer. Head over to Papers download page *(https://papermc.io/downloads)* and click on the button you see in the image below (the numbers will not match up, ignore that).

- Step 2: Create a new folder for the server in a place you can easily access. You really want to make sure to have nothing in it because that folder is going to get quite a lot of files in it. Move the downloaded file here and rename it to `paper.jar`.

- Step 3: You need a start script to start your server. Open your text editor and create a file named `start.bat` (For Linux and Mac: Create a `start.sh` file instead). Open it and copy this into it:

```
java -Xms1G -Xmx1G -jar paper.jar
```

This script tells java to search for a file named `paper.jar`. The `1G` setting in both the `-Xms` and `-Xmx` options is how much RAM you want to give to the server (`1G` = 1 GigaByte RAM, `2G` = 2 GigaByte RAM, `700M` for 700 MegaBytes, etc.). You should not need more then 1GB in most cases though.
Make sure to save it as a `.bat` file! If you save it as a `.txt` file it will not work.

- Step 4: Launch the server by double-clicking on the start file. A console window will appear and close after a while. Just wait until its gone. Now check the server's folder. You will find a bunch of new folders and files that have been generated. You need to accept the EULA (Minecrafts End-User-License-Agreement)to be able to run a Minecraft server. You will find a link in the eula.txt file. After you read it you need to set `eula=false` to `eula=true`.

Next step: Install BetonQuest.

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Learn-Beton/Getting-Started/
Installing-BetonQuest.md)

Installation🔗

In order to use BetonQuest *(https://www.spigotmc.org/resources/betonquest.2117/)* you (obviously) need to add the `BetonQuest.jar` that you downloaded from Spigot to your plugins folder. If there is no plugin folder in your servers folder simply create one. You also need Citizens *(https://www.spigotmc.org/resources/citizens.13811/)* to make some cool looking NPC's. If you can't afford to buy it you can download it for free. Just give the first paragraph of their Spigot page a good read and you will find the link. Also put the `Citizens.jar` file in your servers plugin folder.

Once you have installed these start up your server!

Next step: Setting up the editor

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Learn-Beton/Getting-Started/
Setting-up-the-editor.md)

Setting up the editor🔗

In theory, you can edit quests with any editor. However, using Visual Studio Code gives you some neat productivity advantages and is highly recommended! It is lightweight and clean but allows the installation of extensions that can make your life a lot easier.

Installation🔗

Head over to their site *(https://code.visualstudio.com)* to install the latest version.

Configuration🔗

> **Caution**
>
> Make sure to set these settings in the bottom right corner of VSCode:
>
> `Spaces:2` Indent using Spaces
>
> `UTF-8` file encoding
>
> `LF` end of line characters
>
> YAML language module

Shows the location of the settings

Code Snippets🔗

To make the process of writing quests a lot easier we created some code snippets for VSCode.
When writing an event/condition/objective these snippets will allow you to use the `Tab` key to automatically complete (just like you know it from minecraft commands).
They will also suggest optional attributes and display a preview with a short description from this wiki.
Just have a look at this animation:

Installation⌗

To install the snippet extension head over to the Visual Studio Marketplace *(https://marketplace.visualstudio.com/items?itemName=BetonQuest.betonquest-code-snippets)*.

Marketplace Screenshot

To install the extension all you have to do is to click on the green install button.
If this for some reason doesn't work go to the extensions view in VSCode (`Ctrl+Shift+X`) and search for `BetonQuest`. A full guide on extensions can be found here *(https://code.visualstudio.com/docs/editor/extension-gallery)*.

Usage⌗

Start typing an event/condition/objective and press `Ctrl + Space`. VSCode will list possible completions.

Navigating between options works by pressing `Tab`. If you want to go backwards you can invert the movement of the cursor by pressing `Shift` and `Tab`.

Some options need to be replaced by material names, messages... They will not show a drop-down menu. If an option does, you need to choose between the offered options. Optional options are displayed as an empty field in the drop down.

To exit and insert everything press `Tab` one last time.

Next step: Quick-Start-Tutorial

Last update:

Quick Start Tutorial

This section will teach you the very basics of BetonQuest.

Checking out the "default" example quest.🔗

Let's start by checking out the build in example quest. You can find it in the BetonQuest directory that has been generated in your plugins folder. The folder you are looking for is named "*default*".

Open it up and find a file called *main.yml*. It contains a lot of options, but you only need to look at this section:

```
1    npcs:
2      '0': innkeeper
```

You will need to change the `'0'` to the ID of the Citizens NPC you want to learn Beton with. You obtain a NPC's ID by selecting it with /npc sel while looking at it and then running /npc id. Execute /q reload and right-click the NPC.

The conversation should start. If it did not, check if you correctly assigned the ID. Ask the Innkeeper for some quests. He will tell you to cut some trees. If you want, type /journal to get the journal and see a new entry. Now, don't try to place any wood blocks. BetonQuest will detect that and increase the number of blocks to destroy. Just go and find some trees that you can cut down. Now you can return to Innkeeper and give him the wood. You will receive the reward.

Using events and conditions🔗

Now that you know how a (very) simple quest looks like time to start learning how to write something similar. Let's start with events. We won't do conversations just now, since they heavily use events and conditions, so you need to know them first. You can read complete reference to events in the Reference chapter. Do that now or just continue with this tutorial.

Events🔗

Let's just open *events.yml* file inside the *default* package. At the end add a new line:

```
1    mega: "message Hello world!"
```

This is an event instruction. BetonQuest will use it to determine what type of event it is and what exactly should it do. `mega` is the name, `message` is the events type and `Hello world!` tells the message event what it needs to display. In this case, if you run `mega` event, it will display to you `Hello world!` message. Now save the file, issue /q reload command and run the event with /q e {name} mega command (`q` is shortcut for `quest`, `e` is shortcut for `event`, `{name}` is your Minecraft name without the brackets and `mega` is the name of the event we've just created). It should show you white `Hello world!` message in the chat.

Let's create another event, more complicated one. `teleport` seems complicated enough. As you can read in the Events list, it needs a single argument, the location. Press F3 and check out your current location (it's shown on the left, three numbers, x, y and z). Now add in *events.yml* another line:

```
1    tp: "teleport 100;200;300;world"
```

and replace `100` with your `x` coordinate, `200` with `y` and `300` with `z`. `world` needs to be replaced with your current world's name. Save the file, reload the plugin (/q reload) and run this event with a command described before. It should teleport you to the location you have specified.

Congratulations, you have just created your first events. Go ahead and do some experiments with other event types. You can find them in Events list chapter. Once you're done let's start learning conditions.

Conditions🔗

Open the *conditions.yml* file and add a new line:

```
1    mega: "location 100;200;300;world 5"
```

Can you see how we named the `mega` condition in the same way as the `mega` event? They are not connected in any way. Condition names and event names are separated, so you can give them the same name without any problems. Now let's look at the instruction string. As you can suspect, `location` is a type of the condition. This one means that we'll be checking if the player is near that location (you should change the location to the place where you're standing right now, so you don't have to run around the world). Note that at the end of location argument there is an additional number, 5. This is the maximum distance you can be away from the location to meet the condition. Alright, save the file and reload the plugin.

Now walk to the location you have defined in the condition. Try to stand on the exact block corresponding to that location. Issue /q c {name} mega command (`c` is shortcut for `condition`). It should show you "checking condition blah blah blah: true". We're focusing on that last word, true. This means that you're meeting the condition: you're standing withing 5 block radius of the

location. Now move 2 blocks away and issue that command again. You should still be meeting the condition. Walk 4 more blocks away and try now. It should show false. You are now outside of that 5 block radius. Get it? Great.

Now I'll show you the simplest use of those conditions. Open the *events.yml* file again, and at the end of `mega` instruction add `conditions:mega` argument. By the way, rename your events to something that actually fits the type of the event, otherwise it will get confusing really fast. Example:

events.yml

```
# This is a comment btw
# Old:
mega: "message Hello world!"

# New:
sayHello: "message Hello world! conditions:isAtSpawn"
```

conditions.yml

```
# Old:
mega: "location 100;200;300;world 5"

# New:
isAtSpawn: "location 100;200;300;world 5"
```

Now `sayHello` event will run only if it meets `isAtSpawn` condition. Reload the plugin, walk outside of the 5 block radius and try to run `sayHello` event. Puff, nothing happens. It's because you're not meeting `isAtSpawn` condition. Walk into the radius again and try to run that event now. It should happily display the `Hello world!` message.

It's very nice that we can add such conditions, but the problem is: what if you wanted to display the message only if the player is *outside* the radius? Don't worry, you don't have to specify `inverted_location` condition or anything like that. You can simply negate the condition. Negation makes the condition behave in the exact opposite way, in this case it `isAtSpawn` will be met only if the player is outside of the 5 block radius, and it won't be met if he's inside. Open the *events.yml* and add an exclamation mark before the `isAtSpawn` condition, so it looks like this:

```
1    sayHello: "message Hello world! conditions:!isAtSpawn"
```

This means "display message `Hello world!` if the `isAtSpawn` condition is *not met*". Save the file, reload the plugin and run the event inside and outside of the radius to see how it works.

Basic tags🔗

Now that you know how to use events and conditions I'll show you what tags are. Create new events:

```
1    add_beton_tag: "tag add beton"
2    del_beton_tag: "tag del beton"
```

It's a good practice to give your events names that describe what they are doing. Imagine you have 100 events, `foo24`, `bar65`, `baz12` etc. You would get lost pretty quickly. So, `add_beton_tag` event here simply adds `beton` tag to the player, `del_beton_tag` removes it. Save the file, reload the plugin and run this event. Nothing happens... or does it? Issue /q t {name} command (`t` is shortcut for `tags`). It should show you a list with few entries. Right now focus on `default.beton`, the rest are used by the default quest for Innkeeper. Alright, `default` is the name of the package in which the tag is, and `beton` is the name of the tag, as defined in `add_beton_tag` event. Now run `del_beton_tag` event. Guess what, `default.beton` disappeared from the list! And that's it, you know how to add and remove tags. Pretty useless.

Nothing could be more wrong. Tags are one the most powerful things in BetonQuest. They just need to be used with `tag` condition. Open *conditions.yml* and add

```
1    has_beton_tag: "tag beton"
```

line. As you can imagine, `tag` is the type of condition (the same as `tag` event, but these are not the same things - one is an event, the other one is a condition) and `beton` is the name of the tag. You don't have to specify `default.beton`, but you can if you want. Now save, reload and check it with a command. It should show false, since you have removed the tag with `del_beton_tag` event. Add it again with `add_beton_tag` event and check the `has_beton_tag` condition again. Now it will show true.

Now you probably understand how powerful this system is. You could for example set a tag on the first time the player talks with an NPC, and if the NPC sees that tag next time they talk, he will tell something different, like "welcome back".

Creating objectives🔗

Time to write some objectives! Open the *objectives.yml* file and add a new line:

```
1    kill_creepers: "mobkill CREEPER 3 events:bar conditions:has_beton
```

Now let's analyze it. `kill_creepers` is a name of the objective. `mobkill` is a type. In this case, to complete the objective the player will have to kill some mobs. `creeper` is a type of the mob, so we know that these mobs will have to be Creepers. `3` is the amount. It means that the player has to kill 3 Creepers. `events:bar` means than once the player kills those Creepers, the `bar` event will be run (it's the teleportation event). `conditions:has_beton_tag` tells us that the player will have to have `beton` tag while killing Creepers to complete the objective. Save it, reload the plugin and issue /q o {name} add kill_creepers command (`o` is for `objective`, `add` tells the plugin to add an objective).

Now you can check if you actually have this objective with /q o {name} command, it will show you all your active objectives. It should show `default.kill_creepers`. Alright, remove (yes, remove!) the `beton` tag from you and find some Creepers to kill. Once you killes 3 of them you will notice that nothing happened. It's because `has_beton_tag` condtion is not met, so the objective does not count your progress. Now add the tag again and kill another Creepers. When the third is dead you should be teleported to the location defined in `bar` event.

Congratulations, now you know how to use objectives. You should experiment with other types now, since objectives will be used very often in your quests. Once you're done check out the *Writing your first conversation* chapter to use your knowledge to write your fisrt conversation.

Writing your first conversation🔗

Now that you have seen BetonQuest in action and understood events, conditions and objectives, it's time for writing your first conversation. There's a *conversations* directory inside the default package. It contains a single file, *innkeeper.yml*. This is the conversation with Innkeeper, the one who asks you to cut some trees. Open it, we'll use that for reference. Now create a new file, let's say *miner.yml*. Now type (don't copy-paste it, you'll learn better while typing) that into the file:

```
1   quester: Miner
2   first: greeting
3   NPC_options:
4     greeting:
5       text: Hi there, traveler!
```

It's the most basic conversation possible. The NPC named `Miner` upon starting the conversation will use `greeting` option, which means he will say `Hi there, traveler!`. Then the conversation will end, because there are no player options defined. Now you need to link the conversation with an NPC. You do that in the *main.yml* file. Open it now. As you can see, conversation is linked to `Innkeeper` word. It's the one you have put on the sign, remember? Now, add another line under the Innkeeper: `Miner: miner.yml`, save the file and reload the server. This will link our new conversation with the NPC named "Miner". Construct a new NPC on the server, give him a sign with "Miner" name and click on the head.

Guess what, the conversation finished right after it started. The Miner just said `Hi there, traveler!`, as expected. Now go to the conversation file and edit it (again, manually, no copy-paste!) so the options look like this:

```
1   NPC_options:
2     greeting:
3       text: Hi there, traveler!
4       pointers: hello, bye
5   player_options:
6     hello:
7       text: Hello!
8     bye:
9       text: I need to go, sorry.
```

When you save the file, reload the plugin and start the conversation again you will notice that there are two options for you to choose: `Hello!` and `I need to go, sorry.` Choosing any of them will end the conversation, because these options did not specify any pointers.

Now add a new NPC option, for example `weather` with text `Nice weather.` and make `hello` player option point to it. When you save&reload, the Miner should say `Nice weather.` when you tell him `Hello!`. I think you get how it works.

```
1    NPC_options:
2      greeting:
3        text: Hi there, traveler!
4        pointers: hello, bye
5      weather:
6        text: Nice weather.
7    player_options:
8      hello:
9        text: Hello!
10       pointer: weather
11     bye:
12       text: I need to go, sorry.
```

Now, every time you talk to the Miner, he will say the same thing. It would be nice if the second time you talk to him he knew your name. We can do that with tags. Define a `meet_miner` event and `has_met_miner` condition. When you talk to the Miner for the first time, he will check if you have met him. If not, he will meet you (with that event) and next time you talk, the condition will be passed and he will use your name.

Now, rename `greeting` NPC option to `first_greeting`. Add `meet_miner` event and negated `has_met_miner` condition (negated because this option should only show if the player

has not met the Miner yet). You will need to surround the condition with `''`, because strings cannot start with exclamation marks in YAML. It should look like this:

```
1  first: first_greeting
2  NPC_options:
3    first_greeting:
4      text: Hi there, traveler!
5      condition: '!has_met_miner'
6      event: meet_miner
7      pointers: hello, bye
```

This means: `first_greeting` should be used if the player does not pass `has_met_miner` condition (meaning he doesn't have a tag because he haven't talked to the NPC yet). When this option is used, it will fire `meet_miner` event and display `hello` and `bye` options. Alright, but what happens if the player met the Miner and now negated `has_met_miner` condition doesn't work? NPC will try to use next option defined in `first` setting. There is none yet, so let's add it.

```
1  first: first_greeting, regular_greeting
2  NPC_options:
3    regular_greeting:
4      text: Hi %player%!
5      pointers: hello, bye
```

This option does not have any conditions, so if the `first_greeting` fails, the NPC will always choose this one. Now take a look at the `%player%` thing. It's a variable. In this place it will show your name. There are more than this one, they are described in *Reference* chapter. Alright, save&reload and start the conversation. If you did everything correctly, the Miner should greet you as a "traveler", and the second time you talk to him, he should greet you with your Minecraft name.

Here's the whole conversation you created, so you can check if you understood everything correctly:

```
1   first: first_greeting, regular_greeting
2   NPC_options:
3     first_greeting:
4       text: Hi there, traveler!
5       condition: '!has_met_miner'
6       event: meet_miner
7       pointers: hello, bye
8     regular_greeting:
9       text: Hi %player%!
10      pointers: hello, bye
11    weather:
12      text: Nice weather.
13  player_options:
14    hello:
15      text: Hello!
16      pointer: weather
17    bye:
18      text: I need to go, sorry.
```

Now you should experiment some more with this conversation, you can help yourself by looking at the innkeeper.yml *(https://github.com/BetonQuest/BetonQuest/blob/master/src/main/ resources/default/defaultConversation.yml)* file. Try to understand how that conversation works step by step. As the excercise you should complete the Miner NPC, so he asks you to mine some iron ore, then smelt it in the furnace, next craft an armor with it and return to him wearing this armor.

You might want to check out the Reference chapter to see how to handle items in your quests and how to add entries to the journal. If you want to use Citizens NPCs instead of the ones made with clay you will find information you need in that chapter too. To find out more about events, conditions, objectives and variables, take a look at the appropriate lists (after the *Reference* chapter).

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Learn-Beton/Frequently-Asked-Questions.md)

Frequently Asked Questions🗔

If you have any questions please read it first. It's very likely that it has been already asked and answered. If not, feel free to ask us in the discord *(https://discordapp.com/invite/rK6mfHq)* !.

Q: *Can you make conversation options clickable?*

A: Open *config.yml* file and set `default_conversation_IO` option to "tellraw". You can also set it to "chest" if you want conversations to be displayed in an inventory GUI.

Q: *Can you add particles over NPCs' heads like in `Quests` plugin?*

A: Install EffectLib *(https://dev.bukkit.org/bukkit-plugins/effectlib/)*.

Q: *The players don't know they have to end a conversation, can you add "auto-ending" when they walk away?*

A: Set `stop` option to "false" in conversation file.

Q: *I have an error which says "Cannot load plugins/BetonQuest/{someFile}.yml", what is wrong?*

A: You have incorrect YAML syntax in your conversation file. Check it with YAML Lint *(http://yamllint.com)* to see what's wrong. Usually it's because you started a line with ! or &, forgot colons or made some weird things with apostrophes.

Q: *Where is a command for creating quests?*

A: There is no such command. BetonQuest is too complex to edit it with chat, commands and inventory windows. If you don't like editing files directly you can get the editor *(https://github.com/BetonQuest/BetonQuest-Editor)*.

Q: *Conversations are not working! I created NPC "Innkeeper" and he won't talk to me.*

A: Conversations are not linked to an NPC through names, as you can have multiple Innkeepers. You need to connect them with their ID. Read this.

Q: *Could you add some feature?*

A: Check if it wasn't already added in DEV-builds *(https://betonquest.pl)* . You can see all changes in the changelog *(https://github.com/BetonQuest/BetonQuest/blob/master/ CHANGELOG.md)* .

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Learn-Beton/Tips-and-tricks.md)

Tips and tricks🔗

Handling death in your quests🔗

Sometimes, while writing a dangerous quest you will want something specific to happen when the player dies. If it's a boss battle you may want to fail the quest, if it's a dungeon you may want to respawn the player at the beginning of a level etc. You can do that with `die` objective - simply start it for the player at the beginning of the quest and make it fire events that will do the thing you want (like teleporting the player to desired respawn point, removing tags set during the quest etc). You can add `persistent` argument to the objective instruction string to make it active even after completing it. Remember to `delete` it after the quest is done!

Creating regions for one player at the time🔗

Imagine you have a room to which the player is teleported. Then suddenly mobs start to spawn and the player must kill them (because it's a trap or something). The player has killed all the mobs, he got a tag and wants to proceed but all of the sudden another player teleports into the room and all the mobs start to spawn again. The first player is quickly killed and the second one easily kills all mobs. You can prevent such situations by using `party` condition. Just check with it if the party consisting of "players inside the room" has greater amount of players that 1. Set the range to something big enough so it covers the room and the party condition can be tag or location.

Racing with folder event🔗

Since `folder` event can run `tag` events even for offline players you can create races. Create `location` objective where you want the finish line to be and condition it with negated "race_failed" tag (or similar). It will mean that "if the player has not failed the race, he can win it by reaching the location". Now when the race starts fire `folder` event with the amount of time you want to give your players to complete the race. This event should set "race_failed" tag. If the player reaches the location before this tag is set, he will fire all events in that `location` objective, but if the time has passed, the objective will not be completed. You can figure the rest out for yourself.

Random daily quests🔗

Starting the random quest must be blocked with a special tag. If there is no such tag, the conversation option should appear. Create a few quests, each of them started with single `folder` event (they must be started by single event!). Now add those events to another `folder`

event and make it `random:1`. At the end of every quest add `delay` which will reset the special blocking tag. Now add that `folder` event to the conversation option. When the player chooses it he will start one random quest, and the conversation option will become available after defined in `delay` objective time after completing the quest.

Each day different quest (same for every player)🔗

To do this use something called "Static event". Using the static event run `folder` event every day at some late hour (for example 4am). The `folder` event should be `random:1` and contain several different `setblock` events. These events will set some specific block to several different material types (for example dirt, stone, wood, sand etc). Now when the player starts the conversation and asks about the daily quest the NPC should check (using `testforblock` condition) which type of block is currently set and give the player different quest, depending on the block type.

Make the NPC react randomly🔗

Imagine you want to lie to NPC and he has 15% chance of believing you completely, 35% of being suspicious and 50% of not believing at all. The common denominator for those percentages is 20, so we can write it as 3/20, 7/20 and 10/20. The NPC will check options one after another until it finds one which meets all conditions. We will use `random` condition with our options. The first one will have `3-20` chance (that's the format used by `random` condition). If this condition fails, the NPC will check next option. But it won't be `7-20`, because we already "used" 3 of 20. If you wrote it like that, the chance would be too low. That's why it will be `7-17`. The third option should have `10-10` (because `17 - 7 = 10` and 50% is 10/20), but as you can see it will always be true. It's because we want the last option to be shown if both previous fail. You don't have to add the last condition at all.

Quest GUI🔗

If you want your players to be able to choose a quest everywhere, every time, then you can create a conversation which can be started with an item. This one is a little hacky but it shows flexibility of BetonQuest. First you need a conversation which behaves as a quest choosing GUI. Name the NPC "Quester", add one option for each quest etc. Now you need an objective which will start this conversation using `conversation` event. It should be `action` objective, set to right click on any block. Add `hand` condition to make it accept only clicks with a specific item and make the objective `persistent` (so players can use it multiple times). The item used here should be marked as Quest Item so players can't drop it. Now define new global location covering your whole map and using it start the objective and give players the item. This way all players (existing and new) will get the quest item, which opens a GUI with quests when right clicked.

Last update:

User Documentation

Reference🔗

This chapter describes all aspects of BetonQuest in one place. You should read it at least once to know what you're dealing with and where to search for information if you ever have any problems.

Conditions, Events and Objectives🔗

Conditions, events and objectives are defined with an "instruction string". It's a piece of text, formatted in a specific way, containing the instruction for the condition/event/objective. Thanks to this string they know what should they do. To define the instruction string you will need a reference, few pages below. It describes how something behaves and how it should be created. All instruction strings are defined in appropriate files, for example all conditions are in *conditions.yml* config. The syntax used to define them looks like this: `name: 'the instruction string containing the data'`. Apostrophes are optional in most cases, you can find out when to use them by looking up "YAML syntax" in Google.

Conditions🔗

Conditions are the most versatile and useful tools in creating advanced quests. They allow you to control what options are available to player in conversations, how the NPC responds or if the objective will be completed. The reference of all possible conditions is down below.

You can negate the condition (revert its output) by adding an exclamation mark (`!`) at the beginning of it's name (in the place you use it, i.e. in conversations, not in the *conditions.yml* file).

You can use conversation variables instead of numeric arguments in conditions. If the variable fails to resolve (i.e. it will return an empty string) BetonQuest will use 0 instead.

Events🔗

In certain moments you will want something to happen. Updating the journal, setting tags, giving rewards, all these are done using events. You define them just like conditions, by specifying a name and instruction string. You can find instruction strings to all events in the event reference. At the end of the instruction string you can add `conditions:` or `condition:` (with or without `s` at the end) attribute followed by a list of condition names separated by commas, like `conditions:angry,!quest_started`. This will make an event fire only when these conditions are met.

You can use conversation variables instead of numeric arguments in events. If the variable fails to resolve (i.e. it will return an empty string) BetonQuest will use 0 instead.

Objectives🔗

Objectives are the main things you will use when creating complex quests. You start them with a special event, `objective`. You define them in the *objectives.yml* file, just as you would conditions or events. At the end of the instruction string you can add conditions and events for the objective. Conditions will limit when the objective can be completed (e.g. killing zombies only at given location in quest for defending city gates), and events will fire when the objective is completed (e.g. giving a reward, or setting a tag which will enable collecting a reward from an NPC). You define these like that: `conditions:con1,con2 events:event1,event2` at the end of instruction string . Separate them by commas and never use spaces! You can also use singular forms of these arguments: `condition:` and `event:`.

If you want to start an objective right after it was completed (for example `die` objective: when you die, teleport you to a special spawnpoint and start `die` objective again), you can add `persistent` argument at the end of an instruction string. It will prevent the objective from being completed, although it will run all its events. To cancel such objective you will need to use `objective delete` event.

Objectives are loaded at start-up, but they do not consume resources without player actually having them active. This means that if you have 100 objectives defined, and 20 players doing one objective, 20 another players doing second objective, and the rest objectives are inactive (no one does them), then only 2 objectives will be consuming your server resources, not 100, not 40.

Packages🔗

All the content you create is organized into packages. Each package must contain the *main.yml* file with package-specific settings. Additionally it can have *conversations* directory with conversation files, *events.yml*, *conditions.yml*, *objectives.yml*, *items.yml* and *journal.yml*. The default package is called simply "default". It is always present, and if you delete it, it will be regenerated with a sample quest.

If you want, you can simply ignore the existence of packages and write all your quests in the default one. You will however come to a point, when your files contain hundreds of lines and it gets a little bit confusing. That's why it's better to split your quests into multiple packages, for example "main" for the quests in the main city, "dungeon" for some interesting dungeon story etc.

Each package can be disabled/enabled in the *main.yml* file, by setting `enabled` to `true` or `false`.

It would be limiting if you couldn't interact between packages. That's why you can always access stuff from other packages by prefixing its name with package name. If you're writing a conversation in package `village` and you want to fire an event `reward` from package `beton`, you simply name the event as `beton.reward`. The plugin will search for `reward` in `beton` package instead of the one in which the conversation is defined. All events, conditions, objectives, items and conversations behave this way. Note that you can't cross-reference journal entries!

Packages can be nested together in folders. A folder can either contain packages or be a package, never both. The name of such nested package is prefixed by names of all folders, separated with a dash. This directory tree (*main.yml* represents a package):

```
BetonQuest/
├─default/
└─quests/
  ├─village1/
  │ ├─quest1/
  │ │ └─main.yml
  │ └─quest2/
  │   └─main.yml
  └─village2/
    └─quest1/
      └─main.yml
```

contains these packages:

- *default*
- *quests-village1-quest1*
- *quests-village1-quest2*
- *quests-village2-quest1*

Relative paths🔖

You don't have to always specify a full package name. In the example above, if you wanted to reference package `quests-village1-quest2` from `quests-village1-quest1`, you could write it as `_-quest2` - this means "from the current package (`quest1`) go up one time and find there package `quest2`". Name `_` has special meaning here. Analogously, if you wanted to reference `quests-village2-quest1` from that package, you would use `_-_-village2-quest1`. Using `_` twice will get you to `quests` package, and from there you're going into `village2` and then `quest1`.

Relative paths can be useful if you want to move your packages between directories. Instead of rewriting every package name to match current directory tree, you can use relative paths and it will just work. It can also be useful if you want to create a downloadable quest package which can be placed anywhere and not just on the root directory (*BetonQuest/*) or when working with

BetonQuest-Editor + BetonQuestUploader combo - quest writers don't have to prefix every package call with their name (this feature will work soon, needs an update in the editor).

Unified location formating🔗

Whenever you want to define some location in your events, conditions, objectives or any other things, you will define it with this specific format. The location can consist of 2 things: base and vector. Only the base is always required.

The base is a core location. Currently there are two types: absolute coordinates and variables. Absolute coordinates are defined like `100;200;300;world`, where `100` is X coordinate, `200` is Y, `300` is Z and `world` is the name of the world. These can have decimal values. If you want you can also add two more numbers at the end, yaw and pitch (these are controlling the rotation, for example in teleportation event, both are needed if you decide to add them; example: `0.5;64;0.5;world;90;-270`).

To use a variable as the location's base it must resolve to a valid absolute coordinates. An example of such variable is `%location%`, which shows player's exact location. Simply place it instead of coordinates. There is one rule though: you can't use variable base types in events running without players (for example static events or the ones run from folder event after their player left the server). BetonQuest won't be able to resolve the location variable without the player!

The vector is a modification of the location. It will be useful if you use global variables instead of base (described in the next subsection) or the base itself is variable, like `player`. Vectors look like `->(10;2.5;-13)` and are added to the end of the base. This will modify the location, X by 10, Y by 2.5 and Z by -13. For example, location written as `100;200;300;world_nether->(10;2.5;-13)` will generate a location with X=110, Y=202.5 and Z=287 on `world_nether` world.

Global variables🔗

You can insert a global variable in any instruction string. It looks like this: `$beton$` (and this one would be called "beton"). When the plugin loads that instruction string it will replace those variables with values assigned to them in *main.yml* file before the instruction string is parsed. This is useful for example when installing a package containing a WorldEdit schematic of the quest building. Instead of going through the whole code to set those locations, names or texts you will only have to specify a few variables (that is, of course, if the author of the package used those variables properly in his code).

Note that these variables are something entirely different than conversation variables. Global ones use $ characters and conversation ones use % characters. The former is resolved before the instruction string is parsed while the latter is resolved when the quests are running, usually on a per-player basis.

```
variables:
  village_location: 100;200;300;world
  village_name: Concrete
```

Canceling quests🔗

If you want to let your players cancel their quest there is a function for that. In *main.yml* file there is `cancel` branch. You can specify there quests, which can be canceled, as well as actions that need to be done to actually cancel them. You can find an example in the `default` package. The arguments you can specify are:

- `name` - this will be the name displayed to the player. All _ characters will be converted to spaces. If you want to include other languages you can add here additional options (`en` for English etc.)
- `conditions` - this is a list of conditions separated by commas. The player needs to meet all those conditions to be able to cancel this quest. Place there the ones which detect that the player has started the quest, but he has not finished it yet.
- `objectives` - list of all objectives used in this quest. They will be canceled without firing their events.
- `tags` - this is a list of tags that will be deleted. Place here all tags that you use during the quest.
- `points` - list of all categories that will be entirely deleted.
- `journal` - these journal entries will be deleted when canceling the quest.
- `events` - if you want to do something else when canceling the quest (like punishing the player), list the events here.
- `loc` - this is a location to which the player will be teleported when canceling the quest (defined as in teleport event);

To cancel the quest you need to open your backpack and select a "cancel" button (by default a bone, can be changes by naming an item "cancel_button" inside default package). There will be a list of quests which can be canceled. Just select the one that interests you and it will be canceled.

Global objectives🔗

If you want a objective to be active for every player right after joining you can create a global objective. This is done by adding `global` argument to the instruction of the objective. When you then reload BetonQuest it is started for all online players and also will be started for every player who joins.

To prevent the objective from being started every time a player joins a tag is set for the player whenever the objective is started and as long as the player has the tag the objective wont be

started again if the player joins.

These tags follow syntax `<package>.global-<id>`, where `<id>` is the objectives id and `<package>` the package where the objective is located.

Possible use cases would be a quest which starts if a player reaches a specific location or breaks a specific block.

Example:

```
start_quest_mine: 'location 100;200;300;world 5 events:start_quest_m
```

Static events🔗

Static events are events that will fire at the specified time of the day. They are not tied to a specific player, so not all of event types can be used as static. (Which player should receive a tag or objective? From which one should the items be taken?) Also, static events cannot have conditions defined (`event-conditions:` argument), as the plugin cannot check any condition without the player. Events, that can be used as static are flagges with `static` keyword in this documentation. You can define your static events in *main.yml* file under `static` section, as such:

```
static:
  '09:00': beton
  '23:59': lightning_strike
  '11:23': some_command
```

The hour must be in `''` to avoid problems, it needs leading zero if less than 10. `beton`, `lightnint_strike` etc. are IDs of events. There can only be one event specified, but it can be of type "folder".

Journal🔗

The journal is a book in which all your adventures are described. You can obtain it by typing /j command or /b and selecting it from backpack. You cannot put it into any chests, item frames and so on. If you ever feel the need to get rid of your journal, just drop it - it will return to your backpack. The journal is updated with the `journal` event, and the text inside is defined in *journal.yml* config file. If you update these texts and reload the plugin, all players' journals will reflect changes. Colors in the journal can be altered in config.yml. The entries can use color codes, but the color will be lost between pages.

The journal by default appears in the last slot of the hotbar. If you want to change that use `default_journal_slot` option in *config.yml*, experiment with different settings until you're ok with it.

If you want to translate the entry do the same thing as with conversation option - go to new line, add language ID and the journal text for every language you want to include.

You can control behavior of the journal in *config.yml* file, in `journal` section. `chars_per_page` specifies how many characters will be placed on a single page. If you set it too high, the text will overflow outside of the page, too low, there will be too much pages. `one_entry_per_page` allows you to place every entry on a single page. The `chars_per_page` setting is in this case ignored, BetonQuest will put entire entry on that page. `reversed_order` allows you to reverse order of entries and `hide_date` lets you remove the date from journal entries.

You can control colors in the journal in `journal_colors` section in *config.yml*: `date` is a color of date of every entry, `line` is a color of lines separating entries and `text` is just a color of a text. You need to use standard color codes without & (eg. `'4'` for dark red).

You can also add a main page to the journal. It's a list of texts, which will show only if specified conditions are met. You can define them in the *main.yml* file, in the `journal_main_page` section:

```
journal_main_page:
  title:
    priority: 1
    text:
      en: '&eThe Journal'
      pl: '&eDziennik'
    conditions: 'quest_started,!quest_completed'
```

Each string can have text in different languages, list of conditions separated by commas (these must be met for the text to show in the journal) and `priority`, which controls the order of texts. You can use conversation variables in the texts, but they will only be updated when the player gets his journal with the /journal command. Color codes are supported.

If you want your main page take a separate page (so entries will be displayed on next free page), set `full_main_page` in *config.yml* to "true".

Tags🔗

Tags are little pieces of text you can assign to player and then check if he has them. They are particularly useful to determine if player has started or completed quest. They are given with `tag` event and checked with `tag` condition. All tags are bound to a package, so if you add `beton` tag from within the `default` package, the tag will look like `default.beton`. If you're checking for `beton` tag from within `default` package, you're actually checking for `default.beton`. If you

want to check a tag from another package, then you just need to prefix it's name with that package, for example `quest.beton`.

Points🔗

Points are like tags, but with amount. You can earn them for doing quest, talking with NPC's, basically for everything you want. You can also take the points away, even to negative numbers. Points can be divided to categories, so the ones from *beton* category won't mix with points from *quests* group. Of course then you can check if player has (or doesn't have) certain amount and do something based on this condition. They can be used as counter for specific number of quest done, as a reputation system in villages and even NPC's attitude to player.

NPCs🔗

Conversations can be assigned to NPCs. You do it in the *main.yml* file inside a package, in "npcs" section:

```
npcs:
  '0': innkeeper
  'Innkeeper': innkeeper
```

The first string is the name of the NPC, second one is the corresponding conversation name. In case you use Citizens, name is the ID of an NPC (*don't try to put Citizens NPC's name here, it must be the ID*). To acquire it just select the NPC and type `/npc`. If you don't want to use Citizens, you can also build NPCs as any other building in Minecraft:

Place somewhere a block of stained clay, no matter the color. Then place a head on top of it (type doesn't matter, it must be head). Now place a sign on the side of the clay block (it can be on it's back) and type in the first line `[NPC]`, and on the second line the ID of the NPC (in case of the above code example, the ID would be `Innkeeper`). You need to have permission `betonquest.createnpc` for that. Congratulations, you have created the NPC. Now you can add levers (hands) to it and maybe even a fence gate (legs). Conversation is started by right clicking it's head.

> **Note**
>
> When using Citizens ID's they must be enclosed in quotes.

Items🔗

Items in BetonQuest are defined in *items.yml* file. Each item has an instruction string, similarly to events, conditions etc. Basic syntax is very simple:

```
item: BLOCK_SELECTOR other arguments...
```

BLOCK_SELECTOR is a type of the item. It doesn't have to be all in uppercase. Other arguments specify data like name of the item, lore, enchantments or potion effects. There are two categories of these arguments: the ones you can apply to every item and type specific arguments. Examples would be name (for every item type) and text (only in books).

Every argument is used in two ways: when creating an item and when checking if some existing item matches the instruction. The first case is pretty straightforward - BetonQuest takes all data you specified and creates an item, simple as that. Second case is more complicated. You can require some property of the item to exist, other not to exist, or skip this property check altogether. You can also accept an item only if some value (like enchantment level) is greater/less than *x*. You can use wildcards in the BLOCK_SELECTOR to match multiple types of items.

These are arguments that can be applied to every item:

- `name` - the display name of the item. All underscores will be replaced with spaces and you can use & color codes. If you want to specifically say that the item must not have any name, use `none` keyword.

- `lore` - text under the item's name. Default styling of lore is purple and italic. All underscores will be replaced with spaces and you can use & color codes. To make a new line use `;` character. If you require the item not to have lore at all, use `none` keyword. By default lore will match only if all lines are exactly the same. If you want to accept all items which contain specified lines (and/or more lines), add `lore-containing` argument to the instruction string.

- `data` - data value of the item. An example of this would be different wool colors or damage of a pickaxe. You can require an exact value by specifying a number, or a value greater/less (and equal) than specified by adding +/- character at the end of the number.

- `enchants` - a list of enchantments and their levels. Each enchantment consists of these things, separated by colons:

  - name *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/enchantments/ Enchantment.html)*
  - level (a positive number)

  For example `damage_all:3` is *Sharpness III*. You can specify additional enchantments by separating them with commas.

  You can require the item not to have any enchantments by using `none` keyword. You can also add +/- character to the enchantment level to make the check require levels greater/less (and equal) than specified. If you don't care about the level, replace the number with a question mark.

By default all specified enchantments are required. If you want to check if the item contains a matching enchantment (and/or more enchants), add `enchants-containing` argument to the instruction string. Each specified enchantment will be required on the item by default unless you prefix its name with `none-`, for example `none-knockback` means that the item must not have any knockback enchantment. Do not use `none-` prefix unless you're using `enchants-containing` argument, it doesn't make any sense and will break the check!

- `unbreakable` - this makes the item unbreakable. You can specify it either as `unbreakable` or `unbreakable:true` to require an item to be unbreakable. If you want to check if the item is breakable, use `unbreakable:false`.

Examples:

```
name:&4Sword_made_of_Holy_Concrete
name:none
lore:&cOnly_this_sword_can_kill_the_Lord_Ruler
lore:&2Quest_Item lore-containing
lore:none
data:5
data:500-
enchants:damage_all:3+,none-knockback
enchants:power:? enchants-containing
enchants:none
unbreakable
unbreakable:false
```

These are the arguments that can be applied only to specific item types:

Books📖517

This applies to a written book and a book and quill.

- `title` - the title of a book. All underscores will be replaced with spaces and you can use & color codes. If you want to specifically say that the book must not have any title, use `none` keyword.

- `author` - the author of a book. All underscores will be replaced with spaces, you cannot use color codes here. If you want to specifically say that the book must not have any author, use `none` keyword.

- `text` - the text of the book. All underscores will be replaced with spaces and you can use & color codes. The text will wrap to the next page if amount of characters exceeds `journal.chars_per_page` setting in *config.yml*. If you want to manually wrap the page, use | character. To go to new line use \n. Keep in mind that you can't use any spaces

here, you must only use underscores (_). This needs to be a single argument, even if it's really long. If you don't want the book to have any text, use `none` keyword instead.

Examples:

```
title:Malleus_Maleficarum
author:&eGallus_Anonymus
text:Lorem_ipsum_dolor_sit_amet,\nconsectetur_adipiscing_elit.|Pelle
```

Potions📗

This applies to potions, splash potions and lingering potions.

- `type` - type of a potion. Here's the list *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/ potion/PotionType.html)* of possible types. Do not mistake this for a custom effect, this argument corresponds to the default vanilla potion types.

- `extended` - extended property of the potion (you can achieve it in-game by adding redstone). It can be specified as `extended` or `extended:true`. If you want to check the potion that is NOT extended, use `upgraded:false`.

- `upgraded` - upgraded property of the potion (you can achieve it in-game by adding glowstone). It can be specified as `upgraded` or `upgraded:true`. If you want to check the potion that is NOT upgraded, use `upgraded:false`.

- `effects` - a list of custom effects. These are independent of the potion type. The effects must be separated by commas. Each effect consists of these things, separated by colons:

  - type *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/potion/ PotionEffectType.html)* (this is different stuff that the link above!)
  - power
  - duration (in seconds)

An example would be `WITHER:2:30`, which is a wither effect of level 2 for 30 seconds.

If you want to target only potions without custom effects, use `none` keyword. You can target potions with level and time greater/less (and equal) than specified with +/- character after the number. If you don't care about the level/time, you can replace them with question mark.

By default all specified effects are required. If you want to check if the potion contains these effects among others, add `effects-containing` argument to the instruction string. Now if you want to make sure the potion doesn't contain a specific effect, prefix the effect name with `none-`. Don't use that prefix unless you're also using `effects-containing` argument, it doesn't make any sense and it will break the check.

Examples:

```
type:instant_heal
extended
upgraded:false
effects:poison:1+:?,slow:?:45-
effects:none-weakness,invisibility:?:? effects-containing
```

Heads🔗

This applies to human heads.

- `owner` - this is the name of the head owner. It will not use color codes nor replace underscores with spaces. If you want to check for heads without any owner, use `none` keyword.

Examples:

```
owner:Co0sh
owner:none
```

Leather armor🔗

This applies to all parts of leather armor.

- `color` - this is the color of the armor piece. It can be either one of these values *(https:// hub.spigotmc.org/javadocs/spigot/org/bukkit/DyeColor.html)*, a hexadecimal RGB value prefixed with # character or its decimal representation without the prefix. You can also check if the armor piece doesn't have any color with `none` keyword.

Examples:

```
color:light_blue
color:#ff00ff
color:none
```

Fireworks👤

This applies to fireworks.

- `firework` - this is a list of effects of the firework rocket. They are separated by commas. Each effect consists of these things separated by colons:

  - effect type *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/ FireworkEffect.Type.html)*
  - a list of main colors (refer to leather armor colors above for syntax) separated by semicolons
  - a list of fade colors
  - `true`/`false` keyword for trail effect
  - `true`/`false` keyword for flicker.

  Note the separation characters, this is important: commas separate effects, colons separate effect properties, semicolons separate colors.

  If you want to target fireworks without any effects, use `none` keyword. If you want to target any effect type, use question mark instead of the effect name. If you don't want the effect to have any main/fade colors, use `none` keyword in the place of colors. If you don't care about main/fade colors, use question marks in that place. If you don't care about trail/ flicker effect, use question marks instead of `true`/`false` keyword.

  By default the check will require all specified effects to be present on the firework. You can check if the firework contains specified effects among others by adding `firework-containing` argument to the instruction string. To match the item which must not have an effect, prefix the effect name with `none-` keyword. Don't use that prefix unless you're also using `firework-containing` argument, it doesn't make any sense and will break the check.

- `power` - flight duration of the firework, in levels. You can use +/- character to target greater/less (and equal) levels.

Examples:

```
firework:ball:red;white:green;blue:true:true,ball_large:green;yellow
firework:burst:?:none:?:? firework-containing
firework:none-creeper firework-containing
firework:none
power:3
power:2+
```

Firework charges🔗

This applies to firework charges.

- `firework` - this is almost the same as fireworks. You can only specify a single effect and the `power` argument has no effects.

Backpack🔗

Sometimes you'll want some items to be persistent over death. If the player has lost them the quest would be broken. You can add a apecific line to item's lore to make it persistent (`&2Quest_Item` by default, _ is a space in item's definition). Note that this must be a whole new line in the lore! Such item wouldn't be dropped on death, instead it would be placed in player's backpack. Example: `important_sword: 'DIAMOND_SWORD name:Sword_for_destroying__The_Concrete lore:Made_of_pure_Mithril;&2Quest_Item'`

To open your backpack just type /j command. The inventory window will open, displaying your stored items. The first slot is always the journal, and if you get it, the slot will stay empty. You can transfer quest items back and forth between inventories by clicking on them. Left click will transfer just one item, right click will try to transfer all items. Normal items cannot be stored into the backpack, so it's not an infinite inventory.

If you will ever have more than one page of quest items, the buttons will appear. You can customize those buttons by creating `previous_button` and `next_button` items in *items.yml* file. Their name will be overwritten with the one defined in *messages.yml*.

Quest items cannot be dropped in any way other than using them. This way you can create a quest for eating cookies by giving the player a stack of cookies flagged as quest items and not continuing until there are no more cookies in his inventory/backpack. The player cannot drop the cookies, so he must eat every one of them to complete the quest.

Don't worry if the item-dropping filter isn't working for your items when you're in creative mode - it's not a bug. It's a feature. Creative-mode players should be able to easily put quest items in containers like TreasureChests.

Party🔗

Parties are very simple. So simple, that they are hard to understand if you already know some other party system. Basically, they don't even have to be created before using them. Parties are defined directly in conditions/events (`party` event, `party` conditions, check them out in the reference lists below). In such instruction strings the first argument is a number - range. It defines the radius where the party members will be looked for. Second is a list of conditions. Only the players that meet those conditions will be considered as members of the party. It's most

intuitive for players, as they don't have to do anything to be in a party - no commands, no GUIs, just starting the same quest or having the same item - you choose what and when makes the party.

To understand better how it works I will show you an example of `party` event. Let's say that every player has an objective of pressing a button. When one of them presses it, this event is fired:

```
party_reward: party 50 quest_started cancel_button,teleport_to_dunge
```

Now, it means that all players that: are in radius of 50 blocks around the player who pressed the button AND meet `quest_started` condition will receive `cancel_button` and `teleport_to_dungeon` events. The first one will cancel the quest for pressing the button for the others (it's no longer needed), the second one will teleport them somewhere. Now, imagine there is a player on the other side of the world who also meets `quest_started` condition - he won't be teleported into the dungeon, because he was not with the other players (not in 50 blocks range). Now, there were a bunch of other players running around the button, but they didn't meet the `quest_started` condition. They also won't be teleported (they didn't start this quest).

Block Selectors🔗

When specifying a way of matching a block, a `block selector` is used. This differs a little depending if your Minecraft version is older than 1.13 or newer.

Pre 1.13 Minecraft🔗

The format of a block selector is: `material:data`

Where:

- `material` - What material the block is made of. You can look this up in this list *(https:// hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html)*.

- `data` - (optional) A number representing the data of the block. If left out then the selector will match all data types.

Examples:

- `LOG` - Matches all LOGS

- `LOG:1` - Matches SPRUCE LOGS

1.13 and above🔗

The format of a block selector is: `prefix:material[state=value,...]`

Where:

- `prefix` - (optional) The material prefix. If left out then it will be assumed to be 'minecraft'

- `material` - The material the block is made of. You can look this up in _this list_ _(https://_ _hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html)_. Wildcards are supported (both * and ?).

- `state` - (optional) The block states can be provided in a comma separated list surrounded by square brackets. You can look up states in _this list_ _(https://minecraft.gamepedia.com/_ _1.13/Flattening#Block_states)_. Any states left out will be ignored when matching.

Examples:

- `minecraft:stone` - Matches all blocks of type STONE

- `redstone_wire` - Matches all blocks of type REDSTONE_WIRE

- `redstone_wire[power=5]` - Matches all blocks of type REDSTONE_WIRE and which have a power of 5

- `redstone_wire[power=5,facing=1]` - Matches all blocks of type REDSTONE_WIRE and which have both a power of 5 and are facing 1

- `*_LOG` - Matches all LOGS

- `*` - Matches everything

- `*[waterlogged=true]` - Matches all waterlogged blocks

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/Objectives-List.md)

Objectives List🔗

Action: `action`🔗

This objective completes when player clicks on given block type. This can be further limited by location condition and item in hand condition. First argument is type of the click, it can be right, left or any. Next is a `block selector`. You can also specify `loc:` argument, followed by standard location format and `range:` followed by a number (or variable). It will define where the clicked block needs to be, as opposed to "where you must be" in location condition. If you add argument `cancel`, the click will be canceled (chest will not open, button will not be pressed etc.)

Action objective contains one property, `location`. It's a string formatted like `X: 100, Y: 200, Z:300`. It does not show the radius.

> **Example**
>
> ```
> action right DOOR:1 conditions:holding_key loc:100;200;300;world range:5
> ```

Arrow Shooting: `arrow`🔗

To complete this objective the player needs to shoot the arrow into the target. There are two arguments, location of the target and precision number (radius around location where the arrow must land, should be small). Note that the position of an arrow after hit is on the wall of a _full_ block, which means that shooting not full blocks (like heads) won't give accurate results. Experiment with this objective a bit to make sure you've set the numbers correctly.

> **Example**
>
> ```
> arrow 100.5;200.5;300.5;world 1.1 events:reward conditions:correct_player_position
> ```

Block: `block`🔗

To complete this objective player must break or place specified amount of blocks. First argument after name is a `block selector`. Next is amount. It can be more than 0 for placing and less than 0 for destroying. You can also use `notify` keyword to display messages to the player each time he updates amount of blocks, optionally with the notification interval after colon.

This objective has two properties, `amount` and `left`. Amount is current amount of blocks in the objective, left is amount needed to complete the objective. Note that it may sometimes be negative!

Example

```
block LOG:2 -16 events:reward notify:5
```

Breed animals: `breed`🔗

This works only on Spigot 1.10.2 and later!

This objective is completed by breeding animals of specified type. The first argument is animal type (types *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/EntityType.html)*) and second is amount (positive integer). You can add `notify` argument to display a message with remaining amount each time the animal is bred, optionally with the notification interval after colon. While you can specify any entity, the objective will be completable only for breedable ones.

Example

```
breed cow 10 notify:2 events:reward
```

Put items in a chest: `chestput`🔗

This objective requires the player to put specified items in a specified chest. First argument is a location of the chest, second argument is a list of items (from *items.yml* file), separated with a comma. You can also add amount of items after a colon. The items will be removed upon completing the objective unless you add `items-stay` optional argument.

Example

```
chestput 100;200;300;world emerald:5,sword events:tag,message
```

Eat/drink: `consume`🔗

This objective is completed by eating specified food or drinking specified potion. The only required argument is the ID of an item from *items.yml*.

Example

```
consume tawny_owl events:faster_endurance_regen
```

Crafting: `craft`🔗

To complete this objective the player must craft specified item. First argument is ID of the item, as in *items.yml*. Next is amount (integer).

Crafting objective has two properties, `amount` and `left`. Amount is current amount of crafted items and left is amount needed to complete the objective.

Example

```
craft saddle 5 events:reward
```

Enchant item: `enchant`🔗

This objectie is completed when the player enchants specified item with specified enchantment. The first argument is an item name, as defined it *items.yml*. Second one is the enchantment *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/enchantments/Enchantment.html)* and a level, separated with a colon. If you need to check for multiple enchantments you can add a list of them, separated by colons.

Example

```
enchant sword damage_all:1,knockback:1 events:reward
```

Experience: `experience`🔗

This objective can by completed by reaching specified amount of experience points. You can check for whole levels by adding the `level` argument. The conditions are checked when the player levels up, so if they are not met the first time, the player will have to meet them and levelup again.

Example

```
experience 25 level events:reward
```

Delay: `delay`🔗

This objective is just a long, persistent delay for firing events. It will run only after certain amount of time (measured in minutes) and only when player is online and meets all conditions. If a player is offline at that time it will just wait for them to log in. You should use it for example to delete tags so the player can complete quests multiple times. First argument is time, by default in minutes. You can also use `ticks` or `seconds` argument to use different units, but keep in mind that it's not very precise - it will complete roughly after the time ends. To control that precision you can specify an optional `interval:` argument, which specifies how many ticks should pass between checks. One second is 20 ticks. Less makes the objective more precise, at the expense of performance. The rest is just like in other objectives.

Delay has two properties, `left` and `date`. The first one will show how much time needs to pass before the delay is completed (i.e. `23 days, 5 hours and 45 minutes`), the second one will show a date of completing the objective formatted using `date_format` setting in *config.yml* (it will look like the one above every journal entry).

Example

```
delay 1000 ticks interval:5 events:event1,event2
```

Death: `die`🔗

Death objective completes when the player dies meeting all conditions. You can optionally cancel death with `cancel` argument. It will heal player and optionally teleport him to respawn location. There can be two arguments: `cancel`, which is optional, and `respawn:`, which is also optional and only used if there is the `cancel` argument set. You can add them right after type of objective.

Example

```
die cancel respawn:100;200;300;world;90;0 events:teleport
```

Fishing: `fish`🔗

Requires the player to catch a fish. It doesn't have to be a fish, it can also be a treasure or junk. The first argument is material *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html)* name of the item to catch, optionally with data value after a colon. Second argument must be amount of fish to catch. You can also add `notify` argument if you want to display progress, optionally with the notification interval after colon.

Fishing has the same properties as mob kill objective.

> **Example**
>
> ```
> fish raw_fish:1 5 notify events:tag_fish_caught
> ```

Interact with entity: `interact`🔗

The player must click on an entity to complete this objective. The first argument is the type of a click. Available values are `right`, `left` and `any`. Second required argument is the mob type *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/EntityType.html)*. Next is an amount of mobs required to click on. These must be unique, so the player can't simply click twenty times on the same zombie to complete it. There is also an optional `name:` parameter which specifies what name the entity must have and `marked:` if the clicked entity needs to be marked by the `spawn` event (see its description for marking explanation). You can also add `notify` argument to make the objective notify players whenever they click a correct entity, optionally with the notification interval after colon and `cancel` if the click shouldn't do what it usually does (i.e. left click won't hurt the entity). This can be limited with an optional `loc` and `range` attribute to limit within a range of a location.

> **Example**
>
> ```
> interact right creeper 1 marked:sick condition:syringeInHand cancel
> ```

Kill player: `kill`🔗

To complete this objective the player needs to kill another player. The first argument is amount of players to kill. You can also specify additional arguments: `name:` followed by the name will only accept killing players with this name, `required:` followed by a list of conditions separated with commas will only accept killing players meeting these conditions and `notify` will display notifications when a player is killed, optionally with the notification interval after colon.

```
kill 5 required:team_B
```

Location: `location`🔗

This objective completes when player moves in specified range of specified location and meets all conditions. The first argument after objective's name must be location, the second - radius around the location. It can be a variable.

Location objective contains one property, `location`. It's a string formatted like `X: 100, Y: 200, Z:300`.

Example

```
location 100;200;300;world 5 condition:test1,!test2 events:test1,test2
```

Logout: `logout`🔗

To complete this objective the player simply needs to leave the server. Keep in mind that running `folder` event here will make it run in "persistent" mode, since the player is offline on the next tick.

Example

```
logout events:delete_objective
```

Password: `password`🔗

This objective requires the player to write a certain password in chat. All attempts of a player will be hidden from public chat. The password consists of a prefix followed by the actual secret word:

```
Solution: The Cake is a lie!
^prefix   ^secret word(s)
```

The objective's instruction string is defined as follows:

1. The first argument is the password, use underscore characters (_) instead of spaces. The password is a regular expression *(https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285)*. They are a little complicated but worth the effort if you want more control over what exactly matches. Websites like regex101.com *(https://regex101.com/)* help with that complexity though. The offical documentation *(https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/regex/Pattern.html#sum)* for regular expressions in Java might also help you. If you don't want to get into them just write down the password but keep in mind that the players answer needs to be an exact match!

2. The prefix can be changed: The default (when no prefix is set) is the translated prefix from the *messages.yml* config in the user's language.
   Note that every custom prefix is suffixed with `:` , so `prefix:Library_password` will require the user to enter `Library password: myfancypassword`.
   To disable the prefix use an empty `prefix:` declaration, e.g. `password myfancypassword prefix: events:success`. Be aware of these side effects that come with disabling the prefix:

   - Nothing will be hidden on failure, so tries will be visible in chat and commands will get executed!
   - If a command was used to enter the password, the command will not be canceled on success and thus still be executed!
   - This ensures that even if your password is `quest` you can still execute the `/quest` command.

3. You can also add the `ignoreCase` argument if you want a password's capitalization to be ignored. This is especially important for regex matching.

4. If you want to trigger one or more events when the player failed to guess the password you can use the argument `fail` with a list of events (comma separated). With disabled prefix every command or chat message will trigger these events!

Example

```
password beton ignoreCase prefix:secret fail:failEvent1,failEvent2 events:message,
```

Pickup item: `pickup`🗒

To complete this objective you need to pickup the specified amount of items. The first argument must be the internal name of an item defined in `items.yml`. This can also be a comma-separated list of multiple items. You can optionally add the `amount:` argument to specify how

many of these items the player needs to pickup. This amount is a total amount though, it does not count per each individual item.

You can also add the `notify` keyword to display how many items are left to pickup.

> **Example**
>
> ```
> pickup emerald amount:3 events:reward notify
> pickup emerald,diamond amount:6 events:reward notify
> ```

This objective has two variable properties: `amount` (shows how much the player already collected) and `left` (displays how much is left to pickup).

Mob Kill: `mobkill`🔗

The player must kill specified amount of mobs You must specify mob type first and then amount. You can find possible mob types here: mob types *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/EntityType.html)*. Additionally you can specify names for mobs with `name:Uber_Zombie`, so only killing properly named mobs counts. All _ are replaced with spaces, so in this example you would have to kill 5 zombies with "Uber Zombie" above their heads. You can also specify `notify` keyword to display messages to the player each time he kills a mob, optionally with the notification interval after colon. If you want to accept only mobs marked with `spawn` event, use `marked:` argument followed by the keyword used in that event.

This objective also has two properties, `amount` and `left`. Amount is current amount of killed mobs, left is amount needed to complete the objective.

> **Example**
>
> ```
> mobkill ZOMBIE 5 name:Uber_Zombie conditions:night
> ```

Potion brewing: `brew`🔗

To complete this objective the player needs to brew specified amount of specified potions. The first argument is a potion ID from *items.yml*. Second argument is amount of potions. You can optionally add `notify` argument to make the objective display progress to players, optionally with the notification interval after a colon.

The brewing will be accepted if the player was the last one to click the ingredient slot in the brewing stand and there were no matching potions there already.

Potion objective has `amount` and `left:` properties.

```
brew weird_concoction 4 event:add_tag
```

Sheep shearing: `shear`🔗

To complete this objective the player has to shear specified amount of sheep, optionally with specified color and/or name. The first, required argument is amount (integer). Optionally, you can add `name:` argument with the name and `color:` with *color name (https://hub.spigotmc.org/javadocs/spigot/org/bukkit/DyeColor.html)*.

Sheep shearing has the same properties as mob kill objective.

```
shear 1 name:Bob color:black
```

Smelting: `smelt`🔗

To complete this objective player must smelt specified item. Note that you must define item as output from furnace, not the ingredient. This one does not support data values (it doesn't have to). First argument is material name. Next is amount (integer).

Smelting has the same properties as crafting objective.

```
smelt IRON_INGOT 5 events:reward
```

Step on pressure plate: `step`🔗

To complete this objective the player has to step on pressure plate at given location. The type of plate does not matter. The first and only required argument is a location. If the pressure plate is not present at that location, the objective will not be completable and will log errors in the console.

Step objective contains one property, `location`. It's a string formatted like `X: 100, Y: 200, Z:300`. It shows an exact location of the pressure plate.

Example

```
step 100;200;300;world events:done
```

Taming: tame🔗

To complete this objective player must tame some amount of mobs. First argument is type, second is amount. The mob must be tameable for the objective to be valid, e.g. on 1.16.1: CAT, DONKEY, HORSE, LLAMA, PARROT or WOLF

Taming has the same properties as mob kill objective.

Example

```
tame WOLF 2 events:wolfs_tamed
```

Variable: variable🔗

This objective is different. You cannot complete it, it will also ignore defined events and conditions. You can start it and that's it. While this objective is active though, everything the player types in chat (and matches special pattern) will become a variable. The pattern is `key: value`. So if you type that, it will create a variable called `key`, which will resolve to `value` string. These are not global variables, you can access them as objective properties. Let's say you defined this objective as `var` in your *objectives.yml* file. You can access the variable in any conversation, event or condition with `%objective.var.key%` - and in case of this example, it will resolve to `value`. The player can type something else, and the variable will change its value. Variables are per-player, so my `key` variable will be different from your `key` variable, depending on what we were typing in chat. You can have as much variables here as you want. To remove this objective use `objective delete` event - there is no other way.

You can also use `variable` event to change variables stored in this objective. There is one optional argument, `no-chat`. If you use it, the objective won't be modified by what players type in chat.

Example

```
variable
```

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/Conditions-List.md)

Conditions List🔗

Advancement: `advancement`🔗

> **Example**
>
> ```
> advancement minecraft:adventure/kill_a_mob
> ```

Conjunction: `and`🔗

Conjunction of specified conditions. This means that every condition has to be met in order for conjunction to be true. Used only in complex alternatives, because conditions generally work as conjunction. Instruction string is exactly the same as in `alternative`.

> **Example**
>
> ```
> and has_helmet,has_chestplate,has_leggings,has_boots
> ```

Armor: `armor`🔗

The armor condition requires the player to wear specified armor, as an item defined in *s.yml* file.

> **Example**
>
> ```
> armor helmet_of_concrete
> ```

Biome: `biome`🔗

This condition will check if the player is in specified biome. The only argument is the biome type *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/block/Biome.html)*.

> **Example**
>
>

```
biome savanna_rock
```

Check conditions: `check`🔗

This condition allow for specifying multiple instruction strings in one, longer string. Each instruction must be started with ^ character and no other dividers should be used. The condition will be met if all inner conditions are met. It's not the same as `and` condition, because you can specify an instruction string, not a condition name.

Example

```
check ^tag beton ^item emerald:5 ^location 100;200;300;survival_nether;5 ^experien
```

Chest Item: `chestitem`🔗

persistent, static

This condition works in the same way as `item` condition, but it checks the specified chest instead of a player. The first argument is a location of the chest and the second one is the list of items defined in the same way as in `item` condition. If there is no chest at specified location the condition won't be met.

Example

```
chestitem 100;200;300;world emerald:5,sword
```

Conversation: `conversation`🔗

This condition will check if a conversation has an available starting option. If no starting option has a condition that returns true then this will return false.

Example

```
conversation innkeeper
```

Day of week: `dayofweek`🔗

It must be a specific day of the week that this condition returns true. You can specify either the english name of the day or the number of the day (1 being monday, 7 sunday,..).

Example

```
dayofweek sunday
```

Potion Effect: `effect`🔗

To meet this condition the player must have an active potion effect. There is only one argument and it takes values from this page: potion types *(https://hub.spigotmc.org/javadocs/spigot/org/ bukkit/potion/PotionEffectType.html)*.

Example

```
effect SPEED
```

Empty inventory slots: `empty`🔗

To meet this condition the players inventory must have the specified amount of empty slots or more. If you want to check for a specific amount (for example for a full inventory with 0 empty slots) you can append the `equal` argument.

Example

```
empty 5
```

Entities in area: `entities`🔗

persistent, static

This condition will return true only if there is a specified amount (or more) of specified entities in the specified area. There are three required arguments - entity type, location and range. Entities are defined as a list separated by commas. Each entity type (taken from here *(https:// hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/EntityType.html)*) can have an additional amount suffix, for example `ZOMBIE:5,SKELETON:2` means 5 or more zombies and 2 or more

skeletons. The location is defined as usual. The number after the location is the range around the location in which will be checked for these entities. You can also specify additional `name:` argument, with the name of the required entity. Replace all spaces with _ here. You can use `marked:` argument to check only for entities marked in `spawn` event.

Example

```
entities ZOMBIE:2 100;200;300;world 10 name:Deamon
```

Experience: `experience`🔗

This condition is met when the player has the specified amount of experience points. You can check for whole levels by adding the `level` argument.

Example

```
experience 30 level
```

Facing direction: `facing`🔗

Checks if the player is looking in the given direction. Valid directions are UP, DOWN, NORTH, EAST, WEST and SOUTH. Up and down start at a pitch of 60°.

Example

```
facing EAST
```

Fly: `fly`🔗

This will check if the player is currently flying (Elytra type of flight).

Example

```
fly
```

Game mode: `gamemode`🔗

This condition checks if the player is in a specified game mode. The first argument is the game mode, i.e. survival, creative, adventure.

Example

```
gamemode survival
```

Global point: `globalpoint`🔗

persistent, static

The same as point condition but it checks the amount for a global point category which has the same value for all players.

Example

```
globalpoint global_knownusers 100
```

Global tag: `globaltag`🔗

persistent, static

This requires a specific global tag to be set and works the same as normal tag condition.

Example

```
globaltag global_areNPCsAgressive
```

Item in Hand: `hand`🔗

This event is met only when player is holding a specified item in his hand. Amount cannot be set here, though it may be checked with `item` condition.

Example

```
hand sword
```

Health: health🔗

Requires the player to have equal or more health than specified amount. The only argument is a number (double). Players can have 0 to 20 health by default (there are some plugins and commands which change the maximum) (0 means dead, don't use that since it will only be met when the player sees the red respawn screen).

Example

```
health 5.6
```

Height: height🔗

This condition requires the player to be *below* specific Y height. The required argument is a number or a location (for example 100;200;300;world). In case of location it will take the height from it and use it as regular height.

Example

```
height 16
```

Item in Inventory: item🔗

This condition requires the player to have all specified items in his inventory or backpack. You specify items in a list separated by commas (without spaces between!) Each item consists of its name and amount, separated by a colon. Amount is optional, so if you specify just item's name the plugin will assume there should be only one item.

Example

```
item emerald:5,gold:10`
```

Journal entry: `journal`🔗

This condition will return true if the player has specified entry in his journal (internal name of the entry, like in *journal.yml*). The only argument is name of the entry.

> **Example**
>
> ```
> journal wood_started
> ```

Location: `location`🔗

It returns true only when the player is closer to specified location than the specified distance. Just two mandatory attributes - location and radius around it (can be a variable).

> **Example**
>
> ```
> location 100;200;300;survival_nether 5
> ```

Looking at a block: `looking`🔗

Checks if the player is looking at a block with the given location or material. You must specify either `loc:` optional (the location of the block) or `type:` optional as a `block selector`. You can also specify both.

> **Example**
>
> ```
> looking loc:12.0;14.0;-15.0;world type:STONE
> ```

Moon Cycle: `mooncycle`🔗

This condition checks the players moon cycle (1 is full moon, 8 is Waxing Gibbous) and returns if the player is under that moon. A list of phases can be found here *(https:// minecraft.gamepedia.com/Moon)*.

> **Example**
>
> ```
> mooncycle 1
> ```

Objective: `objective`🔗

This conditions is very simple: it's true only when the player has an active objective. The only argument is the name of the objective, as defined in *objectives.yml*.

> **Example**
>
> ```
> objective wood
> ```

Alternative: `or`🔗

Alternative of specified conditions. This means that only one of conditions has to be met in order for alternative to be true. You just define one mandatory argument, condition names separated by commas. `!` prefix works as always.

> **Example**
>
> ```
> or night,rain,!has_armor
> ```

Partial date: `partialdate`🔗

The current date must match the given pattern. You can specify the day of the month, the month or the year it must be that this condition returns true or combine them. You can also specify multiple days/months/years by just separating them by `,` or a interval by using `-`. If you have trouble understanding how this works have a look at the example.

The example is true between the 1st and the 5th or on the 20th of each month, but only in the year 2017.

> **Example**
>
> ```
> partialdate day:1-5,20 year:2017
> ```

Party: `party`🔗

To see details about parties read "Party" chapter in Reference section. This condition takes three optional arguments: `every:`, `any:` and `count:`. "Every" is a list of conditions that must be met by every player in the party. Any is a list of conditions that must be met by at least one player in a

party (it doesn't have to be the same player, one can meet first condition, another one can meet the rest and it will work). Count is just a number, minimal amount of players in the party. You don't have to specify all those arguments, you can use only one if you want.

> **Example**
>
> ```
> party 10 has_tag1,!has_tag2 every:some_item any:some_location,some_other_item count
> ```

Permission: `permission`🔗

The player must have a specified permission for this condition to be met. The instruction string must contain permission node as the required argument.

> **Example**
>
> ```
> permission essentials.tpa
> ```

Point: `point`🔗

Requires the player to have amount of points equal to the specified category or more. There are two required arguments, first is the category (string), second is the amount (integer). You can also add optional argument `equal` to accept only players with exactly equal amount of points.

> **Example**
>
> ```
> point beton 20
> ```

Random: `random`🔗

persistent, static

This condition is met randomly. There is one argument: two positive numbers like `5-12`. They mean something like that: "It will be true 5 times out of 12".

> **Example**
>
> ```
> random 12-100
> ```

Armor Rating: `rating`🔗

This one requires the player to wear armor which gives him specified amount of protection (armor icons). The first and only argument should be an integer. One armor point is equal to half armor icon in-game (10 means half of the bar filled).

Example

```
rating 10
```

Real time: `realtime`🔗

There must a specific (real) time for this condition to return true. You need to specify two times (formatted like `hh:mm`) separated by dash. If the first is before the second the time must be between these two, if its after the second the time must be later than the first and earlier than the second to return true.

Example

```
realtime 8:00-12:30
```

Scoreboard: `score`🔗

With this condition you can check if the score in a specified objective on a scoreboard is greater or equal to specified amount. The first argument is the name of the objective, second one is amount (an integer).

Example

```
score kills 20
```

Sneaking: `sneak`🔗

Sneak condition is only true when the player is sneaking. This would probably be useful for creating traps, I'm not sure. There are no arguments for this one.

Example

```
sneak
```

Tag: `tag`🔗

This one requires the player to have a specified tag. Together with ! negation it is one of the most powerful tools when creating conversations. The instruction string must contain tag name.

Example

```
tag quest_completed
```

Test for block: `testforblock`🔗

persistent, static

This condition is met if the block at specified location matches the given material. First argument is a location, and the second one is a `block selector`. The older `data:value` parameter is still supported but now deprecated.

Example

```
testforblock 100;200;300;world STONE:1
```

Time: `time`🔗

There must be specific (Minecraft) time on the player's world for this condition to return true. You need to specify two hour numbers separated by dash. These number are normal 24-hour format hours. The first must be smaller than the second. If you want to achieve time period between 23 and 2 you need to negate the condition.

Example

```
time 2-23
```

Variable: `variable`🔗

This condition checks if a variable value matches given pattern *(https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html)*. The first argument is a variable (with % characters). Second one is the pattern (for example, if you want to check if it's "word", the patter would simply be `word`, but if you want to check if it's a number (positive or negative) you would use `-?\d+` pattern - `-?` means a dash or no dash, `\d` means any digit and + allows that digit to be repeated one or more times).

Example

```
variable %objective.var.price% -?\d+
```

Weather: `weather`🔗

There must be a specific weather for this condition to return true. There are three possible options: sun, rain and storm. Note that `/toggledownfall` does not change the weather, it just does what the name suggests: toggles downfall. The rain toggled off will still be considered as rain! Use `/weather clear` instead.

Example

```
weather sun
```

World: `world`🔗

This conditions checks if the player is in a specified world. The first argument is the name of a world.

Example

```
world world
```

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/Events-List.md)

Events List🔗

Cancel quest: `cancel`🔗

This event works in the same way as a quest canceler in the backpack. Running it is equal to the player clicking on the bone. The only argument is a name of a quest canceler, as defined in *main.yml*

Example

```
cancel wood
```

Chat player message `chat`🔗

This event will send the given message as the player. Therefore, it will look like as if the player did send the message. The instruction string is the command, without leading slash. You can only use `%player%` as a variable in this event. Additional messages can be defined by separating them with | character. If you want to use a | character in the message use \|.

If a plugin does not work with the sudo / command event you need to use this event.

Example

```
sendMSG: "chat Hello!"
sendMultipleMSGs: "chat Hi %player%|ban %player%|pardon %player%"
sendPluginCommand: "chat /someCommand x y z"
```

Chest Clear: `chestclear`🔗

persistent, static

This event removes all items from a chest at specified location. The only argument is a location.

Example

```
chestclear 100;200;300;world
```

Chest Give: `chestgive`🔗

persistent, static

This works the same as `give` event, but it puts the items in a chest at specified location. The first argument is a location, the second argument is a list of items, like in `give` event. If the chest is full, the items will be dropped on the ground. The chest can be any other block with inventory, i.e. a hopper or a dispenser. BetonQuest will log an error to the console when this event is fired but there is no chest at specified location.

Example

```
chestgive 100;200;300;world emerald:5,sword
```

Chest Take: `chesttake`🔗

persistent, static

This event works the same as `take` event, but it takes items from a chest at specified location. The instruction string is defined in the same way as in `chestgive` event.

Example

```
chesttake 100;200;300;world emerald:5,sword
```

Clear mobs: `clear`🔗

This event removes all specified mobs from the specified area. The first required argument is a list of mobs (taken from here *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/EntityType.html)*) separated by commas. Next is location. After that there is the radius around the location (a positive number or a variable). You can also optionally specify `name:` argument, followed by name which removed mobs must have. You can use `marked:` argument to remove only mobs marked in `spawn` event.

Example

```
clear ZOMBIE,CREEPER 100;200;300;world 10 name:Monster
```

Compass: `compass`🔗

When you run this event, you can add or remove a compass destination for the player. You may also directly set the players's compass destination as well. When a destination is added the player will be able to select a specified location as a target of his compass. To select the target the player must open his backpack and click on the compass icon. The first argument is `add`,`del` or `set`, and second one is the name of the target, as defined in *main.yml*. Note that if you set a target the player will not automatically have it added to their choices.

The destination must be defined in the *main.yml* file in `compass` section. You can specify a name for the target in each language or just give a general name, and optionally add a custom item (from *items.yml*) to be displayed in the backpack. Example of a compass target:

```yaml
compass:
  beton:
    name:
      en: Target
      pl: Cel
    location: 100;200;300;world
    item: scroll
```

Example

```
compass add beton
```

Command: `command`🔗

persistent, static

Runs specified command from the console. The instruction string is the command, without leading slash. You can use variables here, but variables other than `%player%` won't resolve if the event is fired from delayed `folder` and the player is offline now. You can define additional commands by separating them with `|` character. If you want to use a `|` character in the command use `\|`.

Example

```
command kill %player%|ban %player%
```

Conversation: `conversation`🔗

Starts a conversation at location of the player. The only argument is ID of the conversation. This bypasses the conversation permission!

Example

```
conversation village_smith
```

Damage player: `damage`🔗

Damages the player by specified amount of damage. The only argument is a number (can have floating point).

Example

```
damage 20
```

Delete Point: `deletepoint`🔗

persistent, static

Delete the player points in a specified category.

Example

```
deletepoint npc_attitude
```

Door: `door`🔗

persistent, static

This event can open and close doors, trapdoors and fence gates. The syntax is exactly the same as in `lever` event above.

Example

```
door 100;200;300;world off
```

Remove Potion Effect: `deleffect`🔗

Removes the specified potion effects from the player. Use `any` instead of a list of types to remove all potion effects from the player.

Example

```
deleffect ABSORPTION,BLINDNESS
```

Potion Effect: `effect`🔗

Adds a specified potion effect to player. First argument is potion type. You can find all available types here *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/potion/PotionEffectType.html)* . Second is integer defining how long the effect will last in seconds. Third argument, also integer, defines level of the effect (1 means first level). Add a parameter `ambient` to make potion particles appear more invisible (just like beacon effects). To hide particles add a parameter `hidden`. To hide the icon for the effect add `noicon`.

Example

```
effect BLINDNESS ambient icon
```

Explosion: `explosion`🔗

static

Creates an explosion. It can make fire and destroy blocks. You can also define power, so be careful not to blow your server away. Default TNT power is 4, while Wither on creation is 7. First argument can be 0 or 1 and states if explosion will generate fire (like Ghast's fireball). Second is also 0 or 1 but this defines if block will be destroyed or not. Third argument is the power (float number). At the end (4th attribute) there is location.

Example

```
explosion 0 1 4 100;64;-100;survival
```

Folder: `folder`🔗

persistent, static

It's something like a container for multiple events. You can use it to clarify your code. It also features optional delay measured in seconds (you can use ticks or minutes if you add `ticks` or `minutes` argument). It is persistent for events marked as *persistent*, which means that the events will be fired even after the player logs out. Beware though, all conditions are false then the player is offline (even inverted ones), so those events should not be blocked by any conditions! The only required argument is a list of events separated by commas. There are also two optional arguments: `delay:` and `random:`. Delay is a number of seconds and it's optional (leaving it blank is the same as `delay:0`. Random is the amount of events, that will be randomly chosen to fire. If set to `0` or omited, it does nothing (all events will fire).

> **Example**
>
> ```
> folder event1,event2,event3 delay:5 random:1
> ```

Give Items: `give`🔗

Gives the player predefined items. They are specified exactly as in `item` condition - list separated by commas, every item can have amount separated by colon. Default amount is 1. If the player doesn't have required space in the inventory, the items are dropped on the ground, unless they are quest items. Then they will be put into the backpack. You can also specify `notify` keyword to display a simple message to the player about receiving items.

> **Example**
>
> ```
> give emerald:5,emerald_block:9
> ```

Give journal: `givejournal`🔗

This event simply gives the player his journal. It acts the same way as /j command would.

> **Example**
>
> ```
> ```

```
givejournal
```

Global point: `globalpoint`🔗

persistent, static

This works the same way as the normal point event but instead to manipulating the points for a category of a specific player it manipulates points in a global category. These global categories are player independent, so you could for example add a point to such a global category every time a player does a quest and give some special rewards for the 100th player who does the quest.

> **Example**
>
> ```
> globalpoint global_knownusers 1
> ```

Global tag: `globaltag`🔗

persistent, static

Works the same way as a normal tag event, but instead of setting a tag for one player it sets it globaly for all players.

> **Example**
>
> ```
> globaltag add global_areNPCsAgressive
> ```

If else: `if`🔗

This event will check a condition, and based on the outcome it will run the first or second event. The instruction string is `if condition event1 else event2`, where `condition` is a condition ID and `event1` and `event2` are event IDs. `else` keyword is mandatory between events for no practical reason.

> **Example**
>
> ```
> if sun rain else sun
> ```

Journal: `journal`🔗

static

Adds or deletes an entry to/from player's journal. Entries are defined in `journal.yml` The first argument is action (add/del), the second one is name of the entry. You can also use only one argument, `update`, it will simply update the journal without addin any entries. It's useful when you need to update the main page.

> **Example**
>
> ```
> journal add quest_started
> ```

> **Example**
>
> ```
> journal update`
> ```

Kill: `kill`🔗

Kills the player. Nothing else.

Kill Mobs: `killmob`🔗

persistent, static

Kills all mobs of given type at the location. First argument is the type of the mob *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/EntityType.html)*. Next argument is the location. Third argument is the radius around the location, in which the mobs must be to get killed.
You can also specify `name:` argument, followed by the name of the mob which should get killed. All _ characters will be replaced with spaces. If you want to kill only mobs that have been marked using the spawn mob event use `marked:` argument followed by the keyword.

Only mobs that are in loaded chunks can be killed by using this event.

> **Example**
>
> ```
> killmob ZOMBIE 100;200;300;world 40 name:Bolec
> ```

Lever: `lever`🔗

persistent, static

This event can switch a lever. The first argument is a location and the second one is state: `on`, `off` or `toggle`.

> **Example**
>
> ```
> lever 100;200;300;world toggle
> ```

Lightning: `lightning`🔗

static

Strikes a lightning at given location. The only argument is the location.

> **Example**
>
> ```
> lightning 100;64;-100;survival
> ```

Message: `message`🔗

static

This event simply displays a message to the player. The instruction string is the message. All & color codes are respected. You can add additional translations by starting them with `{lang}` argument, just like in the example. The player will see his language or the default one if it's not defined. You can use conversation variables with this event. Just make sure not to use `%npc%`.

> **Example**
>
> ```
> message {en} &4You are banned, %player%! {pl} &4Jestes zbanowany, %player%! {de}&4
> ```

Notification: `notify`🔗

Trigger a notification using the NotifyIO system. The first arguments are the message to send. A comma seperated list can be provided to an optional `category` tag to use a Notification Category. You can optionally set which NotifyIO to use by providing it with an `io` tag. You can

also optionally pass flags in the form of `key:value` to provide custom config to the NotifyIO that will override those by the category used.

Please note that if you don't provide a valid category and haven't defined a default category then you must provide an io flag otherwise the default io (Generally chat) will be used.

Please refer to the Notification chapter for more details.

> **Example**
>
> ```
> notify This is a test category:MyCategory io:bossbar barColor:red sound:BLOCK_CHEST
> ```

Objective: `objective`🔗

persistent, static

Manages the objectives. Syntax is `objective <action> name`, where `<action>` can be *start*/*add* (one of the two), *delete*/*remove* or *complete*/*finish*. Name is the name of the objective, as defined in *objectives.yml*.

> **Example**
>
> ```
> objective start wood
> ```

OPSudo: `opsudo`🔗

This event is similar to `command` event, the only difference is that it will fire a command as the player with temporary OP permissions. Additional commands can be defined by separating them with | character. If you want to use a | character in the message use \|.

> **Example**
>
> ```
> sudo spawn
> ```

Party event: `party`🔗

Runs the specified list of events (third argument) for every player in a party. More info about parties in "Party" chapter in Reference section.

> **Example**
>
> ```
> party 10 has_tag1,!has_tag2 give_reward
> ```

Pick random: `pickrandom`🔗

persistent, static

Another container for events. It picks one (ore multiple) of the given events and runs it (but only if all conditions are true, if not it will do nothing). You must specify how likely it is that each event is picked by adding the percentage before the events id. By default it picks one event from the list but you can add a `amount:` optional if you want more to be picked. Note that only as many events as specified can be picked and `amount:0` will do nothing.

> **Example**
>
> ```
> pickrandom 20.5%event1,0.5%event2,79%event3 amount:2
> ```

Play sound: `playsound`🔗

This event will play a specified sound for the player. The only required argument is the sound name in lower case letters (can take custom values if you're using a resource pack). There are also a few optional arguments. `category:` is the sound category *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/SoundCategory.html)* (if not specified it will use MASTER), `volume:` is a decimal responsible for the sound's volume, `pitch:` specifies the pitch and `location:` makes the sound play at specified location.

> **Example**
>
> ```
> playsound: playsound minecraft:entity.ender_dragon.hurt category:MASTER volume:0.8
> ```

Point: `point`🔗

persistent

Gives the player a specified amount of points in a specified category. Amount can be negative if you want to subtract points. You can also use an asterisk to do multiplication (or division, if you use a fraction). First argument after the event name must be a category, and the second one -

amount of points to give/take/multiply. This event also supports an optional `notify` argument that will display information about the change using the notification system.

Example

```
point npc_attitude 10
```

Example

```
point village_reputation *0.75
```

Run events: run🔗

This event allow for specifying multiple instruction strings in one, longer string. Each instruction must be started with ^ character and no other dividers should be used. It's not the same as `folder` condition, because you can specify an instruction string, not an event name. It is also fired on the same tick, not on the next one like in `folder`. Don't use conditions here, it behaves strangely. I'll fix this in 1.9 version.

Example

```
run ^tag add beton ^give emerald:5 ^entry add beton ^kill
```

Scoreboard: score🔗

This event works in the same way as `point` event, the only difference is that is uses scoreboards instead of points. You can add, subtract, multiply and divide scores in objectives on the scoreboard. The first argument is the name of the objective, second one is a number. It can be positive for additon, negative for subtraction or prefixed with an asterisk for multiplication. Multiplying by fractions is the same as dividing.

Example

```
score kills 1
```

Set Block: `setblock`🔗

persistent, static

Sets a block at given location to specified material. Useful for triggering redstone contraptions. There are two required arguments. First is required, and should be material's name (List of materials *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/Material.html)*). Second is a location and is also required. Last, optional is `data:` with an integer, which defines block's data value. Default is 0.

> **Example**
>
> ```
> setblock REDSTONE_BLOCK 100;200;300;world
> ```

Spawn Mob: `spawn`🔗

persistent, static

Spawns specified amount of mobs of given type at the location. First argument is a location. Next is type of the mob *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/entity/ EntityType.html)*. The last, third argument is integer for amount of mobs to be spawned. You can also specify `name:` argument, followed by the name of the mob. All _ characters will be replaced with spaces. You can also mark the spawned mob with a keyword using `marked:` argument. It won't show anywhere, and you can check for only marked mobs in `mobkill` objective.

You can specify armor which the mob will wear and items it will hold with `h:` (helmet), `c:` (chestplate), `l:` (leggings), `b:` (boots), `m:` (main hand) and `o:` (off hand) optional arguments. These take a single item without amount, as defined in *items.yml*. You can also add a list of drops with `drops:` argument, followed by a list of items with amounts after colons, separated by commas.

> **Example**
>
> ```
> spawn 100;200;300;world SKELETON 5 marked:targets
> ```

> **Example**
>
> ```
> spawn 100;200;300;world ZOMBIE name:Bolec 1 h:blue_hat c:red_vest drops:emerald:10
> ```

Sudo: `sudo`🔗

This event is similar to `command` event, the only difference is that it will fire a command as the player. Additional commands can be defined by separating them with | character. If you want to use a | character in the message use \|.

Example

```
sudo spawn
```

Tag: `tag`🔗

persistent, static

This event adds (or removes) a tag to the player. The first argument after event's name must be `add` or `del`. Next goes the tag name. It can't contain spaces (though _ is fine). Additional tags can be added, separated by commas (without spaces).

Example

```
tag add quest_started,new_entry
```

Take Items: `take`🔗

Removes items from player's inventory or backpack (in that order). If the items aren't quest items don't use `take` event with player options in conversations! The player can drop items before selecting the option and pickup them after the event fires. Validate it on NPC's reaction! Defining instruction string is the same as in give event. You can also specify `notify` keyword to display a simple message to the player about loosing items.

Example

```
take emerald:120,sword
```

Time: `time`🔗

Sets or adds time. The only argument is time to be set (integer) or time to be added (integer prefixed with +), in 24 hours format. Subtracting time is done by adding more time (if you think of this, it actually makes sense). Minutes can be achieved with floating point.

Example

```
time +6
```

Title: `title`🔗

This event displays a title or a subtitle. The first argument is the type (`title` or `subtitle`), second argument are title's duration times (in ticks) separated by semicolons - fade in, stay and fade out: `20;100;20`. If you set it to three zeros (`0;0;0`) the plugin will use default Minecraft values. After these two required arguments there is a title message, formatted like in the `message` event, which supports multiple languages, color codes and variables. Keep in mind that the subtitle will only appear if the title is visible - that's how Minecraft works.

Example

```
title subtitle 0;0;0 {en} Lobby joined! {pl} Dołączono do lobby!
```

Teleport: `teleport`🔗

Teleports the player to a specified location, with or without head rotation. It will also end the conversation, if the player has one active.The first and only argument must be location. It's a good idea to use yaw and pitch here.

Example

```
teleport 123;32;-789;world_the_nether;180;45
```

Variable: `variable`🔗

This event has only one purpose - to change variables stored in `variable` objective. The first argument is the ID of a `variable` objective (if you use any other type you will get an error).

Second one is the key of the variable and the third is the value. Both can use `%...%` variables. Refer to `variable` objective documentation for information about storing variables.

> **Example**
>
> ```
> variable some_var_obj name %player%
> ```

Weather: `weather`🔗

Sets weather. The argument is `sun`, `rain` or `storm`.

> **Example**
>
> ```
> weather rain
> ```

Give experience: `experience`🔗

Gives the specified amount of experience points to the player. You can give whole levels by adding the `level` argument.

> **Example**
>
> ```
> experience 4 level
> ```

Last update:

*(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/
Compatibility.md)*

Compatibility🔗

In total 27 plugins have dedicated support for BetonQuest.

BetonQuest can hook into other plugins to extend its functionality. Currently, 24 plugins are supported:
BetonLangAPI, Brewery, Citizens, Denizen, EffectLib, Heroes, HolographicDisplays, JobsReborn, Magic, mcMMO, MMOLib, MMOCore, MMOItems, MythicMobs, PlaceholderAPI, PlayerPoints, ProtocolLib, Quests, Shopkeepers, SkillAPI, Skript, Vault, WorldEdit and WorldGuard.

Some plugins also hook into BetonQuest and provide support by themselves:
nuNPCDestinations *(https://www.spigotmc.org/resources/npcdestinations-create-living-npcs.
13863/)*, RPGMenu *(https://www.spigotmc.org/resources/rpgmenu.55170/)* , NGVexJournal
*(https://www.spigotmc.org/resources/ngvexjournal-multi-function-betonquest-journal-
gui-1-7-10-1-13.76938/)*

BetonLangAPI🔗

Events🔗

Language: `language`🔗

This event changes player's language to the specified one. There is only one argument, the language name.

> **Example**
>
> ```
> language es
> ```

Brewery🔗

Conditions🔗

Drunk: `drunk`🔗

This condition is true if the player is drunken. Only argument is the minimal drunkness (0-100).

```
1    drunk 50
```

Drunk Quality: `drunkquality`🔗

This condition is true if the player has the given drunk quality. Only argument is the minimal drunk quality (1-10).

```
1    drunkquality 3
```

Has Brew: `hasbrew`🔗

This condition is true if the player has the given grew with the specified amount in his inventory.

```
1    hasbrew MY_BREW 2
```

Events🔗

Give Brew: `givebrew`🔗

Gives the player the specified drink. An amount needs to be specified.

```
1    givebrew MY_BREW 2
```

Take Brew: `takebrew`🔗

Removes the specified drink from the players inventory. An amount needs to be specified.

```
1    takebrew MY_OTHER_BREW 2
```

NPC's with Citizens🔗

If you have this plugin you can use it's NPCs for conversations. I highly recommend you installing it, it's NPCs are way more immersive. Having Citizens also allows you to use NPCKill objective and to have moving NPC's.

> **Notice**
>
> When you use Citizens, in main.yml you need to specify the ID of the NPC instead of the name!

Conditions

NPC distance: `npcdistance`

This condition will return true if the player is closer to the NPC with the given ID than the given distance. The NPCs ID is the first argument, the distance is the second. If the npc is despawned the condition will return false.

Example

```
npcdistance 16 22
```

NPC location: `npclocation`

persistent, static

This condition will return true if a npc is close to a location. First argument is the id of the NPC, second the location and third the maximum distance to the location that the npc is allowed to have.

Example

```
npclocation 16 4.0;14.0;-20.0;world 22
```

NPC region: `npcregion`

persistent, static

Notice

This condition also requires WorldGuard to work.

This condition will return true if a npc is inside a region. First argument is the id of the npc second is the name of the region.

Example

```
npcregion 16 spawn`
```

Events🔗

Move NPC: `movenpc`🔗

This event will make the NPC move to a specified location. It will not return on its own, so you have to set a single path point with */npc path* command - it will then return to that point every time. If you make it move too far away, it will teleport or break, so beware. You can change maximum pathfinding range in Citizens configuration files. The first argument in this event is ID of the NPC to move. Second one is a location in a standard format (like in `teleport` event). You can also specify multiple locations separated by colons to let the npc follow a path of locations. You can also specify additional arguments: `block` will block the NPC so you won't be able to start a conversation with him while he is moving, `wait:` is a number of tick the NPC will wait at its destination before firing events, `done:` is a list of events fired after reaching the destination, `fail:` is a list of events fired if this event fails. Move event can fail if the NPC is already moving for another player.

> **Example**
>
> ```
> movenpc 121 100;200;300;world,105;200;280;world block wait:20 done:msg_were_here,g
> ```

Stop moving NPC: `stopnpc`🔗

This will stop all current move tasks for the npc with the given ID.

> **Example**
>
> ```
> stopnpc 16
> ```

Teleport NPC: `teleportnpc`🔗

This event will teleport the NPC with the given ID to the given location.

> **Example**
>
> ```
> teleportnpc 53 100;200;300;world
> ```

Objectives🔗

NPC Interact: `npcinteract`🔗

The player has to right-click on the NPC with specified ID. It can also optionally cancel the action, so the conversation won't start. The first argument is number (ID of the NPC), and the second is optional `cancel`.

Example

```
npcinteract 3 cancel conditions:sneak events:steal`
```

NPC Kill: `npckill`🔗

NPC Kill objective requires the player to kill an NPC with the given ID. You can also define how many times an NPC has to be killed. Right after objective's name there must be na ID of the NPC. You can also add an amount by `amount:`.

Example

```
npckill 16 amount:3 events:reward`
```

NPC Range: `npcrange`🔗

The player has to enter/leave a circle with the given radius around the NPC to complete this objective. First argument is the ID of the NPC, second one is either `enter` or `leave` and the third one is the range.

Example

```
npcrange 3 enter 20 events:master_inRange
```

Denizen🔗

Events🔗

Script: `script`🔗

With this event you can fire Denizen task scripts. Don't confuse it with `skript` event, these are different. The first and only argument is the name of the script.

> **Example**
>
> ```
> script beton
> ```

EffectLib🔗

If you install this plugin on your server you will be able to set a particle effect on NPCs with conversations and use `particle` event.

You can control the behaviour of particles around the NPCs in *custom.yml* file, in `npc_effects` section. Each effect is defined as a separate subsection and consists of EffectLib options (described on the EffectLib page) and several BetonQuest settings. `npcs` is a list of all NPCs on which this effect can be displayed. `conditions` is a list of conditions the player has to meet in order to see the effect. BetonQuest will find the first effect which can be displayed and show it to the player. `interval` controls how often the effect is displayed (in ticks). The effect will be fired from the exact location of the NPC, upwards.

```yaml
npc_effects:
    check_interval: 50
    disabled: false
    farmer:
        class: VortexEffect
        iterations: 20
        particle: crit_magic
        helixes: 3
        circles: 1
        grow: 0.1
        radius: 0.5
        interval: 30
        npcs:
            - 1
        conditions:
```

```
        - '!con_tag_started'
        - '!con_tag_finished'
```

Events🔗

Particle: `particle`🔗

This event will load an effect defined in `effects` section in *custom.yml* file and display it on player's location. The only argument is the name of the effect. You can optionally add `loc:` argument followed by a location written like `100;200;300;world;180;-90` to put it on that location. If you add `private` argument the effect will only be displayed to the player for which you ran the event.

> **Example**
>
> In custom.yml
>
> ```yaml
> effects:
>   beton:
>     class: HelixEffect
>     iterations: 100
>     particle: smoke
>     helixes: 5
>     circles: 20
>     grow: 3
>     radius: 30
> ```
>
> In events.yml
>
> ```
> particle beton loc:100;200;300;world;180;-90 private`
> ```

Heroes🔗

When you install Heroes, all kills done via this plugin's skills will be counted in MobKill objectives.

Conditions🔗

Class: `heroesclass`🔗

This condition checks the classes of the player. The first argument must be `primary`, `secondary` or `mastered`. Second is the name of a class or `any`. You can optionally specify `level:` argument followed by the required level of the player.

```
heroesclass mastered warrior
```

Skill: `heroesskill`🔗

This condition checks if the player can use specified skill. The first argument is the name of the skill.

```
heroesskill charge`
```

Events🔗

Experience: `heroesexp`🔗

This event simply gives the player specified amount of Heroes experience. The first argument is either `primary` or `secondary` and it means player's class. Second one is the amount of experience to add.

```
heroesexp primary 1000
```

HolographicDisplays🔗

Installing this plugin will enable you to create hidden holograms, which will be shown to players only if they meet specified conditions. Note that you need to have ProtocolLib *(https:// www.spigotmc.org/resources/protocollib.1997/)* installed in order to hide holograms from certain players.

In order to create a hologram, you have to add `holograms` section in your *custom.yml* file. Add a node named as your hologram to this section and define `lines`, `conditions` and `location` subnodes. The fist one should be a list of texts - these will be the lines of a hologram. Color codes are supported. Second is a list of conditions separated by commas. Third is a location in a standard format, like in `teleport` event. An example of such hologram definition:

```yaml
holograms:
  beton:
    lines:
    - '&bThis is Beton.'
    - 'item:custom_item'
    - '&eBeton is strong.'
    location: 100;200;300;world
    conditions: has_some_quest, !finished_some_quest
    # How often to check conditions (optional)
    check_interval: 20
```

A line can also represent a floating item. To do so enter the line as 'item:`custom_item`'. It will be replaced with the `custom_item` defined defined in items.yml. If the Item is defined for example as map, a floating map will be seen between two lines of text.

The holograms are updated every 10 seconds. If you want to make it faster, add `hologram_update_interval` option in *config.yml* file and set it to a number of ticks you want to pass between updates (one second is 20 ticks). Don't set it to 0 or negative numbers, it will result in an error.

If Citizens is also installed then you can have holograms configured relative to an npc. Add the following to *custom.yml*.

```yaml
npc_holograms:
  # How often to check conditions
  check_interval: 100

  # Holograms follow npcs when they move (higher cpu usage when true
  follow: false

  # Disable npc_holograms
  disabled: false

  # Hologram Settings
  default:
    # Lines in hologram
    lines:
      - "Some text!"
    # Vector offset to NPC position to place hologram
    vector: 0;3;0

    # Conditions to display hologram
    conditions: has_some_quest, !finished_some_quest

    # NPC's to apply these settings to. If blank, applies by default
```

```
    npcs:
      - 0
      - 22
```

Item lines are also supported here.

JobsReborn🔗

Requires adding the following to *config.yml*:

```
hook:
  jobs: 'true'
```

Conditions🔗

Can Level up: `nujobs_canlevel {jobname}`🔗

Returns true if the player can level up

Has Job: `nujobs_hasjob {jobname}`🔗

Returns true if the player has this job

> **Example**
>
> ```
> nujobs_hasjob Woodcutter
> ```

Job Full: `nujobs_jobfull {jobname}`🔗

Returns true if the job is at the maximum slots

Job Level: `nujobs_joblevel {jobname} {min} {max}`🔗

Returns true if the player has this job, and at a level equal to or between the min/max

> **Example**
>
> ```
> nujobs_joblevel Woodcutter 5 10
> ```

Events

Add Experience: `nujobs_addexp {jobname} {exp}`

Gives the player experience

Increase Level: `nujobs_addlevel {jobname} {amount}`

Increases the player level by amount.

Decrease Level: `nujobs_dellevel {jobname} {amount}`

Decreases the players level by amount.

Join Job: `nujobs_joinjob {jobname}`

Joins the player to job.

Leave Job: `nujobs_leavejob {jobname}`

Removes the player from job.

Set Level: `nujobs_setlevel {jobname} {level}`

Set the player to level.

Objectives

Join Job: `nujobs_joinjob {jobname}`

Triggers when player joins job.

Leave Job: `nujobs_leavejob {jobname}`

Triggers when player leaves job.

> **Notice**
>
> This is not triggered by '/jobs leaveall'

Job Levelup: `nujobs_levelup {jobname}`

Triggers when player levels up.

Job Payment: `nujobs_payment {amount}`🔗

Triggers when player makes {amount} of money from jobs.

Magic🔗

Conditions🔗

Wand: `wand`🔗

This condition can check wands. The first argument is either `hand`, `inventory` or `lost`. If you choose `lost`, the condition will check if the player has lost a wand. If you choose `hand`, the condition will check if you're holding a wand in your hand. `inventory` will check your whole inventory instead of just the hand. In case of `hand` and `inventory` arguments you can also add optional `name:` argument followed by the name of the wand (as defined in *wands.yml* in Magic plugin) to check if it's the specific type of the wand. In the case of `inventory` you can specify an amount with `amount` and this will only return true if a player has that amount. You can also use optional `spells:` argument, followed by a list of spells separated with a comma. Each spell in this list can have defined minimal level required, after a colon.

> **Example**
>
> ```
> wand hand name:master spells:flare,missile:2
> ```

McMMO🔗

Conditions🔗

Level: `mcmmolevel`🔗

This conditions checks if the player has high enough level in the specified skill. The first argument is the name of the skill, second one is the minimum level the player needs to have to pass this condition.

> **Example**
>
> ```
> mcmmolevel woodcutting 50
> ```

Events🔗

Experience: `mcmmoexp`🔗

This event adds experience points in a specified skill. The first argument is the name of the skill, second one is the amount of experience to add.

> **Example**
>
> ```
> mcmmoexp swords 1500
> ```

MMOGroup (MMOCore, MMOItem, MMOLib)🔗

Conditions🔗

MMOCore class: `mmoclass`🔗

Checks if a player has the given MMOCore class. If a level has been specified the player needs to be on that level or higher to meet the condition. You can disable this behaviour by adding the `equal` argument.

```
1   mmoclass WARRIOR
2   mmoclass MAGE 5
3   mmoclass MAGE 5 equal
```

MMOCore attribute: `mmoattribute`🔗

Checks if a player has the specified attribute on the given level or higher. You can disable this behaviour by adding the `equal` argument.

```
1   mmoclass mmoattribute strength 2
2   mmoclass mmoattribute strength 2 equal
```

MMOCore profession: `mmoprofession`🔗

Checks if a player has the specified profession on the given level or higher. You can disable this behaviour by adding the `equal` argument.

```
1    mmoprofession mining 2
2    mmoprofession mining 2 equal
```

MMOItems item: `mmoitem`🔗

Checks if a player has the specified MMOItem in his inventory. If no amount has been defined the default amount is one.

```
1    mmoitem ARMOR SKELETON_CROWN
2    mmoitem GEMS SPEED_GEM 3
```

MMOItems hand: `mmohand`🔗

Checks if a player holds the specified MMOItem in his hand. Checks the main hand if not specified otherwise using the `offhand` argument. If no amount has been defined the default amount is one.

```
1    mmohand ARMOR SKELETON_CROWN
2    mmohand GEMS SPEED_GEM 3 offhand
```

MMOLib stat: `mmostat`🔗

Checks these *(https://github.com/mmopluginteam/mmolib/blob/master/src/main/java/net/mmogroup/mmolib/api/stat/SharedStat.java)* stats that combine all sorts of stats from MMOCore and MMOItems. The player needs to be on the specified level or higher in order to meet this condition. You can disable this behaviour by adding the `equal` argument.

```
1    mmostat DAMAGE_REDUCTION 3
```

MythicMobs🔗

Having MythicMobs allows you to use MythicMobs MobKill objective and MythicMobs SpawnMob event.

Objectives🔗

MobKill: `mmobkill`🔗

To complete this objective you need to kill specified amount of MythicMobs. The first argument must be the mob's internal name (the one defined in MythicMobs' configuration). You can optionally add `amount:` argument to specify how many of these mobs the player needs to kill. You can also add "notify" keyword if you want to display to players the amount of mobs left to kill.

Example

```
mmobkill SkeletalKnight amount:2 events:reward
```

Conditions🔗

MythicMob distance: `mythicmobdistance`🔗

Events🔗

Check whether the player is near a specific MythicMobs entity. The first argument is the internal name of the mob (the one defined in MythicMobs' configuration). The second argument is the distance to check, measured in block lengths in a circular radius.

Example

```
mythicmobdistance SkeletalKnight 7
```

SpawnMob: `mspawnmob`🔗

Spawn specified amount of MythicMobs at given location. The first argument is a location defined like `100;200;300;world`. Second is MythicMobs internal name (the one defined in MythicMobs' configuration) followed by a colon and a level. Third one is amount and it's required!

Example

```
mspawnmob 100;200;300;world SkeletalKnight:1 5
```

PlaceholderAPI🔗

If you have this plugin, BetonQuest will add a `betonquest` placeholder to it and you will be able to use ph variable in your conversations.

Placeholder: `betonquest`🔗

In any other plugin which uses PlaceholderAPI you can use BetonQuest variables with `%betonquest_package:variable%` placeholder. The `package:` part is the name of a package. If you skip this, the plugin will assume you're using that variable in `default` package. The `variable` part is just a BetonQuest variable without percentage characters, like `point.beton.amount`.

> **Example**
>
> ```
> %betonquest_someGreatQuest:objective.killZombies.left%
> ```

Variable: ph🔗

You can also use placeholders from other plugins in BetonQuest. Simply insert a variable starting with ph, the second argument should be the placeholder without percentage characters.

> **Example**
>
> ```
> %ph.player_item_in_hand%
> ```

PlayerPoints🔗

Conditions🔗

PlayerPoints: `playerpoints`🔗

This condition simply checks if the player has specified amount of points in the PlayerPoints plugin. The only argument is a number.

> **Example**
>
> ```
> playerpoints 100
> ```

Events🔗

PlayerPoints: `playerpoints`🔗

This event simply adds, removes or multiplies points in the PlayerPoints plugin. The only argument is a number, it can be positive, negative or prefixed with an asterisk for multiplication.

> **Example**
>
> ```
> playerpoints *2
> ```

ProtocolLib🔗

Hiding NPC's🔗

Having ProtocolLib installed will let you hide Citizens NPCs if specified conditions are met. You can do that by adding `hide_npcs` section to *custom.yml* file in your package. There you can assign conditions to specific NPC IDs:

```
hide_npcs:
  41: killedAlready,questStarted
  127: '!questStarted'
```

Conversation IO: `menu`🔗

A conversation IO that makes use of a chat menu system. A video of it in action can be seen here *(https://www.youtube.com/channel/UCyF806Xfzr4B18dsZ4TEI9w)*.

Customize how it looks by adding the following lines to custom.yml:

```
menu_conv_io:
  line_length: 50
  refresh_delay: 180

  npc_wrap: '&l &r'
  npc_text: '&l &r&f{npc_text}'
  npc_text_reset: '&f'
  option_wrap: '&r&l &l &l &l &r'
  option_text: '&l &l &l &l &r&8[ &b{option_text}&8 ]'
  option_text_reset: '&b'
  option_selected: '&l &r &r&7»&r &8[ &f&n{option_text}&8 ]'
```

```
    option_selected_reset: '&f'
    option_selected_wrap: '&r&l &l &l &l &r&f&n'

    control_select: jump,left_click
    control_cancel: sneak
    control_move: scroll,move

    npc_name_type: chat
    npc_name_align: center
    npc_name_format: '&e{npc_name}&r'
```

Where:

- `line_length` - Maximum size of a line till its wrapped
- `refresh_delay` - Specify how many ticks to auto update display. Default 180
- `npc_wrap` - What text to prefix each new line in the NPC text that wraps
- `npc_text` - How to write the NPC text. Replaces {1} with the npcs text
- `npc_text_reset` - When a color reset is found, what to replace it with
- `option_wrap` - What text to prefix each new line in an option that wraps
- `option_text` - How to write an option. Replaces {1} with the option text
- `option_text_reset` - When a color reset is found, what to replace it with
- `option_selected` - How to write a selected option. Replaces {1} with the option text
- `option_selected_reset` - When a color reset is found, what to replace it with
- `option_selected_wrap` - What text to prefix each new line in a selected option that wraps
- `control_select` - Space separated actions to select. Can be any of 'jump', 'left_click', 'sneak'
- `control_cancel` - Space separated actions to select. Can be any of 'jump', 'left_click', 'sneak'
- `control_move` - Space separated actions to move selection. Can be any of 'move', 'scroll'
- `npc_name_type` - Type of NPC name display. Can be one of: 'none', 'chat'
- `npc_name_align` - For npc name type of 'chat', how to align name. One of: 'left', 'right', 'center'
- `npc_name_format` - How to format the npc name

Variables:

- `{npc_text}` - The text the NPC says
- `{option_text}` - The option text
- `{npc_name}` - The name of the NPC

Chat Interceptor: `packet`🔗

Intercept pretty much anything sent to the player by intercepting packets sent to them. This can be enabled by default by setting the `default_interceptor` to `packet` in config.yml or per conversation by setting `interceptor` to `packet` in the top level of the conversation.

Quests🔗

Quests is another questing plugin, which offers very simple creation of quests. If you don't want to spend a lot of time to write advanced quests in BetonQuest but you need a specific thing from this plugin you can use Custom Event Reward or Custom Condition Requirement. Alternatively, if you have a lot of quests written in Quests, but want to integrate them with the conversation system, you can use `quest` event and `quest` condition.

Condition Requirement (Quests)🔗

When adding requirements to a quest, choose "Custom requirement" and then select "BetonQuest condition". Now specify condition's name and it's package (like `package.conditionName`). Quests will check BetonQuest condition when starting the quest.

Event Reward (Quests)🔗

When adding rewards to a quest or a stage, choose "Custom reward" and then select "BetonQuest event". Now specify event's name and it's package (like `package.eventName`). Quests will fire BetonQuest event when this reward will run.

Conditions🔗

Quest condition: `quest`🔗

This condition is met when the player has completed the specified quest. The first and only argument is the name of the quest. It it contains any spaces replace them with _.

> **Example**
>
> ```
> quest stone_miner
> ```

Events🔗

Quest: `quest`🔗

This event will start the quest for the player. The first argument must be the name of the quest, as defined in `name` option in the quest. If the name contains any spaces replace them with `_`. You can optionally add `check-requirements` argument if you want the event to respect this quest's requirements (otherwise the quest will be forced to be started).

Example

```
quest stone_miner check-requirements
```

Shopkeepers🔗

Conditions🔗

Shop amount: `shopamount`🔗

This condition checks if the player owns specified (or greater) amount of shops. It doesn't matter what type these shops are. The only argument is a number - minimum amount of shops.

Example

```
shopamount 2
```

Events🔗

Open shop window: `shopkeeper`🔗

This event opens a trading window of a Villager. The only argument is the uniqueID of the shop. You can find it in *Shopkeepers/saves.yml* file, under `uniqueID` option.

Example

```
shopkeeper b687538e-14ce-4b77-ae9f-e83b12f0b929
```

SkillAPI🔗

Conditions🔗

Class: `skillapiclass`🔗

This condition checks if the player has specified class or a child class of the specified one. The first argument is simply the name of a class. You can add `exact` argument if you want to check for that exact class, without checking child classes.

> **Example**
>
> ```
> skillapiclass warrior
> ```

Level: `skillapilevel`🔗

This condition checks if the player has specified or greater level is the specified class. The first argument is class name, the second one is the required level.

> **Example**
>
> ```
> skillapilevel warrior 3
> ```

Skript🔗

BetonQuest can also hook into Skript. Firstly, to avoid any confusion, I will refere to everything here by name of the plugin (Skript event is something else than BetonQuest event). Having Skript on your server will enable using BetonQuest events and conditions in scripts, and also trigger them by BetonQuest event.

Skript event triggered by BetonQuest `skript` event🔗

This entry will describe two things: Skript event and BetonQuest event.

1. Skript event - `on [betonquest] event "id"` - this is the line you use in your scripts to trigger the code. `betonquest` part is optional, and `id` is just some string, which must be equal to the one you specified in BetonQuest event.
2. BetonQuest event - `skript` - this event will trigger the above Skript event in your scripts. The instruction string accepts only one argument, id of the event. It have to be the same as the one defined in Skript event for it to be triggered.

Example

In your script

```
on betonquest event "concrete":
```

In events.yml

```
fire_concrete_script: skript concrete
```

Skript condition🔗

You can check BetonQuest conditions in your scripts by using the syntax `player meets [betonquest] condition "id"`. `betonquest` is optional, and `id` is the name of the condition, as defined in *conditions.yml*.

Example

In your script

```
player meets condition "has_ore"
```

In conditions.yml

```
has_ore: item iron_ore:5
```

Skript event🔗

You can also fire BetonQuest events with scripts. The syntax for Skript effect is `fire [betonquest] event "id" for player`. Everything else works just like in condition above.

Example

In your script

```
fire event "give_emeralds" for player
```

In events.yml

```
give_emeralds: give emerald:5
```

Vault🔗

By installing Vault you enable Permission event and Money condition/event.

Conditions🔗

Money: `money`🔗

Checks if the player has specified amount of money. You can specify only one argument, amount integer. It cannot be negative!

Example

```
money 500
```

Events🔗

Money: `money`🔗

Deposits, withdraws or multiplies money on player's account. There is only one argument, amount of money to modify. It can be positive, negative or start with an asterisk for multiplication.

Example

```
money -100
```

Permission: `permission`🔗

Adds or removes a permission or a group. First argument is `add` or `remove`. It's self-explanatory. Second is `perm` or `group`. It also shouldn't be hard to figure out. Next thing is actual string you want to add/remove. At the end you can also specify world in which you want these permissions. If the world name is ommited then permission/group will be global.

Example

```
permission remove group bandit world_nether
```

Variables

Money: `money`

There is only one argument in this variable, `amount` for showing money amount or `left:` followed by a number for showing the difference between it and amount of money.

> **Example**
>
> ```
> %money.left:500%
> ```

WorldEdit

Events

Paste schematic: `paste`

This event will paste a schematic at the given location. The first argument is a location and the second one is the name of schematic file. The file must be located in `WorldEdit/schematics` and have a name like `some_building.schematic`. An optional `noair` can be added to paste ignoring air blocks. If you have only a `.schem` schematic, simply append `.schem` to the schematic name.

> **Example**
>
> ```
> paste 100;200;300;world some_building noair
> ```

WorldGuard

Conditions

Inside Region: `region`

This condition is met when the player is inside the specified region. The only argument is the name of the region.

> **Example**
>

```
region beton
```

Objectives􀀁􀀁

Enter Region: `region`􀀁􀀁

To complete this objective you need to enter WorldGuard region with specified name. A required
argument is the name of the region and you may also pass an optional `entry` and/or `exit` to
only trigger when entering or exiting a region instead of anytime inside a region.

> **Example**
>
> ```
> region beton events:kill
> ```

Last update:

Variables List🔗

Custom Strings🔗

It is possible to save multiple custom strings per player. This works by using the `variable` objective and the `variable` event.

Global point: `globalpoint`🔗

Works the same as normal point variable but instead of displaying points from a players category it displays points in a global, player independent category.

> **Example**
>
> `%globalpoint.global_knownusers.left:100%`

Item: `item`🔗

With this variable you can display amount of specific items in player's inventory or a number needed to reach specific amount. The first argument is the name of an item (as defined in *items.yml*) and the second one is either `amount` or `left:x`, where `x` is a number.

> **Example**
>
> `%item.stick.amount%`

Location: `location`🔗

This variable resolves to player's current location, formatted as an absolute location format (more about it in the *Reference* chapter). The location will contain yaw and pitch. You can use it instead of coordinates as location arguments in events, conditions and objectives.

> **Example**
>
> `%location%`

Calculate mathematical expression: `math.calc`🔗

This variable allows you to perform a calculation based on other variables (for example point or objective variables) and resolves to the result of the specified calculation. The variable always starts with `math.calc:`, followed by the calculation which should be calculated. Supported operations are +, -, *, / and ^. You can use ( ) and [ ] braces and also calculate absolute values with | | (but don't use this in the command event as it splits the commands at every |). If you want to use variables in the calculation, don't put % around them.

> **Example**
>
> `%math.calc:100*(15-point.reputation.amount)%`

NPC: `npc`🔗

It's a very simple variable. It's replaced by the name of the NPC in player's language.

> **Example**
>
> `%npc%`

Objective: `objective`🔗

Using this variable you can display a property of an objective. The first argument is an ID of the objective as defined in *objectives.yml* (not the type). Make sure that the player has this objective active or it will be replaced with nothing (""). Second argument is the name of a property you want to display. All properties are described in "Objectives List" chapter.

> **Example**
>
> `%objective.kill_zombies.left%`

Player: `player`🔗

This variable will be replaced with the name of the player. If you add `display` argument, it will use display name instead of real name.

> **Example**
>
> `%player.display%`

Point: point🔗

This variable displays the amount of points you have in some category or amount of points you need to have to reach a number. The first argument is the name of a category and the second argument is either `amount` or `left:x`, where x is a number.

> **Example**
>
> `%point.reputation.left:15%`

Version: version🔗

This variable displays the version of the plugin. You can optionally add the name of the plugin as an argument to display version of another plugin.

> **Example**
>
> `%version.Citizens%`

Last update:

Notifications🔗

Overview🔗

A notification is any message that is sent to the player either by BetonQuest itself, one of its plugins, or through a custom notify event.

You can customize what is sent by editing `messages.yml`. This contains a list of languages and all the notifications used by BetonQuest.

If you want to customize how a notification is sent to the player then you can edit `custom.yml` to define a NotifyIO to use to for the specific notification. For example you can send a pop up achievement when the journal is updated and a bossbar notification when breaking blocks in a block objective.

What is a NotifyIO🔗

A NotifyIO is some method of sending a notification out. BetonQuest provides some by default and third party plugins may register their own.

The following are provided by default:

- `suppress` - Does not send anything. Good at turning off some notifications.
- `chat` - (Default) Sends the notification as a chat message
- `advancement` - Sends a notification via a popup advancement
- `actionbar`- Sends a notification via the actionbar
- `bossbar` - Sends a notification via a bossbar
- `title` - Sends a notification via a title
- `subtitle` - Sends a notification via a subtitle

Default NotifyIO🔗

If not set, the default NotifyIO is `chat`. You can change the default to `actionbar` by setting the following in \`config.yml':

```
default_notify_IO: actionbar
```

Configuring Notifications🔗

When a notification is generated it will usually have one or more categories assigned to it. These categories are searched for, in order, in all `custom.yml` files under the section `notifications`. The first category found will be used to configured the notification. If none were found then a category of `default` will be search for.

A typical custom.yml file may have the following:

```yaml
notifications:
  # A new journal entry has been added
  new_journal_entry:
    io: advancement
    frame: challenge
    icon: map

  # All infomation notifications
  info:
    io: chat
    sound: BLOCK_CHEST_CLOSE
```

When a new journal entry is added, it will send a notification with the following categories:

- new_journal_entry
- info

In the above file, it will find `new_journal_entry` first so will ignore `info`. This defines the settings for the notification by using the `advancement` notifyIO with a challenge frame and a map for the icon.

When a new changelog occurs it will send a notification using the following categories:

- changelog
- info

In the above file, it will find `info` which defines that it should be sent via the chat with a specific sound played as well.

Categories🔗

Categories are a way of defining settings for a notification. BetonQuest uses many categories itself but you can define your own custom categories and use them through the `notify` event.

In general a BetonQuest notification will use the same category name for a notification as it uses in `messages.yml` to define the language and text of the notification. It will also use more

general categories to allow you to more broadly define settings and only provide specific settings for some notifications. Remember the categories are searched for in order so the first category will be used in preference to those later in the list.

A list of categories used by BetonQuest are as follows:

Notification	Categories
Pullback	pullback, error
Command Blocked	command_blocked,error
No Permission	no_permission,error
Busy	busy,error
Changelog	changelog,info
Inventory Full	inventory_full,error
Language Changed	language_changed,info
Mobs to Kill	mobs_to_kill,info
Money Given	money_given,info
Money Taken	money_taken,info
Quest Cancelled	quest_cancelled,quest_canceled,info
Items Given	items_given,info
New Journal Entry	new_journal_entry,info
Items Taken	items_taken,info
Blocks to Break	blocks_to_break,info
Blocks to Place	blocks_to_place,info
Animals to Breed	animals_to_breed,info
Mobs to click	mobs_to_click,info
Fish to catch	fish_to_catch,info
Players to kill	players_to_kill,info
Potions to brew	potions_to_brew,info
Sheep to shear	sheep_to_shear,info

Configuring NotifyIO's🔗

Each NotifyIO has its own set of configuration that can be used. None are required. The built in ones will be described in this section.

Suppress⚓

Does not output anything. Can be used to stop notifications.

Chat⚓

Writes the notification to the players chat.

Option	Description
sound	Sound to play. If blank, no sound. Can be from here *(https://hub.spigotmc.org/ javadocs/bukkit/org/bukkit/Sound.html)* or the name of a sound including from a custom resource.

Advancement⚓

Shows the notification using an achievement popup.

Option	Description
sound	Sound to play. If blank, no sound. Can be from here *(https://hub.spigotmc.org/ javadocs/bukkit/org/bukkit/Sound.html)* or the name of a sound including from a custom resource.
frame	What Achievement frame to use. Can be: `challenge`, `goal`, `task`
icon	What icon to show. Must be the vanilla name of an item. Example: minecraft:map

Actionbar⚓

Shows the notification using the actionbar.

Option	Description
sound	Sound to play. If blank, no sound. Can be from here *(https://hub.spigotmc.org/ javadocs/bukkit/org/bukkit/Sound.html)* or the name of a sound including from a custom resource.

Bossbar⚓

Shows the notification using a bossbar.

Option	Description
sound	Sound to play. If blank, no sound. Can be from here *(https://hub.spigotmc.org/ javadocs/bukkit/org/bukkit/Sound.html)* or the name of a sound including from a custom resource.

Option	Description
barFlags	What flags to add to bossbar. One of the following from here *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/boss/BarFlag.html)*.
barColor	What color to draw the bar. One of the following from here *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/boss/BarColor.html)*
progress	What progress to show the bar. A floating point number between 0.0 (empty) and 1.0 (full)
style	What style bar to use. One of the following from here *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/boss/BarStyle.html)*
stay	How many ticks to keep the bar on screen. Defaults to 70
countdown	If set, will step the progress of the bar by countdown steps. For example, if set to 10, then 10 times during the time it is on the screen the progress will drop by 1/10

Title⟨⟩

Shows the notification using a title.

Option	Description
sound	Sound to play. If blank, no sound. Can be from here *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/Sound.html)* or the name of a sound including from a custom resource.
fadeIn	Ticks to fade the title in. Default 10
stay	Ticks to keep title on screen. Default 70
fadeOut	Ticks to fade the title out. Default 20
subTitle	Optional subtitle to show. All _'s are replaced with spaces

SubTitle⟨⟩

Shows the notification using a subtitle.

Option	Description
sound	Sound to play. If blank, no sound. Can be from here *(https://hub.spigotmc.org/javadocs/bukkit/org/bukkit/Sound.html)* or the name of a sound including from a custom resource.
fadeIn	Ticks to fade the title in. Default 10
stay	Ticks to keep title on screen. Default 70
fadeOut	Ticks to fade the title out. Default 20

Custom Notifications🔗

Using the `notify` event a custom notification can be sent. It can make use of any category defined or can override by directly defining the NotifyIO configuration options. Please refer to the Events-List chapter for more details on this event.

Examples🔗

Example 1 - Custom Notifications🔗

Assuming the following custom.yml file:

```yaml
notifications:
    # Test Categories
    test_suppress:
        io: suppress

    test_chat:
        io: chat
        sound: BLOCK_CHEST_CLOSE

    test_advancement:
        io: advancement
        sound: BLOCK_CHEST_CLOSE
        frame: challenge # challenge|goal|task
        icon: map

    test_actionbar:
        io: actionbar
        sound: BLOCK_CHEST_CLOSE

    test_bossbar:
        io: bossbar
        sound: BLOCK_CHEST_CLOSE
        #barFlags: create_fog,darken_sky,play_boss_music
        barColor: purple # blue|green|pink|purple|red|white
        progress: 0.0 # 0.0 - 1.0
        style: solid # segmented_10|segmented_12|segmented_20|segment
        stay: 70

    test_title:
        io: title
        sound: BLOCK_CHEST_CLOSE
        fadeIn: 10
```

```
        stay: 70
        fadeOut: 20
        #subtitle:

    test_subtitle:
        io: subtitle
        sound: BLOCK_CHEST_CLOSE
        fadeIn: 10
        stay: 70
        fadeOut: 20
```

And the following events run in order:

```
# Test of Category
notify_cat_suppress: notify Test Notify category:test_suppress
notify_cat_chat: notify Test Notify category:test_chat
notify_cat_advancement: notify Test Notify category:test_advancement
notify_cat_actionbar: notify Test Notify category:test_actionbar
notify_cat_bossbar: notify Test Notify category:test_bossbar
notify_cat_title: notify Test Notify category:test_title
notify_cat_subtitle: notify Test Notify category:test_subtitle

# Test of Category + some custom
notify_catcus_title_sub: notify Test Notify category:test_title subt
notify_catcus_bossbar_red: notify Test Notify category:test_bossbar

# Test of totally custom, needs an io
notify_cus_bossbar_green: notify Test Notify io:bossbar barColor:gre
notify_cus_advancement: notify Test Notify io:advancement icon:hoppe
```

A video can be found here *(https://www.youtube.com/watch?v=mmEfIXp4dxA)*

Example 2 - Bossbar Countdown🗐

Assuming the following events run in order:

```
notify_cus_bossbar_countdown1: notify Countdown Test Bossbar io:boss
notify_cus_bossbar_countdown2: notify Countdown Test Bossbar io:boss
```

A video can be found here *(https://i.imgur.com/x6Ihe7W.mp4)*

Example 3 - Using suppress on a builtin notification

🔗

If you don't want to see the `changelog` notifications then you can use the suppress notifyio for the category `changelog`.

Example custom.yml

```
notifications:
  changelog:
    io: suppress
```

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/Conversations.md)

Conversations

General Information🔗

Each conversation must define name of the NPC (some conversations can be not bound to any NPC, so it's important to specify it even though an NPC will have a name) and his initial options.

```yaml
quester: Name
first: option1, option2
stop: 'true'
final_events: event1, event2
interceptor: simple
NPC_options:
  option1:
    text: Some text in default language
    events: event3, event4
    conditions: condition1, !condition2
    pointers: reply1, reply2
  option2:
    text: '&3This ends the conversation'
player_options:
  reply1:
    text:
      en: Text in English
      pl: Tekst po polsku
    event: event5
    condition: '!condition3'
    pointer: option2
  reply2:
    text: 'Text containing '' character'
```

> **Note**
>
> Configuration files use YAML syntax. Google it if you don't know anything about it. Main rule is that you must use two spaces instead of tabs when going deeper into the hierarchy tree. If you want to write `'` character, you must double it and surround the whole text with another `'` characters. When writing `true` or `false` it also needs to be surrounded with `'`. If you want to start the line with & character, the whole line needs to be surrounded with `'`. You can check if the file is correct using this tool *(http://www.yamllint.com)*.

- `quester` is name of NPC. It should be the same as name of NPC this conversation is assigned to for greater immersion, but it's your call.

- `first` are pointers to options the NPC will use at the beginning of the conversation. He will choose the first one that meets all conditions. You define these options in `npc_options` branch.
- `final_events` are events that will fire on conversation end, no matter how it ends (so you can create e.g. guards attacking the player if he tries to run). You can leave this option out if you don't need any final events.
- `stop` determines if player can move away from an NPC while in this conversation (false) or if he's stopped every time he tries to (true). If enabled, it will also suspend the conversation when the player quits, and resume it after he joins back in. This way he will have to finish his conversation no matter what. *It needs to be in ' '!* You can modify the distance at which the conversation is ended / player is moved back with `max_npc_distance` option in the *config.yml*.
- `interceptor` optionally set a chat interceptor for this conversation. Multiple interceptors can be provided in a comma-separated list with the first valid one used.
- `NPC_options` is a branch with texts said by the NPC.
- `player_options` is a branch with options the player can choose.
- `text` defines what will display on screen. If you don't want to set any events/conditions/ pointers to the option, just skip them. Only `text` is always required.
- `conditions` are names of conditions which must be met for this option to display, separated by commas.
- `events` is a list of events that will fire when an option is chosen (either by NPC or a player), defined similarly to conditions.
- `pointer` is list of pointers to the opposite branch (from NPC branch it will point to options player can choose from when answering, and from player branch it will point to different NPC reactions).

When an NPC wants to say something he will check conditions for the first option (in this case `option1`). If they are met, he will choose it. Otherwise, he will skip to next option (note: conversation ends when there are no options left to choose). After choosing an option NPC will execute any events defined in it, say it, and then the player will see options defined in `player_options` branch to which `pointers` setting points, in this case `reply1` and `reply2`. If the conditions for the player option are not met, the option is simply not displayed, similar to texts from NPC. Player will choose option he wants, and it will point back to other NPC text, which points to next player options and so on.

If there are no possible options for player or NPC (either from not meeting any conditions or being not defined) the conversations ends. If the conversation ends unexpectedly, check the console - it could be an error in the configuration.

This can and will be a little confusing, so you should name your options, conditions and events in a way which you will understand in the future. Don't worry though, if you make some mistake in configuration, the plugin will tell you this in console when testing a conversation. Also, study the default conversation included with the plugin to fully understand how powerful this system can be.

Cross-conversation pointers🔗

If you want to create a conversation with multiple NPCs at once or split a huge conversation into smaller, more focused files, you can point to NPC options in other conversation. Just type the pointer as `conversation.npc_option`.

Keep in mind that you can only cross-point to NPC options. It means that you can use those pointers only in `first` starting options and in all player options. Using them in NPC options will throw errors.

Conversation variables🔗

You can use variables in the conversations. They will be resolved and displayed to the player when he starts a conversation. A variable generally looks like that: `%type.optional.arguments%`. Type is a mandatory argument, it defines what kind of variable it is. Optional arguments depend on the type of the variable, i.e. `%npc%` does not have any additional arguments, but `%player%` can also have `display` (it will look like that: `%player.display%`). You can find a list of all available variable types in the "Variables List" chapter.

> **Note**
>
> If you use a variable incorrectly (for example trying to get a property of an objective which isn't active for the player, or using %npc% in `message` event), the variable will be replaced with empty string ("").

Translations🔗

As you can see in default conversation, there are additional messages in other languages. That's because you can translate your conversations into multiple languages. The players will be albe to choose their preferred one with /questlang command. You can translate every NPC/ player option and quester's name. You do this like this:

```
quester:
  en: Innkeeper
  pl: Karczmarz
  de: Gastwirt
```

As said before, the same rule applies to all options and quester's name. The player can choose only from languages present in *messages.yml*, and if there will be no translation to this language in the conversation, the plugin will fall back to the default language, as defined in *config.yml*. If that one is not defined, there will be an error.

You can also translate journal entries, quest cancelers and `message` events, more about that later.

Conversation displaying🔗

By default BetonQuest uses the most native and safe way of displaying a conversation, which is the Minecraft chat. You choose the option by typing their number in. You can however change it with `default_conversation_IO` option in *config.yml* file. Default value is `simple`. By changing it to `tellraw` you will add a possibility to click on options. Keep in mind that if the chat is quickly flowing, players will sometimes "miss" an option and click another one. There is a display type that doesn't suffer from this problem at all, it's called `chest`. It will display the conversation in an inventory GUI, where the NPC's text and options will be shown as item lore. Alternatively use `slowtellraw` which provides the npc responses line by line delayed by 0.5 seconds. If you have `protocollib` then you can use `menu`.

You can control the colors of conversation elements in the *config.yml* file, in `conversation_colors` section. Here you must use names of the colors.

If you're using the `chest` display method you can change the option's item to something else than Ender Pearl by adding a prefix to that option's text. The prefix is a name of the material (like in *items.yml*) inside curly brackets, with optional damage value after a colon. Example of such option text: `{diamond_sword}I want to start a quest!` or `{wool:10}Purple!`

In case you want to use a different type of conversation display for a specific conversation you can add `conversationIO: <type>` setting to the conversation file at the top of the YAML hierarchy, which is the same level as `quester` or `first` options).

Chat Interceptors🔗

While engaged in a conversation, it can be distracting when messages from other players or system messages interfere with the dialogue. A chat interceptor provides a method of intercepting those messages and then sending them after the conversation has ended.

You can specify the default chat interceptor by setting `default_interceptor` inside the `config.yml`. Additionally, you can overwrite the default for each conversation by setting the `interceptor` key inside your conversation file.

The default configuration of BetonQuest sets the `default_interceptor` option to `packet,simple`. This means that it first tries to use the `packet` interceptor. If that fails it falls back to using the `simple` interceptor.

BetonQuest adds following interceptors: `simple`, `packet` and `none`:

The `simple` interceptor works with every Spigot server but only supports very basic functionality and may not work with plugins like Herochat.

The `packet` interceptor requires the ProtocolLib plugin to be installed. It will work well in any kind of situation.

The `none` interceptor is an interceptor that won't intercept messages. That sounds useless until you have a conversation that you want to be excluded from interception. In this case you can just set `interceptor: none` inside your conversation file.

Advanced: Extends🔗

Conversation also supports the concept of inheritance. Any option can include the key `extends` with a comma delimited list of other options of the same time. The first option that does not have any false conditions will have it's text, pointers and events merged with the extending option. The extended option may itself extend other options. Infinite loops are detected.

```
NPC_options:

  ## Normal Conversation Start
  start:
    text: 'What can I do for you'
    extends: tonight, today

  ## Useless addition as example
  tonight:
    # Always false
    condition: random 0-1
    text: ' tonight?'
    extends: main_menu

  today:
    text: ' today?'
    extends: main_menu

  ## Main main_menu
  main_menu:
    pointers: i_have_questions, bye
```

In the above example, the option *start* is extended by both *tonight* and *today*, both of whom are extended by *main_menu*. As *tonight* has a false condition the *today* option will win. The *start* option will have the pointers in main_menu added to it just as if they were defined directly in it and the text will be joined together from *today*. If you structure your conversation correctly you can make use of this to minimize duplication.

Last update:

*(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/
Configuration.md)*

Configuration🔗

The configuration of BetonQuest is mainly done in the `config.yml` file. All of its options are
described on this page. There is also additional information about backups, updates and
database transfers. If you fail to understand options just keep their default values. You can
always change them when you have gained a more complete understanding of this plugin.

Config Options🔗

> **Warning**
>
> Do not touch `version:` option! It may corrupt your files!

MySQL Database🔗

Setup🔗

In order to use a MySQL Database for saving all the data you need to fill out the mysql config
section.

```
1  mysql:
2    enabled: true   #Set this to true.
3    host: ''         #This is the IP of your MySQL server. If it run
4    port: ''         #This is the port your MySQL server runs on.
5    user: ''         #The name of the database user that BetonQuest
6    pass: ''         #The password of that user.
7    base: ''         #The database that BetonQuest will write to.
8    prefix: betonquest_  #The table prefix of BetonQuests data in t
```

Migrating a database from SQLite to MySQL and back🔗

Follow these few simple steps to migrate your database easily:

1. Create a backup with /q backup command.
2. Extract database backup from it.
3. Turn the server off.
4. Place the *database-backup.yml* file inside the plugin's directory.
5. Edit which database type you want to use by setting the `enabled` option in the `mysql`
   section to true or false.

6. Start the server.

7. Check for errors.

8. If there are no errors, enjoy your migrated database.

9. If there are any errors, post them to the developer or try to fix them if you know how.

Default Language⏷

`language:` is the default translation of the plugin for every new player. Currently, there are 9 languages available: English (en), Polish (pl), German (de), French (fr), Spanish (es), Chinese (cn), Dutch (nl), Italian (it) and Hungarian (hu).

Updating⏷

The `update` section controls the Auto-Updater. It has the following settings:

- `enabled` (default `true`). Enables or disables the Updater. If set to false, it is not possible to update with the updater and no version checks are executed.
- `strategy` (default `MINOR`). The update strategy is the most important feature of the Auto-Updater. An explanation is available here.
- `automatic` (default `true`). If true the updater will download new Versions automatically. Otherwise, the updater will only download new versions when the update command is executed.

Journal slots⏷

`default_journal_slot` is the inventory slot in which the journal will appear after using the `/journal` command. BetonQuest will try to move items out of the way if the slot is occupied. If the inventory is full the journal will not be added. You can disable this behaviour by setting the option to `-1`. BetonQuest will then just use any free slot.

Citizens identifier⏷

`citizens_npcs_by_name` sets whether NPCs from Citizens 2 should be identified in main.yml by their name instead of their id. This is a dangerous setting as two different NPC's at the opposite edges of your world that share the same name by accident will trigger the same quest.

Conversation End Distance⏷

`max_npc_distance` is the distance you need to walk away from the NPC for the conversation to end (in the case of using chat-based conversation interface).

Default conversation style🔗

`default_conversation_IO` is a comma-separated list of conversation interfaces with the first valid one used. Read this page for more information about conversation interfaces.

Default Chat interceptor🔗

`default_interceptor` is a comma-separated list of chat interceptors with the first valid one used. Read this page for more information about chat interceptors.

Conversation Chat Display options🔗

`display_chat_after_conversation` this will prevent all chat messages from displaying during a conversation and it will show them once it's finished.

Combat Delay🔗

`combat_delay` is a delay (in seconds) the player must wait before starting a conversation after combat.

Conversation pullback message🔗

`notify_pullback` will display a message every time the player is pulled back by the `stop` option in conversations (in the case of chat-based conversations). It notifies players that they are in a conversation, and the pullback is not a bug.

Default Package name🔗

`default_package` is a name of the package that should be used when a package is not specified in /q command. This is for your convenience.

Adjusting the backpack to the KeepInventory gamerule🔗

`remove_items_after_respawn` option should be turned on if "keepInventory" gamerule is not being used. It prevents other plugins from duplicating quest items after death.

When a player dies, their quest items are removed from drops and stored in the backpack, but some plugins may try to restore all items to the player (for example WorldGuard custom flag keep-inventory). That is why BetonQuest removes the quest items that are in a player's inventory after they respawn again, to be sure they were not readded. The "keepInventory" gamerule, however, works differently - the items are never dropped, so they cannot be added to backpack. Removing them from the inventory would destroy them forever. Sadly, Bukkit does not allow for gamerule checking, so it is up to you to decide.

Once again, if you have "keepInventory" gamerule true, this setting has to be false and vice versa.

Quest Items break behaviour🔗

`quest_items_unbreakable` controls whether quest items can be broken by using them. This was used in the past when the `unbreakable` tag couldn't be added to items. Turn it off and make your quest items unbreakable by vanilla means.

Sounds🔗

This section defines what sounds will be played on these occasions:

- `start` and `end` refer to start and end points of conversations.
- `journal` is played upon updating the journal.
- `update` is played when there is a changelog file, used to draw your attention.
- `full` is played when the player executes `/journal` but his inventory is full.

A list of all possible sounds for the latest Minecraft version can be found here *(https:// hub.spigotmc.org/javadocs/spigot/org/bukkit/Sound.html)*.

Conversation Command Banlist🔗

`cmd_blacklist` is a list of commands that cannot be used during a conversation. Remember that you can only type single words (command names) here!

Compatibility Hooks🔗

`hook` controls compatibility with other plugins. You can turn off each hook here.

Journal🔗

`journal` controls various settings of the journal:

- `chars_per_page` is the number of characters before a page break. If it is set too high, the text on a journal page can overflow and become invisible. This was replaced by `chars_per_line` and `lines_per_page` and is only required if you don't like the new behaviour.
- `chars_per_line` is the number of characters before a line break. If it is set too high, the text on a journal page can overflow and become invisible. If this is not set, BQ will fall back on the old page wrapping behaviour configured through `chars_per_page`.
- `lines_per_page` is the number of lines before a new page. If it is set too high, the text on a journal page can overflow and become invisible. This is only required if `chars_per_line` is set.

- `one_entry_per_page` makes each entry take a single page. Note that it will not expand to other pages even if it overflows, so keep your entries short.
- `reversed_order` controls the chronological order of entries in the journal. By default, the entries are ordered from newest to oldest. It is reversible, but this will force players to click through many pages to get to the most recent entry.
- `hide_date` hides the date of each entry. Set it to true if you don't want this functionality.
- `full_main_page` makes the main page always take a full page. If a lot of information is being displayed, it is advised to make this true. If you use the main page only for small notifications, set it to false, so the entries can follow immediately.
- `show_separator` shows a separator between journal entries (default: true). Customize the separator in `messages.yml` with the key `journal_separator`.

Journal Colors🎨

`journal_colors` controls the colors used in the journal. It takes color codes without the & character.

- `date.day` is the day number
- `date.hour` is the hour number
- `line` is the delimiter between entries
- `text` is the text of the entry

Conversation Colors🔵

`conversation_colors` controls the colors of the conversation. It takes color names *(https://hub.spigotmc.org/javadocs/spigot/org/bukkit/ChatColor.html)*. If you want to add a font style (bold, italic etc.) do so after placing a comma.

- `npc` is the name of the NPC
- `player` is the name of the player
- `text` is the NPC's text
- `answer` is the text of player's answer (after choosing it)
- `number` is the option number
- `option` is the text of an option
- `date_format` is the Java date format *(https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html)* used in journal dates. It needs to have a space between the day and hour.
- `debug` is responsible for logging the plugin's activity to *debug.log* file in *logs* directory. Turning this on can slow your server down. However, if you experience any errors, turn this on and let the plugin gather the data and send logs to the developer. Note that the first run of the plugin will be logged anyway, just as a precaution.
- `conversation_IO_config` manages settings for individual conoversation IO's:
    - `chest` manages settings for the chest conversation IO
        - `show_number` will show the player number option if true (default: true)

- **show_npc_text** will show the npc text in every player option if true (default: true)

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/Commands-and-permissions.md)

Commands and permissions🔗

Commands🔗

- /j - gives the journal
- /backpack - opens the backpack
- /q - lists all available admin commands
- /q reload - reloads the plugin
- /q objectives {playerName} [list/add/del/complete] [instruction] - shows player's currently active objectives
- /q tags {playerName} [list/add/del] [tag] - lists all player's tags
- /q points {playerName} [list/add/del] [category] [amount] - lists all player's points in all categories
- /q journal {playerName} [list/add/del] [package.pointer] [date] - lists
- /q event {playerName} {package.eventID} - fires an event for the player
- /q condition {playerName} {package.conditionID} - shows if the player meet specified condition or not
- /q item {package.itemID} - creates an item based on what you're holding in hand
- /q give {package.itemID} - gives you an item defined in the configuration
- /q config {set/add/read} {path} [value] - sets, adds or reads values from configuration
- /q purge {playerName} - deletes all player's data from the database
- /q rename {tag/point/globalpoint/objective/entry} {oldName} {newName} - renames all specified things in the database
- /q delete {tag/point/objective/entry} {name} - deletes all specified things in the database
- /q backup - creates a backup of configuration files and database
- /q update - updates the plugin to the newest version.
- /q create {package}: creates new package with given name, filled with default quest
- /q vector {packname.variable} {newvariable}: calculates the vector from first location variable to you position and saves it as second variable
- /q version: displays the versions of BetonQuest, the server and all hooked plugins
- /q debug [true/false]: enable debug mode and write all down in a log file or disable the debug mode
- /questlang {lang} - changes the language for the player (and globally if used from console). `default` language will use the language defined in *config.yml*.

Aliases🔗

- /j: bj, journal, bjournal, betonjournal, betonquestjournal
- /backpack: b, bb, bbackpack, betonbackpack, betonquestbackpack

- /q: bq, bquest, bquests, betonquest, betonquests, quest, quests
    - objective: o, objectives
    - tag: t, tags
    - point: p, points
    - event: e, events
    - condition: c, conditions
    - journal: j, journals
    - item: i, items
    - give: g
    - rename: r
    - delete: d, del
    - create: package
- /questlang: ql

Permissions🔗

- betonquest.admin - allows using admin commands (/q ...) and creating an NPC from blocks
- betonquest.journal - allows using /j command (default for players)
- betonquest.backpack - allows using /backpack command (default for players)
- betonquest.conversation - allows talking with NPCs (default for players)
- betonquest.language - allows changing the language (default for players)

> **Warning**
>
> Don't give betonquest.admin permission to people you don't fully trust. They can use /q config command to add a `command` event, and this way execute any command as the console. This might be dangerous.

Main command details🔗

Reloading loads all data from configuration, but not everything is updated. Player's data isn't touched to avoid lags made by database saving. The database is also the same, you will have to reload/restart the whole server for the database to change.

Tags subcommand allows you to easily list and modify tags. `'/q tags Beton'` would list tags for player Beton. `'/q tags Beton add test'` would add "test" tag for that player, and `'/q tags Beton del test'` would remove it.

Points subcommand is similar - listing points is done the same way. Adding points to a category looks like that: `'/q points Beton add reputation 20'` (adding 20 points to "reputation" category). You can also subtract points with negative amounts. Removing the whole point category can be achieved by `'/q points Beton del reputation'`.

Journal subcommand works in the same way as those two above. Adding and removing looks like /q `journal Beton add default.wood_started` (or `del`), and you can also specify the date of entry when adding it, by appending date written like this: `23.04.2014_16:52` at the end of the command. Note that there is _ character instead of space!

Objective subcommand allows you to list all active objectives (shown as their labels) of the player. It can also directly add or cancel objectives using instruction strings. You can also complete the objective for the player using `complete` argument - it will run all events and remove the objective.

Running events for online players can be done with event argument: '/q `event Beton give_emeralds`' would run "give_emeralds" for player Beton (if he's online) from default package (not necessarily "default" but rather the default one specified in *config.yml*). If you want to run a static event, replace player's name with `-`.

There is also condition argument for checking conditions, for example '/q `condition Beton has_food`'. Events and conditions need to be defined in their files, this command doesn't accept raw instructions. You can skip package name, the plugin will assume you're reffering to package specified in `default_package` option in *config.yml* file. If you want to check a static condition replace the player's name with `-`.

If you need to create for example "Nettlebane" quest item, just hold it in your hand and type '/q `item nettlebane`'. It will copy the item you're holding into the *items.yml* file and save it there with the name you specified (in this case "nettlebane"). You can skip the package name here as well.

The '/q `give package.item`' command will simply give you specified item.

Config subcommand is used to modify or display values in configuration files. `set` option replaces the value with what you typed, `add` simply adds your string to the existing value. (Note on spaces: by default the plugin won't insert a space between existing and added value. You can however achieve that by prefixing the string with _ character. For example: existing string is `objective location`, and you want to add `100;200;300;world;10`. Your command will look like /q `config add default.events.loc_obj _100;200;300;world;10`). `read` option allows you to display config value without modifying it.

Path in this command is like an address of the value. Next branches are separated by dots. For example language setting in main configuration has path `config.language`, and a text in "bye" player option in default quest has path `default.conversations.innkeeper.player_options.bye.text`

You can purge specific player with '/q `purge Beton`' command, where Beton is the name of the player. To purge the entire database at once simply change the prefix in *config.yml* or delete *database.db* file.

Delete command ('/q `delete`') allows you to delete from the database every tag, point, objective or journal entry with specified name.

Rename command ('`/q rename`') allows you to rename every tag, point, globalpoint, objective or journal entry in the database. In case of an objective it will also rename the objective in *objectives.yml*, so it continues to work correctly.

If you want to backup your configuration and database make sure that your server is empty (this process requires all data to be saved to database -> all players offline) and run '`/q backup`' command. You will get a zip file containing all your data, ready to be unzipped for restoring the plugin.

Update command ('`/q update`') will try to download the newest version of the plugin and save it to the update folder. This folder is then handled by Spigot to update the plugin. If you accidentally use this command but do not wish to update the plugin, you should remove `BetonQuest.jar` file from the `plugins/update` folder before restarting/reloading the server.

Using '`/q create beton`' command you will create new package named '`beton`'. It will contain the default quest.

The `/q vector` command allows you to create vector variables from the specified in first argument location variable to your position. The result will be saved to the "vectors.{second argument}" variable.

The debug command ('`/q debug`') allow you to enable or disable the debug mode. If the debug mode is enabled, the last 1000 log entries are written down to the latest.log file as history and write everything down until the debug mode os disabled. The file `/plugins/BetonQuest/logs/latest.log` is renamed to the current date and time on server start, if the debug mode is not longer active. The debug mode is useful, if you want to report an issue but it is not clear why the issue occurs.

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/Updating-and-Backups.md)

Updating and Backups

Updating in BetonQuest🔗

There are two ways to update BetonQuest. You can either use the comfortable Auto-Updater or do a manual update by yourself.

Disclaimer for all versions below 2.0: The Auto-Updater has a single flaw - all comments in your configurations will be lost during the update process.

Deciding what version to use🔗

Each plugin version is a number that consists of three parts. For example 2.4.3, the first number (2 in this example) is named MAYOR, the second MINOR (4 in this example) and the third PATCH (3 in this example). When we release a new version of BetonQuest we will change these numbers in a specific way. Each number has a fixed meaning.

Update Strategy	MAYOR	MINOR	PATCH
Bug Fixes	✅	✅	✅
New Features	✅	✅	❌

Update Strategy	MAYOR	MINOR	PATCH
Breaking Changes	✅	❌	❌

You can also append `_DEV` to each strategy. This will download the dev builds for the corresponding version. This is not recommended for production/live servers, as devbuilds can contain bugs. Therefore, enabling a `_DEV` strategy will automatically disable automated updates. If you don't care about the risk you can re-enable it, otherwise check what the new devbuilds do before executing `/q update`.

Deciding if auto-updates are reasonable🔗

Apart from the version you also have to choose if you want BetonQuest to update automatically or only after confirmation. Having automatic updates enabled is handy for `PATCH` and `MINOR` but risky on `MAJOR` strategies. Automatic updates for any `_DEV` version are dangerous as these can hold very bad bugs. Only use this for test servers.

Disabling automated updates still allows for the use of `/q update`! You won't have to go to Spigot ever again

Setup of the Auto-Updater🔗

Read the update section to correctly setup the Auto-Updater.

After updating to a new version (manually or automatically), configuration files and database will be automatically backed up to a zip file to prevent losing your work due to errors. Then, configuration will be converted to a new version. At the end, the localization will be updated with new languages, and a *changelog.txt* file will be created.

If there were any unexpected errors during an update process, download the previous version, restore your configs from backup, and disable autoupdating feature. Don't forget to post your

error as an Issue on GitHub *(https://github.com/BetonQuest/BetonQuest/issues/new?template=bug_report_template.md)* so it can be fixed!

When you enter the server, BetonQuest will alert you about changes and ask you to read changelog.txt file located in plugin's main directory.

Updating manually🔗

Configs update automatically when a new dev build has been installed but all comments in your configs will be lost due to Spigots limitations. We will make our own config system in BetonQuest 2.0. though so this will not last long.

To keep your comments follow this guide: This is a manual update, so you will have to change some errors in your quests afterwards. If you already did this once and want to update to a newer dev build (using /q update) just check if the version has changed by clicking the link in the third step. If it did not your comments are safe.

- • Make a backup
- • Stop server.
- • Open up /plugins/BetonQuest/config.yml
- • Set the version: option to the one that you see here: This might change so please look it up each time you update!
  https://github.com/BetonQuest/BetonQuest/blob/master/src/main/resources/config.yml *(https://github.com/BetonQuest/BetonQuest/blob/master/src/main/resources/config.yml)*
- • Update BetonQuest.jar
- • Start Server
- • Fix all issues (we changed the stuff down below.)

Backups🔗

Every time the plugin updates the configuration, a backup will be created. This is especially important if a development version is being used because they may be unstable. A backup can also be created manually by running /q backup command. It needs to be run from the console on an empty server because it heavily uses the database.

You can find your backups in *backup* directory in the plugin's folder. They are .zip files containing all your configuration and *database-backup.yml* file, which - as the name says - is your database backup. To replace your configuration with an older backup, delete all the files (except backups and logs) and replace them with the files from .zip file.

If you want your database loaded, place *database-backup.yml* file in plugin's directory. When the plugin sees this file while enabling, it will backup the current database and load all data from that file to the database. A backup of the old database can be found in *backups* folder, so if you ever need to load it back, just rename it to *database-backup.yml* and place it back in main plugin's

directory. Note that *database-backup.yml* file will be deleted after loading, so it does not replace your database on next plugin start.

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/User-Documentation/CHANGELOG.md)

Changelog🔗

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog *(https://keepachangelog.com/en/1.0.0/)*, and this project adheres to Semantic Versioning *(https://semver.org/spec/v2.0.0.html)*.

[1.12.0-DEV-170] - 2020-08-29🔗

Added🔗

- Tags and Objectives can now be removed with a static event for all players, even if they are not online
- deletepoint event can now also be called to delete all points for all players
- journal del event can now also be called as static
- Added integration for MMOGroup plugins (MMOCore, MMOItems, MMOLib)
- MMOClass condition (type & class)
- MMOProfession condition
- MMOAttribute condition
- MMOItems item condition (item in inventory)
- MMOItems hand condition (item in main/offhand)
- MMOLib stats condition (a ton of stats from Core and Items combined)
- equal argument for condition 'empty'
- Condition 'wand' can now have an option '
- Implementing 1.15 support for Events and Conditions
- New Chat event, that write chat messages for a player
- Added 'pickup' objective
- Added stopnpc event, that will stop the movenpc event
- Added teleportnpc event, that will stop the movenpc event and teleport the npc to a given location
- Added option check_interval for holograms in custom.yml andd added GlobalVariable support
- Added deletepoint event to delete player points
- Added mythicmobdistance condition that will check if a specific MythicMobs entity is near the player
- Added level argument to 'experience' objective and condition
- Added prefix argument in password objective
- Added level argument to 'experience' objective and condition
- Added prefix argument in password objective

- Added fail argument in password objective
- Added notify option to point event
- Added an interceptor that does not intercept: 'none'

Changed🔗

- devbuilds always show notifications for new devbuilds, even when the user is not on a _DEV strategy
- Items for HolographicDisplays are now defines in items.yml
- Command 'bq rename' can now be used for globalpoints
- The old updater was replaced with a new one
- AchievementCondition is replaced with AdvancementCondition
- Renamed objective Potion to Brew
- Renamed 'monsters' condition to 'entities'
- Renamed 'xp' event to 'experience'
- new config option mysql.enabled
    - if you already have an installation, you can add this manually to get rid of the mysql warning during startup
- events in conversation options are now executed before npc or player responses are printed
- message event now ignores chat interceptors during conversation
- tame objective now works with all tamable mobs, including possible future ones
- improved chestput waring for locations without a chest

Deprecated🔗

Removed🔗

- Removed Deprecated Exceptions
- Removed RacesAndClasses support
- Removed LegendQuest support
- Removed BoutifulAPI support
- Removed the CLAY NPC
- removed legacy material support

Fixes🔗

- linebreaks in strings
- notify:1 for block objective did not work
- asynchronous database access for objectives
- Renaming an NPC will not cause an NPE for a NPC Hologram
- Objective 'craft' now supports shift-clicking
- Fixed generation of default package
- fixed line breaks

- fixed events notify interval of 1
- fixed potion/brew objective notify
- fixed workaround when chest converationIO has no available start points
- fixed journal line breaking
- fixed movement of movenpc event
- fixed npcmove event
- fixed bug, where a player causes an exception when he spams right left clicks in menu conversationIO
- fixed outdated Brewery dependency
- fixed message dublication when using the packet interceptor
- fixed Journal interaction with Lectern
- fixed QuestItems ignoring durabilty
- fixed QuestItem interaction with Lectern, Campfire and Composter
- fixed journal update after magic inventory is closed
- fixed lever event not toggling the lever
- fixed ConcurrentModificationException in PlayerData

Security🔗

- fixed issue, where objectives that count things are out of sync with the database. This has also affected BungeeCord support

[1.11.0] - 2020-01-02🔗

Added🔗

- Support Minecraft 1.8 - 1.13.2+
- New Block Selector to select blocks by material and attributes. Can use wildcards as well.
- New 'mooncycle' condition - Determine what phase the moon is in
- Chest ConversationIO can now be configured to show NPC text per option.
- New 'extends' keyword in conversation to allow inheritance
- New 'conversation' condition that will return true if there is at least 1 conversation option available to an NPC
- New 'nujobs_canlevel' condition - True if player can level in Jobs Reborn
- New 'nujobs_hasjob' condition - True if player has job in Jobs Reborn
- New 'nujobs_jobfull' condition - True if a job is full in Jobs Reborn
- New 'nujobs_joblevel' condition - True if player has level in Jobs Reborn
- New 'nujobs_addexp' event - Add experience to player in Jobs Reborn
- New 'nujobs_addlevel' event - Add a level to player in Jobs Reborn
- New 'nujobs_dellevel' event - Remove a level from player in Jobs Reborn
- New 'nujobs_joinjob' event - Joins a player to a job in Jobs Reborn
- New 'nujobs_leavejob' event - Leaves a job in Jobs Reborn
- New 'nujobs_setlevel' event - Set a players level in Jobs Reborn

- New 'nujobs_joinjob' objective - Triggers when player joins job in Jobs Reborn
- New 'nujobs_leavejob' objective - Triggers when a player leaves job in Jobs Reborn
- New 'nujobs_levelup' objective - Triggers when a player levels up in Jobs Reborn
- New 'nujobs_payment' objective - Triggers when a player is paid in Jobs Reborn
- New Notification System
- New 'notify' event - Create custom notifications on the ActionBar, BossBar, Title, Subtitle and Achievement
- New 'menu' conversation IO - Requires ProtocolLib. See: https://www.youtube.com/watch?v=Qtn7Dpdf4jw&lc *(https://www.youtube.com/watch?v=Qtn7Dpdf4jw&lc)*
- New 'packet' chat interceptor - Requires ProtocolLib.
- new '/q debug' command - Enable or disable the debug mode

Changes🔗

- Event 'effect' can have 'ambient', 'hidden' and 'noicon' parameters
- Event 'effect' has '--ambient' parameter deprecated with a non fatal warning.
- Priority for 'journal_main_page' entries not unique anymore, it only orders the entries. Same priority sort it alphabetic
- Objective 'interact' can have 'loc', 'range' parameters
- Objective 'region' can optionally have 'entry' and/or 'exit' to only trigger when entering or exiting named region
- The old 'Debug' class was replaced by a more useful and powerful 'LogUtils' class

Fixed🔗

- Resolve variables in journal pages.
- WATER and LAVA can be specified in Action Objective
- Journals without dates now don't leave blank lines
- Journal separator can be disabled or customized
- NPCs now spawn correct, if they have a npc_hologram
- fixed NPE when no journal entry exists
- The default package is now compatible with all versions

[1.10] - 2019-09-16🔗

- Development versions can be full of bugs. If you find any, please report them on GitHub Issues.
- This version is only compatible to Shopkeepers v2.2.0 and above

Added🔗

- npc holograms above the head that follow the npc (requires HolographicDisplays)
- New 'facing' condition - check if player is facing a direction

- New 'looking' condition - check if player looks at a block
- New 'deleffect' event - delete potion effects of a player
- New '%citizen%' variable - display a npcs name or coordinates (requires Citizens)
- New 'npcrange' objective - player has to go towards a npc (requires Citizens)
- New 'npcdistance' condition - check if a player is close to a npc (requires Citizens)
- New 'npclocation' condition - check if a npc is at a location (requires Citizens)
- New 'npcregion' condition - check if a npc is inside a region (requires Citizens & WorldGuard)
- New 'killmob' event - remove the mobs that you spawned with 'spawn' event
- New '/q version' command - get the version used
- New 'partialdate' condition - check if the date matches a pattern
- New 'dayofweek' condition - check if its weekend or monday
- New 'realtime' condition - check if its a specific time
- New 'xp' event - give a player xp.
- Global objecties (objectives that are active for all players directly after start)
- Global tags and points (tags ad points that are not set for one specific player)
- New 'globaltag' event
- New 'globaltag' condition
- New 'globalpoint' event
- New 'globalpoint' condition
- New 'opsudo' event - Sudo commands with op permissions
- Brewery integration ('drunk', 'drunkquality' and 'hasbrew'conditions, 'givebrew' and 'takebrew' events)
- New 'title' event - display titles without the whole command hassle
- New 'playsound' event - plays a sound
- New 'fly' condition - check if the player is flying with Elytra
- New 'biome' condition - check the player's current biome
- New 'interact' objective - interact with an entity
- Conversations can individually override conversation IO type
- NPCs can be individually hidden from players if ProtocolLib is installed

Changes

- 'compass' event can now directly set a players compass
- holograms from HolographicDisplays now can display items
- 'movenpc' event now allows multiple locations to create a path
- 'enchant' objective now allows multiple enchantments
- 'particle' event can now create client side only particles
- 'chest' converstionIO now dosn't display messages to chat for the old behaviour use 'combined'
- 'money' event can now notify you about how much you recieved
- 'mmobkill' objective now allows multiple mobs
- Translation system is integrated with BetonLangAPI
- NPC heads in "chest" conversation IO will display correct Citizens skin

- NPC particles (EffectLib integration) can be displayed to individual players
- Condition command allows checking static conditions
- 'testforblock' condition can now check for specific data value
- 'delay' objective and 'folder' event accept more time units
- 'password' objective also accepts commands
- Commands can be tab-completed

Fixed

- Fixed bug where players could take out items from the chest conversationIO
- Removed possibilities of dropping/transfering quest items and the journal
- Lots of smaller bugfixes

[1.9.6] - 2017-11-27

Fixed

- Update version to 1.9.6

[1.9.5] - 2017-11-27

Fixed

- Fixed global locations loading before the worlds
- Fixed loading order of Citizens/EffectLib integration
- Fixed restarting of persistent objectives not working correctly
- Fixed "unbreakable" tag not being read from items

[1.9.4] - 2017-11-02

Fixed

- Fixed broken integration loading

[1.9.3] - 2017-11-01

Fixed

- NPC and mob kills will be correctly registered when killed by indirect means
- Replaced error with a nice message when config updating fails to start
- Unbreakable items are no longer breakable in newer Spigot releases

- Moved compatibility hooks to the first server tick to hook into lazy plugins
- Colors of text in "chest" conversations are now correctly applied over text breaks
- Added a nice message when conversation option is missing "text"
- Fixed a rare crash when NPC was stopped and its target was outside of loaded chunks
- Fixed checking item amounts in the backpack
- Allowed negative data in items for compatibility with dark magics
- Removed Denizen script checking, since it didn't work sometimes

[1.9.2] - 2017-07-09🔗

Fixed🔗

- Conversations won't allow taking items from GUI windows
- When using wrong 'point' or 'item' variable there will be a nice error message
- NPCs can be safely despawned while in the middle of a conversation
- Error on '/q reload' when NPC particles are disabled is now gone
- Items for compass buttons are now correctly loaded

Changes🔗

- These events are now correctly persistent: clear, explosion, lightning, setblock, spawn
- BetonQuest is using bStats instead of McStats

[1.9.1] - 2017-04-18🔗

Fixed🔗

- Holograms are now correctly loaded

[1.9] - 2017-04-03🔗

Notes: - This version breaks compatibility with plugins hooking into BetonQuest. I'm sorry for that. Ask devs to update these plugins. - The error reporting feature was improved. If you see a lot of error messages when reloading the plugin (not stack traces, just regular, human-readable messages), it's probably because there are real problems in your quests. - BetonQuest won't accept ".yml" extensions at the end of conversation names in "main.yml". If your conversations aren't working (the plugin says they don't exist), check if you have these extensions IN THE "MAIN.YML" file and remove them.

Fixed🔗

- 'action' objective now detects fire interaction

- 'empty' condition now skips armor and off-hand slots
- Items can be used cross-package
- New sound names are now used by default
- Fixed doubled quest items when dropping them is blocked by another plugin
- Lore and name now appear on heads and written books with custom data
- Fix error when trying to add air (empty hand) with "/q item" command
- Main page now can exceed a single page in the journal
- The plugin will reconnect to the database if something goes wrong
- Fishing objective now only accepts stuff from water
- Properties in 'mobkill' objective (left and amount) has switched places

Changes🔗

- Complete rewrite of item conditioning - read the docs to discover new features (previous syntax is still working without any behavior changes)
- Books in items.yml now automatically wrap pages, like the journal and main page
- Main page and entries in the journal can manually split pages with '|' character
- New lines in conversations can be made with "\n"
- Interval in 'delay' objective is now configurable
- 'craft' and 'potion' objectives now use items defined in items.yml file
- Potion items are now defined with 'type:' argument instead of data value
- You can now use spaces between "first" options in conversations
- Static events can now be fired with "/q event - eventID" command
- Locations can have vectors defined directly in instruction strings
- Locations can be variables which resolve to location format
- Point condition can now check exact point amount with 'equal' argument
- In 'chest' conversation IO items can be specified with durability values after a colon
- Mobs spawned with 'spawn' event can have armor, items in hands and custom drops
- Unbreakability of quest items can be disabled (if you want to use "unbreakable" tag instead)
- Ranges in locations are now a separate argument ("10;20;30;world;4" is now "10;20;30;world 4")
- "main.yml" is now the only required file in the package. Empty files can be deleted
- Custom settings (i.e. EffectLib particle effects) are moved from "main.yml" to "custom.yml"

Added🔗

- Compatibility with Shopkeepers ('shopkeeper' event, 'shopamount' condition)
- Compatibility with PlaceholderAPI ('ph' variable and 'betonquest' placeholder)
- Compatibility with HolographicDisplays (holograms visible based on conditions)
- Compatibility with RacesAndClasses (race, class, exp, level, mana conditions/events/ variables)
- Compatibility with LegendQuest (race, class, attribute, karma conditions/variables)
- Compatibility with WorldEdit ('paste' a schematic event)

- New condition 'riding' - check if the player is riding an entity
- New condition 'world' - check the world in which the player is
- New condition 'gamemode' - check player's game mode
- New condition 'achievement' - check if the player has an achievement
- New condition 'variable' - check if a variable matches a pattern
- New event 'lever' - switches a lever
- New event 'door' - opens/closes doors, trapdoors and gates
- New event 'if' - run one of two events, depending on condition
- New event 'movenpc' - move Citizens NPC to a location
- New event 'variable' - set a variable in "variable" objective
- New objective 'vehicle' - entering a vehicle entity
- New objective 'variable' - lets players define their own variables for you to use
- New objective 'kill' - kill players who meet specified conditions
- New objective 'breed' - breed animals (only 1.10.2+)
- New variable '%location%' - resolves to player's location
- Keyword "unbreakable" can be used in items to make them unbreakable
- When a conversation option is selected, a Bukkit event is called (for developers)
- Chat can be paused while in conversation, it will display when finished
- Objectives can be completed for players with "/q objective player complete"
- Option 'full_main_page' controls if the main page is a separate page in the journal
- Mobs spawned with 'spawn' can be "marked"; you can require marked mobs in 'mobkill' objective
- Firework support in items
- Relative package paths, where '_' means "one package up"

[1.8.5] - 2016-05-14

Fixed

- Objectives are now correctly deleted with "objective delete" event and do notreappear after "/q reload".
- Objectives are no longer duplicated in the database when using "/q reload".

[1.8.4] - 2016-05-06

Fixed

- Conversations are no longer started twice

[1.8.3] - 2016-05-06🔗

Fixed🔗

- Events are no longer run in async thread when completing "password" objective
- Replaced stacktrace with error message when objective is incorrect in "objective" event
- Made color codes work with "one_entry_per_page" setting enabled
- Fixed a bug where taken backpack items were not removed from the database
- Quest items can now be equipped
- "die" objective now correctly handles damage done to the player
- Fixed error when conversation is started without any possible options
- Fixed error when killing NPCs with equipment
- Fixed problems with relogging while in conversations with "stop" option enabled
- Fixed error when loading corrupted item from the database

Changes🔗

- Updater is now based on GitHub Releases, no longer downloads major updates automatically, it is more configurable and can also download development versions with "/q update --dev"

Added🔗

- Added console message about the cause of "/q give" errors (tells you what is wrong with item instruction string)

[1.8.2] - 2016-02-18🔗

Fixed🔗

- Fixed NPE when killing a mob without any "mobkill" objectives

[1.8.1] - 2016-02-18🔗

Fixed🔗

- Removing journal entries from the database now works correctly
- MobKill objective now correctly handles kills
- Nested package names are now correctly resolved
- The formatting at the end of every main page line is reset
- Fixed Apache dependency problem

• Material name is no longer displayed in "chest" GUI conversations
• Fixed "notify" option in give/take events

[1.8] - 2016-02-13🔗

Notes: - As always in big updates, compatibility with plugins hooking into BetonQuest is broken. You need to check if everything is working.

Fixed🔗

• Die objective now reacts to death caused by other plugins
• Static events now are started correctly
• Static events now are canceled correctly
• Action objective now correctly checks locations
• Combat tag is removed after death
• Block, Craft and MythicMobs MobKill objectives now correctly save data
• Take event now correctly takes items from inventory, armor slots and backpack

Added🔗

• New variable system in conversations (check out the documentation)
• More options for journal, including one entry per page and removing date
• Compatibility with mcMMO (level condition and experience event)
• Compatibility with EffectLib ('particle' event, NPC particles)
• Compatibility with PlayerPoints (points event and condition)
• Compatibility with Heroes (class and skill condition, experience event, Heroes kills in 'mobkill' objective)
• Compatibility with Magic ('wand' condition)
• Compatibility with Denizen (running task scripts with 'script' event)
• Compatibility with SkillAPI (class and level condition)
• Compatibility with Quests (checking for done quests, starting them, custom event reward, custom condition requirement)
• Optional prefix for conversations (contributed by Jack McKalling)
• Optional material for buttons in "chest" conversation IO
• Configurable main page in the journal
• New argument in objectives: "persistent" - makes them repeat after completing
• New condition 'check' - allows for specifying multiple instructions in one
• New condition 'objective' - checks if the player has an active objective
• New condition 'score' - check scores on scoreboards
• New condition 'chestitem' - checks if a chest contains items
• New event 'run' - allows for specifying multiple instructions in one
• New event 'givejournal' - gives journal to the player
• New event 'sudo' - forces the player to run a command

- New event 'compass' - point player's compass to a location
- New event 'cancel' - cancels a quest (as in main.yml)
- New event 'score' - modify scores on scoreboards
- New events 'chestgive', 'chesttake' and 'chestclear' - put and remove items in chests
- New objective 'logout' - the player needs to leave the server
- New objective 'password' - the player needs to type the password in the chat
- New objective 'fish' - catching fish
- New objective 'enchant' - enchanting an item
- New objective 'shear' - shearing a sheep
- New objective 'chestput' - putting items in a chest
- New objective 'potion' - brewing a potion
- New commands: /cancelquest and /compass - directly open backpack sub-pages
- New subcommand '/q delete' - delete all specific tags/points/objectives/entries
- New subcommand '/q rename' - rename all specific tags/points/objectives/entries
- New subcommand '/q give' - gives you an item from items.yml

Changes

- Administrative messages are now English-only in new installations
- Journal event can remove entries from the journal
- In conversations, %quester% variable changed to %npc%
- In inventory GUI there is NPC's text in every option, for convenience
- Conversations can point to NPC options in other conversations within the package
- You can use spaces between events, conditions and pointers in conversations
- All tags and points are internally associated with a package now
- Some conditions are now static and persistent (just like events)
- Point event can now multiply points
- Vault Money event can now multiply money
- Journal event can now use "update" argument for updating variables on the main page
- Packages can now be moved to another directories
- Quest cancelers are now defined in a more convenient way
- /q command renamed to /betonquest, /j to /journal; previous forms are now aliases
- Conditions and events in objective instructions (and conditions in event instructions) can now be defined with "condition:" and "event:" argument (without "s" at the end)

[1.7.6] - 2015-10-17

Fixed

- Conversation can no longer be started multiple times at once if it happens on the same tick

Added🔗

- Dutch translation by Jack McKalling

[1.7.5] - 2015-09-12🔗

Fixed🔗

- Restored compatibility with MythicMobs 2.1.0

[1.7.4] - 2015-08-29🔗

Fixed🔗

- Fixed error when player was quitting with active "stop" conversation while he had not changed his language with /ql command

Changes🔗

- Inventory GUI will close itself if there's nothing left to display

[1.7.3] - 2015-08-20🔗

Fixed🔗

- Combat tagging does not work if the attack has been canceled

Changes🔗

- Options in conversation can also be defined using "event:", "condition:" and "pointers:" argument names (with and without 's' at the end). "text:" argument is unchanged.

[1.7.2] - 2015-07-27🔗

Fixed🔗

- "mobkill" objective now displays correct amount of mobs left to kill
- "delay" objective can be set to 0 delay

[1.7.1] - 2015-07-19

Fixed

- Quests are loaded after other plugins register their types
- Journal condition correctly resolves package names

Changes

- Updated French translation

[1.7] - 2015-07-17

Notes: - BetonQuest no longer supports servers without UUID handling - There were a lot of changes since previous version, check carefully if everything is working - Compatibility with plugins hooking INTO BetonQuest is broken, they need to update

Fixed

- Objectives no longer mysteriously double events
- Greatly improved performance in almost every aspect
- Finally fixed issues with special characters on some servers
- Fixed database saving/loading issues
- Fixed player options in conversations being white on next lines when using tellraw

Added

- Quest canceling system
- New inventory GUI for conversations
- Added the "random" parameter in "folder" event - choose randomly X events to fire
- Action objective can be "canceled" - the click will not do anything
- Added "static events" mechanism for firing events at specified time of the day
- Optional message when the player is pulled back by stop option
- Optional message for take and give events
- Optional message when advancing in "block" and "mobkill" objectives
- Variable system for quick changing quest parameters (for example location of a quest)
- "/q vector" command for easy calculating location vector variables
- New "empty" condition - amount of empty inventory slots
- New "party" condition - manages the conditions in the party
- New "monsters" condition - true if there are monsters in the area
- New "clear" event - kills specified monsters in the area
- New "region" objective - reach WorldGuard region
- Blacklist of commands which cannot be used while in conversation

- Option to disable compatibility with other plugins
- Added remove_items_after_respawn option - for servers using keepInventory gamerule

Changes🔗

- The plugin now uses package system: configuration has been moved into "default" package
- Objectives has returned to "objectives.yml" - it's improving performance
- The database is now updated in real time
- All quests can (but don't have to) be translated into multiple languages
- Players can change their language with /questlang command
- Conversations with stop option are resumed when the player logs out and in again
- Metrics are now toggled in PluginMetrics/config.yml
- All conditions, events, objectives, conversations etc. are loaded when the plugin starts/ reloads
- Citizens NPC will stop when talked to
- Quest blocks cannot be placed, quest items will not break
- Conversations cannot be started while in combat
- Cannot fight while in conversation
- Tellraw conversations no longer spam the console
- Mobs can be spawned with a name (spawnmob event, "name:" argument)
- /q command is now more beautiful
- Removed unnecessary argument prefixes from conditions and events
- Removed "tag:" from objective instruction strings
- Conversations no longer need those empty lines everywhere ('')
- Dependencies updated: WorldGuard/WorldEdit 6.1, MythicMobs 2.0.4

[1.6.2] - 2015-04-10🔗

- Fixed errors on data loading when MySQL is being used.
- Changes messages system to use simple file as default. If you want to use advanced translation just rename "advanced-messages.yml" to "messages.yml".

[1.6.1] - 2015-03-26🔗

- Fixed errors on updating journals when using MySQL.

[1.6] - 2015-03-16🔗

Notes: - There is a bug/feature in 1.8 which adds '§0' at the end of every line in books generated by plugins. This breaks the conditions/events based on books with more than one line of text.

The detailed instruction on how to work it around is in "Other important stuff" chapter, in the part about items.

Fixed

- Items given by event that don't fit in the inventory will now drop instead of being deleted This does not apply to quest items, they will be added to backpack
- Events fired from conversations won't throw async errors
- Conversation can be started after plugin's reload without relogging
- /q reload no longer lags the server
- Corrected description in /q command
- Added input validation for global locations - if event is incorrect it will display an error instead of breaking the whole functionality
- The plugin should run fine on machines not supporting some special characters
- Inverted item condition now behave correctly
- Time condition now checks time correctly

Added

- Added backpack for storing quest items, which cannot be dropped in any way
- Added database backups
- Added prefix for the database. New installations will use "betonquest_" prefix for tables, existing configuration will use empty prefix to maintain compatibility with other programs
- Players can chat while in conversations by prefixing their messages with '#' character
- New "random" condition - true with specified probability
- New "sneak" condition - true if player is sneaking
- New "journal" condition - true if player has journal entry
- New "testforblock" condition - true if block at given location matches given material
- New "arrow" objective - completed when arrow hits the specified target
- New "experience" objective - completed when player reaches certain level
- New "npcinteract" objective - completed when player right-clicks Citizens NPC
- New "damage" event - damages the player
- Skript support (event, effect and condition)
- WorldGuard support (region condition)
- Errors are logged to the "error.log" file in "logs" directory
- Debug option in config.yml for logging plugin's activity to "debug.log" file
- New commands for opening backpack: b, bb, backpack, bbackpack or betonbackpack
- Items are now aware of leather armor color, head owner and enchantments in books

Changes

- Added and changed a lot of subcommands in /q command:
    ◦ event and condition can be run for every online player
    ◦ tag, point, objective and (new) journal can edit every (even offline) player

- config (new) can set configuration files from command line
    - backup (new) backups the whole configuration and database
- Folder event now runs these events even after the player logs out: command, tag, objective, delete, point, setblock
- Changed /j command to open the backpack instead of just giving the journal
- Tellraw clicking on options in conversation now ignores old (used) options
- Using color codes in journal entries is now possible
- Give/take events and item condition can now check for multiple items with syntax 'give stick:2,stone:4,sword'
- Give/take events and item/hand conditions can now check for items only without enchantments/effects/name/lore etc.
- Inverting condition is now done by prefixing their name with "!" (in the place where you use them, like conversation, not in conditions.yml)
- Configuration updater is no longer based on plugin's version
- Backup files are now kept in "backups" directory, old ones are moved to it
- Changed internal structure of the code (may matter to developers - QuestEvent, Condition and Objective classes has been moved from "core" package to "api", update your imports)

[1.5.4] - 2015-03-12🔗

- This version is almost the same as 1.5.3. The only difference is that it can load database backups created by 1.6 version. When updating to 1.6, the database format will change, so it won't be possible to go back, unless by loading the backup using this version of the plugin.

[1.5.3] - 2014-12-26🔗

- Small fix of /q purge command not working on offline players.

[1.5.2] - 2014-12-23🔗

- Fixed errors that were spamming the console when a player with active Location objective was teleporting to other worlds.

[1.5.1] - 2014-12-22🔗

Changes🔗

- Multiple tags in one event are now possible
- Change /q event command to run from console
- Add color codes to item's name and lore

- Fix "stop" option in conversations not working
- Fix NPE on unknown answer in conversations

[1.5] - 2014-12-21🔗

Changes🔗

- Added support for MythicMobs and Vault (see wiki for more info)
- AutoUpdater is now enabled by default! If you want you can change this and reload the plugin, nothing will be downloaded in that case
- Books saving format has changed. All books were automatically converted, but you need to check them if everything looks like it's supposed to.
- Command event accepts multiple commands separated by "|", eg. "command say beton| say quest"
- Event command now accepts optional argument at the end; this will fire event for player. eg. "/q event wood_reward Steve"
- Journal title and lore can now use colors (&4 etc.) and journal is colorful; options in config.yml
- Added aliases for /q command: bq, bquest, bquests, betonquest, betonquests, quest, quests
- Added aliases for /j command: bj, journal, bjournal, betonjournal
- Objectives are now defined directly in event instruction, not in objectives.yml (which was deleted, if you want to restore something check the backup)
- Replies in conversations are now optionally clickable (tellraw option in config.yml)
- Added permission for starting a conversation: betonquest.conversation
- Conversation starting/ending, updating journal, plugin's update and full inventory can now make sounds; you can find a list of possible values here: jd.bukkit.org/rb/apidocs/org/bukkit/Sound.html
- Conditions for events are now defined as 'event_conditions:' instead of simply 'conditions:'. This is to distinguish conditions for objectives and for events, as both of them can exist in one instruction
- Updater is now run when disabling the plugin (it does matter if your server restarts every night) Notes:
- All Objective events has been converted to new format. The objectives.yml file has been deleted, so if it contained any objectives that weren't covered by an event they may seem lost. However there is a backup file and you can easily extract everything from it. Please refer to the wiki to learn how objectives are now defined or just study converted ones (it's pretty straightforward).
- AutoUpdater is now enabled by default. Every future update will be working exactly like before, all changes will be automatically updated by a converter, there is always a backup and you are informed about all changes in this file. So it's pretty safe to say that keeping this plugin up to date won't give you any trouble. If you don't want to have latest fixes and features you can disable updating but this will make the developer sad.

- Because of changes in how books behave since 1.8 you may experience some strange bugs with saving books to items.yml. Generally you should open a book before saving it using /q item command. And don't start or end your books with " character, as it's part of a workaround of this bug/feature.

[1.4.3] - 2014-12-15🔗

- Removed debug messages from ActionObjective. You could have told me, any of you guys...

[1.4.2] - 2014-12-09🔗

- Really fixed an updater.

[1.4.1] - 2014-12-09🔗

- Fixed few bugs in Action objective.
- Fixed updater, hopefully.

[1.4] - 2014-12-07🔗

Changes🔗

- Conversations are now divided into multiple files in "conversations" directory
- Items are now saved to items.yml file and referenced by "take", "give", "item" and "hand" events/conditions
- Added /q item command which saves currently held item to the config as specified itemID
- Added location to Action objective, which checks the location of the block (unlike location condition which checks location of the player)
- Added /q event command which fires specified event
- Fixed multiple bugs with conversation starting and ending
- Block NPCs can now be used with Citizens enabled
- Added NPCKill objective for killing NPCs
- Added SetBlock event for setting a block at specified location
- Improved Material matching in configs
- Modified Action objective for greater flexibility:
    ◦ It is now possible to detect clicking in air
    ◦ It is no longer possible to detect clicking on any block (as this accepts clicking on air)
    ◦ Can be used to detect book reading (with help of updated Hand condition)
- Added AutoUpdater; it's disabled by default Notes:

- Conversion of configuration should have been done automatically, you don't have to worry about anything. If something went wrong you can revert changes from generated backup file, which contains all your previous configs.
- You can enable AutoUpdater by setting "autoupdate" to true in config.yml. It is completely safe because all next versions will generate backups and convert all files automatically. You will be notified on joining the server about new changelog file.
- Please refer to the wiki for changes in formatting instruction strings for various things: https://github.com/Co0sh/BetonQuest/wiki *(https://github.com/Co0sh/BetonQuest/wiki)*
- You probably should also change names of converted items to something else than "item12". But that works too of course.

[1.3] - 2014-11-30🔗

Changes🔗

- UUID support (optional)
- NPCs made from a clay block, head and sign, for servers without Citizens2 plugin
- Global, long and persistent delay for events (as an objective)
- Folder event for multiple events, with optional short delay
- French translation (thanks to fastlockel)
- If you want to convert names to UUIDs run the plugin once and then change in the config "uuid: false" to true. Do not touch the "convert: true" option unless you want your database wiped! Conversion will happen on next plugin reload (eg. /q reload). This is not revertable!
- Remember to backup your config files before updating! It shouldn't destroy anything but you never know.

[1.2] - 2014-11-23🔗

- Global locations now automatically run only once, no need for blocking it with tags and conditions. They use however tags that follow the syntax "global_", where is global location objective tag.
- Added optional respawn location for cancelled death objective, just add "respawn: 100.5;200;300.5;world;90;0" to instruction string.
- Added German translation, thanks to coalaa!
- Added optional movement blocking while in conversation, just add option "stop: true" or "stop: false" in every conversation.
- Changed priority of conversation chat event to lowest, should work even for muted players.
- Fixed data values in block objective.
- Added metrics, you can disable them by setting "metrics: false" in config.yml
- Added support for SQLite, plugin will use it when connecting to MySQL fails.
- Fixed death objective not working every time and not removing all effects.

[1.1] - 2014-11-08🔗

- Fixed many bugs including but not limited to:
  - negated conjunction condition
  - unnecessary debug messages
  - not working global locations
- Replaced config examples with default quest
- Leaving data values in item's definition will make plugin ignore data value in most cases
- Improved journal to stop text leaks
- Item names now replace _ with spaces

[1.0] - 2014-11-06🔗

- Initial release

Last update:

Developer Documentation

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Developer-Documentation/
Contributing-Guidelines.md)

Contributing Guidelines🔗

The BetonQuest organisation recommends *IntelliJ Community Edition (https://*
www.jetbrains.com/idea/) as the preferred IDE. This makes it possible to ship a complete project
setup that ensures a consistent code quality and style. The project has been setup using maven
and will build immediately on your device without any extra work (like manually adding
dependencies).

Everyone that wants to use another IDE can do so but has to ensure that their code meets the
guidelines. Notice that everything that is prescribed in the Coding Guidelines will be checked by
the Build-Pipeline. The Build-Pipeline will not accept commits that fail to respect the guidelines.

Walkthrough of the Guidelines🔗

Setup🔗

The first thing you should do is to clone the project from our GitHub repository:

```
git clone https://github.com/BetonQuest/BetonQuest.git
```

Make sure to set this git config setting in your project before cloning or touching any code:

Windows:

```
git config --global core.autocrlf true
```

MacOS & Linux:

```
git config --global core.autocrlf input
```

Get ready to create a Pull Request!🔗

Test your changes!🔗

Please ensure your changes actually work and haven't introduced any bugs. Therefore, you
should compile the plugin and run it on a local 1.13+ test server. Please install any related

dependencies! If you changed code related to MySQL then setup a MySQL on your machine and test if BetonQuest still saves its data correctly.

Does your change require updates to users configs?🔧

This would be the case if you removed arguments or renamed e.g. an event. You then have to write code to update any related configs to your new syntax. Search for the ConfigUpdater class in the project to see how its done.

Validate your codes quality🔧

Please check you code with PMD. IntelliJ's plugin store offer the *QAPlug & QAPlug - PMD* plugins for free. We provide our own ruleset for PMD (*config/pmd-ruleset.xml*) that needs to be imported in QAPlug. To do so open IntelliJ's settings and navigate to QAPlug -> Coding Rules. Click the plus and select Project Profile. Now you just need to tick "Import Profile" to add our ruleset (*config/pmd-ruleset.xml*).

Please also remove any TODO's along with commented out code.

You can run `mvn verify` locally to ensure that you meet (nearly) all conditions before creating a PR.

Update the Changelog🔧

You need to add your changes to the *Changelog.md* file that can be found in the projects root folder.

Update the documentation🔧

You also need to document changes or additions to the plugins feature set in our documentation. Please notice that the documentation also has it's own guidelines.

Clean the commit history🔧

You need to have a clean commit history in order to get you PR accepted. Check out this guide *(https://medium.com/@catalinaturlea/clean-git-history-a-step-by-step-guide-eefc0ad8696d)* if you don't know how to do so.

Last update:

API

Accessing the plugin🔗

Add this to your pom.yml.

```
<repositories>
    <repository>
        <id>betonquest-repo</id>
        <url>https://betonquest.org/nexus/repository/betonquest/</url
    </repository>
</repositories>

<dependencies>
    <dependency>
        <groupId>pl.betoncraft</groupId>
        <artifactId>betonquest</artifactId>
        <version>1.12.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

Writing events🔗

Writing events is the easiest. You need to create a class extending `QuestEvent` for each new event. The constructor must take one argument, an `Instruction` object. In the constructor you must extract all information from the instruction, for example skill names, locations etc. The description of the `Instruction` class is down below. Don't worry about checking event conditions, these are handled by the rest of BetonQuest's logic.

Events are not bound to any player so firing it is done through `fire(String playerID)` method. You have to override it with your code responsible for doing stuff your event should do. Here you should use data previously parsed by the constructor. Don't access `Instruction` object here, it will lower the performance. You can convert `playerID` to `Player` object using the `PlayerConverter` class (it's a relict of times when both UUIDs and names could be used in Bukkit to identify players).

If you want your event to be *persistent*, you need to set `super.persistent` variable to `true` in the constructor. This will make BetonQuest run this event even if the `playerID` points to an offline player, so prepare your code for that.

If you want your event to be *static*, you need to set `super.staticness` variable to `true` in the constructor. This will allow BetonQuest to run this event with `playerID` set to `null`, so prepare your code for that.

When you'll finish your class you need to invoke `registerEvents(String name, Class<? extends QuestEvent> class)` from BetonQuest instance (which you can get using `BetonQuest.getInstance()` static method). The name for your event will be used in instruction strings (such as "journal" for journal event). The class argument is the `Class` object of your event. You can get it using `YourEvent.class`. That's it, you created an event. Don't forget to check it for bugs!

Writing conditions🔗

Writing conditions is easy too. They must extend `Condition` and override `execute(String playerID)` method, which should return `true` or `false`, depending on if the condition was met. You register them using `registerConditions(String name, Class<? extends Condition)` method from BetonQuest instance as well. The rest is almost the same, you're defining the constructor which will parse the `Instruction` object and overriding `execute(String playerID)` method to check if the player meets the condition. Don't worry about inverting it, as it's automatically done by BetonQuest.

Conditions are always getting an online player in the `execute(String playerID)` method, so you don't need to check that manually.

Writing objectives🔗

Objectives are more complicated because they use event handlers and they must store players' data. They extend `Objective` class. As always, you need to extract all data from supplied `Instruction` object in the constructor. Don't register listeners in the constructor!

If your objective handles changing data (like amount of mobs left to kill) you should create a class extending `ObjectiveData`. For example `block` objective does need to store amount of blocks left to place/break, and it does that using "BlockData" class. In the constructor it receives three strings: data string, ID of the player and ID of the objective. The latter two are used by BetonQuest to correctly save and load the former one from the database.

The data string should contains all the information you need in your objective. You must write a parser which will extract the information, methods used in the objective to alter the information, and override the `toString()` method in so it returns data string in the format parsable by your parser. Everytime the data in your object changes (like when killing a mob), you need to call `update()` method. It will save the data to the database.

Now you should override `getDefaultDataInstruction()` method. It must return the default data instruction understandable by your parser. For example in `tame` objective it will return the

amount of mobs to tame. If you don't use data objects, just return an empty string (not `null`, just
`""`).

In order for your objective to use the data object you have created you need to set the
`template` variable to this object's class. If you're not defining the data object (because you
don't need to handle the changing data), you should set the `template` simply to
`ObjectiveData.class`.

Every time your objective accepts the player's action (for example killing the right mob in MobKill
objective) it must be also verified with `checkConditions()` method. You don't want your
objective ignoring all conditions, right? When you decide that the objective is completed you
should call `completeObjective()` method. It will fire all events for you, so you don't have to
do this manually.

`start()` and `stop()` methods must start objective's listeners and stop them accordingly. It's
because the plugin turns the objective's listeners off if there are no players having it active. Here
usually you will register/unregister listeners, but some objectives may be different. For example
`delay` objective starts and cancels a runnable, instead of using listeners.

If your objective has some properties (used in variables) you should override the `String`
`getProperty(String property, String playerID)` method. At runtime, if anyone uses
`%objective.yourObjective.theProperty%` variable, BetonQuest will call that method
with `theProperty` keyword as the first argument. Using it you should parse the data of the
objective and return it as a String. If the supplied property name is incorrect or there was an
error during getting the value, return an empty String and optionally log an error
(`LogUtils.getLogger().log(...)`).

Objectives are registered the same way as conditions and events, using
`registerObjective(String name, Class<? extends Objective>)` method.

Reading `Instruction` object🔗

The `Instruction` object parses the instruction string defined by the user and splits it into
arguments. You can ask it for required arguments one by one with `next()` method or a parser
method like `getQuestItem()`. Required arguments are the ones specified at the very
beginning of an instruction string, for example `add someTag` in `tag` event. It will automaticly
throw `InstructionParseException` for you if it encounters an error, for example when there
were no more arguments in user's instruction or it can't parse the argument to the type you
asked for.

You can also ask for optional arguments: if the instruction string contains argument
`arg:something` and you ask for optional `arg`, it will give you `something`. If there is no
optional argument, it will return `null`. Don't worry about passing that `null` to parser methods
like `getLocation(String)`, they won't throw an error, they'll simply return that `null`.

Parser methods are there for your convenience. You could write a location parser for yourself, but there's no need for that, you can just use `getLocation()` or `getLocation(String)` method and receive `LocationData` object. The former method is simply `getLocation(next())`.

If your instruction is more complicated and `Instruction` class doesn't provide necessary methods, you can still parse the instruction string manually. You can get it with `getInstruction()` method. Just remember to throw `InstructionParseException` when the instruction supplied by the user is incorrect. BetonQuest will catch them and display a message in the console.

Writing variables🔗

All variables need to extend `Variable` class. In the constructor you must parse the instruction and extract all information about your variable's behavior. Then you have to override the `String getValue(String playerID)` method. It should return the value of the variable for the supplied player. If it's impossible, it should return an empty String. Registering variables is done via `BetonQuest.registerVariable(String name, Class<? extends Variable> variable)` method.

Firing events🔗

The plugin has a static method for firing events - `event(String playerID, EventID eventID)`. First parameter is ID of the player. Second one represents ID of the event. To get it, simply create an instance of the `EventID` class. You can't fire an event directly using an instruction string.

Checking conditions🔗

BetonQuest has static boolean method `condition(String playerID, String conditionID)`. It works similarly as event method described above.

Starting objectives🔗

The `newObjective(String playerID, String objectiveID)` method will launch the objective from start. You can however use `resumeObjective(String playerID, String objectiveID, String instruction)` to pass your own `ObjectiveData` instruction to the objective. It will not be saved to the database, because it is assumed that the objective has just been loaded from it and it exists there without any change. You should save it manually.

Creating additional conversation input/output methods🔗

In order to register an object as the conversation input/output it needs to implement `ConversationIO` interface. The constructor will receive three arguments: Conversation object, playerID String and NPC name String. It needs to parse the required data here and register all needed listeners. The `setResponse(String response)` method will receive NPC's text from the conversation. The `addOption(String option)` method will be called by the conversation for each reply option for this NPC text. The object must store all this data and when `display()` is called, it must use it to display the player the output. When it detects that the player chose an answer, it should pass it to the conversation using `Conversation.passPlayerAnswer(int number)` method. The integer is the number of the answer, starting at 1. `clear()` method will be called at the beginning of the new conversation cycle. It should clear all the previous options, so they do not overlap. `end()` method will be called when the conversation ends and it should unregister all listeners. You can also call that message when you detect that the player forced conversation ending (for example by moving away from the NPC). Remember to notify the conversation about that using `Conversation.end()`.

Registering the conversation inputs/outputs is done in the same way as objectives, events and conditions, through `BetonQuest.registerConversationIO(String name, Class<? extends ConversationIO>)` method.

Listening to BetonQuest (Bukkit) events🔗

BetonQuest calls Bukkit events on a few occasions: when a conversation is started, finished and when an option is selected. You can find these events in `pl.betoncraft.betonquest.api` package and use them in your plugins. If you need any additional events just open and issue on GitHub or send me a pull request.

Debugging🔗

You can debug your code using `LogUtils` class by simply call `LogUtils.getlogger().log(..)` to log something.

We use the following levels for this aspects: - `SEVER` - Anything happen, that breaks the plugin, or a main function of the plugin - `WARNING` - The most things, where something not normal or unexpected happens, but the plugin still work correctly for the rest(all catch blocks, when not `SEVER`) - `INFO` - Anything you want to log, that also should appear in the normal console - `CONFIG` - Not in use at the moment - `FINE` - All messages, that you want to only appear in the debug log file, not in the console - `FINER` - Do not use this, this is reserved, for exception

logging, like you call LogUtils.logThrowable or LogUtils.logThrowableIgnore - `FINEST` - Not in use at the moment

If you have a `catch` block, please log this with a message, that calls `exception.getMessage()` and log the complete exception by using the `Logutils.logThrowable(e)`. The methods `logThrowableReport()` and `logThrowableIgnore()` may only be executed by BetonQuest.

Last update:

Releasing🔗

This project uses a custom buildpipeline that utilizes GitHub('s) actions.

You need to follow all of the following steps for each release otherwise the build pipeline will break!

Step 1: Prerequisites🔗

Decide if the current version number in all of the files (e.g. pom.xml) is the actual version number that should be used for that release. Use Semantic Versioning *(https://semver.org/)* to do so.

Step 2: Build a release🔗

You can create a release by tagging a commit with a version tag. A version tag needs to tag the commit that should be a release with the version from the pom.xml.

Step 3: Post-Release🔗

Set a new version in the pom.xml following the Semantic Versioning *(https://semver.org/)* specification. Then edit the changelog.md in the projects root folder and replace the current `Unreleased` sections title with the just released version and date. Now copy the template from below into the file.

```
## [Unreleased] - CURRENT_DATA_HERE
### Added
### Changed
### Deprecated
### Removed
### Fixes
### Security
```

Last update:

*(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Developer-Documentation/
Versioning.md)*

Versioning

BetonQuest Versioning🔗

These are the build types that are used in the build pipeline:

1. Version tags `v*` (In the official repository)
2. Commits to `master` or branches called `master_v*` (In the official repository)
3. Commits to other branches (or repos) and Pull requests
4. Local builds

These result in a versioning like this:

1. 1.12.0
2. 1.12.0-DEV-1
3. 1.12.0-DEV-ARTIFACT-5522
4. 1.12.0-DEV-UNOFFICAL

The output jar does not contain the version in its name due to limits with the Spigot updater. Another reason is to make it clear which one of the buildpipeline output jars is the correct one for users.

Therefore, the runnable jar file is `BetonQuest.jar`. The version can only be seen Ingame by using /q version.

Last update:

Contributing

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Contributing/Contributing.md)

Contributing🔗

First of all, thank you for looking at this page!

You have numerous options to contribute to BetonQuest. Check this list to see what fits your skills:

Reporting bugs🔗

You just need to follow the bug report template on GitHub to submit a bug report. Please give us **all information that is requested in the template!**

Suggesting features🔗

Before you suggest a feature please make sure it does not already exist! Search in the docs and if you are not sure ask in the *discord! (https://discord.com/invite/rK6mfHq)* Once you are sure the feature does not already exist please also make sure nobody has already suggested it. Both things will save us time and therefore make it possible to implement more features.

Once you did that too simply *open a new issue on GitHub (https://github.com/BetonQuest/ BetonQuest/issues/new?template=feature_request_template.md)*. Maybe someday someone will find the time to make your dream come true.

Translations🔗

We love to see this plugin used by people all around the globe. Therefore, we would be happy if you could translate it to your language and share the translation with me. You can send us the edited messages.yml file via discord or submit a pull request.

Improving the docs🔗

If you want to help improve the docs in any way check the Docs Contributing Guidelines!

Writing code🔗

If you want to write code for BetonQuest check the Developer Documentation!

Last update:

Docs

*(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Contributing/Documentation/
Setup.md)*

Setup

Setup🔗

Quick Edits🔗

Quick changes can be made by clicking the pencil icon at the top of each page. This lets you edit
the docs directly in your browser. The Guidelines still apply!

Advanced Edits🔗

More advanced edits can be done with IntelliJ *(https://www.jetbrains.com/idea/download/)* or any
other IDEA. This guide will show the process with IntelliJ and includes some neat tricks that are
exclusive to IntelliJ.

Clone in IntelliJ and select the "Docs" project scope. [IMG]

Create a new branch before you start editing anything.

All the files are markdown files. Markdown is the stuff you partially know from discord text
highlighting. `**Bold text**` result `_Underlinded text_` *result*

The final docs are generated using mkdocs-material!

Make sure Python3 *(https://www.python.org/downloads/)* is installed on your local system. If you
use Python for more than this you might want to look into python virtual environments *(https://
docs.python.org/3/library/venv.html)* to avoid conflicts. This should not be the case for any non-
devs though.

Install all other dependencies by entering `pip install -r config/docs-
requirements.txt` in the console.

See your changes live🔗

Mkdocs enables you to create a website that shows you your changes while you make them. To
start an http document server on `http://127.0.0.1:8000` execute:

```
mkdocs serve
```

Last update:

(https://github.com/BetonQuest/BetonQuest/edit/master/docs/Contributing/Documentation/Guidelines.md)

Guidelines

You need to follow these rules in order to contribute to the docs. They are important for a good user experience and provide a consistent baseline for other contributors to work with.

Links🔗

Internal🔗

Links to internal pages can be opened in the same tab. This works with the normal markdown link syntax:

```
1    Click the [highlighted words](Contributing.md).
```

Result: Click the highlighted words.

External🔗

Links to external sites must be opened in new tabs using this html code:

```
1    <a href="https://betonquest.pl/" target="_blank">Clickable text t
```

Result: Clickable text that opens a new tab *(https://betonquest.pl/)*

Displaying (YAML) code🔗

You must use codeboxes with the specific language set in the header (YAML in this example). You also have to enable line numbers by adding the `linenums="1"` argument.

```
1    ``` YAML linenums="1"
2    use: "codeboxes for code"
3    ```
```

Result:

```
1    use: "codeboxes for code"
```

Refering to YAML elements in written text🔗

An example of this would be to reference an event name in an explanation.

```
1    The event `someEvent` prints a message to the player!
```

Result: The event `someEvent` prints a message to the player!

File names🔗

Replace all spaces in file and folder names with `-`!

Lists🔗

Lists must be declared as such:

```
 * Top Level
    - Second Level
    - Second Level
 * Another Top level
```

Result:

- Top Level
  - Second Level
  - Second Level
- Another Top level

Line length🔗

A single line may only have 170 characters. Please wrap at 121 though.

Last update: