Tutorial 2 - Cuboid Tools

Table Of Contents

This tutorial will treat the following concepts:

- Simple command script mechanics.
- If commands and conditions.
- Stopping the queue.
- Complex items with data keys and flags.
- Building formatted text from input.
- Nesting tags.
- Custom event switches.
- Special item event switches.
- Simple player and server flags.
- Basic list handling.
- The choose command.
- Basic cuboid tools and usages.
- Converting flag values into tag types.

Introduction

While some might prefer to directly jump into some fun projects, it's been proven useful to move in angles in order to reach a goal. That's why this second **Denizen2Sponge** tutorial will be focused on preparing some tools that will be needed later on.

More specifically, we'll be scripting utilities related to **cuboids**. These will save us a lot of work everytime we need to use cuboids in another script.

Cuboid Wand

What is this? It's just an ingame item that will generate cuboids for us just by simply clicking the two desired opposite corner blocks. We will need these cuboids for our next tutorial. First of all, we'll choose which item will serve as a wand. In our case, a shiny **blaze rod** will probably do. We want to have easy access to this wand, so we'll make a command for it.

Let's make a new script and build a command script container. We specify a name, set the *type:* key to *command*, *debug:* to *full* and then place a *name:* key. Command scripts accept multiple names (or aliases), so we'll take advantage of that including both *cuboidtool* and the shortened version *ct*. We should also specify a clear yet simple description, like *Gives yourself a cuboid wand.*, and then open an empty script section.

Our brand new command script container should be similar to this one:

```
Cuboid_Tool_Command:
  type: command
  debug: full
  name:
  - cuboidtool
  - ct
  description: Gives yourself a cuboid wand.
  script:
```

Since commands can be executed through the console, which can't receive items, we'll restrict this command's usage to only players. We can achieve that by checking the command **source** and then giving an error message and **stopping the queue**. Ifs are useful script commands that execute a script block depending on the outcome of a condition, so we can use one.

Command scripts, like events, have some special context tags with information linked to them. According to the documentation, *<context.source>* will return either *server*, *player*, *block* or *minecart*, so it's exactly what we needed. Just add an operator and a comparing value, *!= player* in our case.

Now we want to print an error in console, so we'll use a simple *- echo*. As we learnt on the first tutorial, echo only needs a message argument, and *This is a player command only!* will do. We also want to stop the running queue, and the command for that is just a *- stop*.

Our script section should look like this:

```
  - if <context.source> != player:
    - echo "This is a player command only!"
    - stop
```

We also don't want every player to have access to this command, so we need to make another similar check. Since **Sponge** doesn't have **op** players, we'll check if the player using the command is in **creative gamemode** for now. We can retrieve the player's gamemode through the simple *<player.gamemode>* tag, and then check if it's not equal to *creative* inside an if.

In this case, we want to send an error message directly to the player, so we'll use the *- tell* command, with the player as first argument and the message as second. Don't forget to also stop the queue afterwards!

This if block should look like the following:

```
- if <player.gamemode> != creative:
  - tell <player> "You don't have permission to use this command!"
  - stop
```

Time to add the real functionality to our command. We want to give ourselves a wand, so we'll just use the give command, which was also shown in the first tutorial. The first argument is clear, since we just want to give it to the player using the command. The item argument, on the other hand, should be our own custom wand item.

We'll create an item script for that called *Cuboid_Tool_Item*. Let's follow the documentation and start by setting its *type* to *item* and *debug* to *full*. Now we will set its *static* key to *true* because we only need to get copies of the item, which will be generated on server start.

We're now ready to specify *display name* and *lore* keys. These accept formatted text, so let's build some!.The way of doing this is through the *<texts* base tag. We prefer to build it from input, so our display name will look like *"<texts.for_input[text:Cuboid Tool|color:blue]>"*. Keep in mind the quotes are needed, as there's a space inside the tag and the argument would be split otherwise.

Let's do the same for the *lore* key, except there can be multiple lines of text so it'll take the shape of a list. Our two lore lines, formatted correctly, will be *"<texts.for_input[text:Left click to set the first location|color:aqua]>"* and *"<texts.for_input[text:Right click to set the second location|color:aqua]>"*.

We also want to store some hidden information inside the item. This will let us know it's a wand later on. This can be done by adding a *flags* key, with key and value pairs inside it. In our case, a *cuboid_tool: true* should be enough.

Our item script will now look like the following:

```
Cuboid_Tool_Item:
  type: item
  debug: full
  static: true
  display name: "<texts.for_input[text:Cuboid Tool|color:blue]>"
  lore:
    - "<texts.for_input[text:Left click to set the first location|color:aqua]>"
    - "<texts.for_input[text:Right click to set the second location|color:aqua]>"
  flags:
    cuboid_tool: true
```

Adding the give command with our new custom item, our whole command script should look like this:

```
Cuboid_Tool_Command:
  type: command
  debug: full
  name:
  - cuboidtool
  - ct
  description: Gives yourself a cuboid wand.
  script:
  - if <context.source> != player:
    - echo "This is a player command only!"
    - stop
  - if <player.gamemode> != creative:
    - tell <player> "You don't have permission to use this command!"
    - stop
  - give <player> Cuboid_Tool_Item
```

Got it? Neat! Let's test it now. Save the script first and reload it ingame. We now want to test all the options, so start by running the command from console, which should be just *ct* (no backslash needed). If everything goes according to plan, you should now see an error message there.

Let's test ingame now, but on survival gamemode. You should now see the other error instead, and receive no wand. Time for the final test! Swap to creative and run the command again. You should now have the wand in your inventory.

Selecting The Cuboid

So we have the wand, but it's not magical at all. Our next step is being able to select cuboids by clicking their corners with the wand. Since we need to execute some code when we click, we'll need to use events inside a world script. We're already familiar with those, so we can build one fast and search for the appropiate events. We intend to use left click for selecting the first corner, and right click for the second one, so let's grab those two events: *on player left clicks block* and *on player right clicks block*.

Our brand new world script should look like this:

```
Cuboid_Tool_Events:
  type: world
  debug: full
  events:
    on player left clicks block:
    on player right clicks block:
```

But we only want the selection to work if the item in hand is our cuboid wand. Luckily, these two events have *with_item:* switches, that will check for an item. Item switches are a bit special and offer more options than other switches. We'll need to specify a type of item, and the flag it contains. This is simpler than it sounds, since we just need to write *with_item:type:blaze_rod|flagged:Cuboid_Tool*.

We also want the selection to only be possible for creative players, but there's no direct switch for checking gamemodes. That's not a problem though, because we're just going to learn how to create our own **custom event switches**. This is possible thanks to the general *require:* switch, which will check if the condition specified is true before triggering the event.

The condition we need is similar to the one we used for the if command, except shaped as a tag. We start with the *<player* base, add the *.gamemode* modifier and close it with an *.equals[creative]>*.

The two events with our brand new switches should now look like this:

```
    on player left clicks block
with_item:type:blaze_rod|flagged:Cuboid_Tool
require:<player.gamemode.equals[creative]>:
    on player right clicks block
with_item:type:blaze_rod|flagged:Cuboid_Tool
require:<player.gamemode.equals[creative]>:
```

Once our switches are ready, we can go ahead and add the actual functionality. We want to store the first and second block's location selected by the player, and we'll use flags for that. **Flags** let us store information (a key and a value) inside an entity or globally inside the server.

The command for creating or editing flags is just *- flag*. We want to flag the player, just in case there are more than a single moderator creating cuboids at the same time. Finally, we'll call the key *Cuboid_Tool.Primary_Block* for the first event, and *Cuboid_Tool.Secondary_Block* for the second one. The value should just be the location clicked, which can be obtained from the event's context.

These two flag commands would look like this:

```
    - flag <player> Cuboid_Tool.Primary_Block:<context.location>
    - flag <player> Cuboid_Tool.Secondary_Block:<context.location>
```

We should now be able to select blocks easily, but let's make sure! With our magic wand all we have to do is left and right click different blocks. Then we can just use the */ex* command to tell ourselves the value of the flags we've used. It would look something like this: */ex tell <player> <player.flag[Cuboid_Tool.Primary_Block]>*, and *Secondary_Block* for the second. Can you see the location you clicked in chat? Perfect!

We can also just automatically give that information when a block is selected. Let's add a *- tell* to both events that narrates the location clicked. We also want to cancel the left click breaking the target block, so we'll slip a *- determine cancelled* into the left clicks event.

Putting it together, our world script will look like this:

```
Cuboid_Tool_Events:
  type: world
  debug: full
  events:
    on player left clicks block
with_item:type:blaze_rod|flagged:Cuboid_Tool
require:<player.gamemode.equals[creative]>:
    - determine cancelled
    - flag <player> Cuboid_Tool.Primary_Block:<context.location>
    - tell <player> "Primary block selected: <context.location>"
    on player right clicks block
with_item:type:blaze_rod|flagged:Cuboid_Tool
require:<player.gamemode.equals[creative]>:
    - flag <player> Cuboid_Tool.Secondary_Block:<context.location>
    - tell <player> "Secondary block selected: <context.location>"
```

Let's select two blocks again. You should now automatically see the locations in chat, which is very handy.

Creating The Cuboid

After making sure our cuboid selection code works as intended, it's time to continue scripting. We want to be able to create a brand new cuboid (and remove it if needed) from these two locations and store it globally on a server flag. We'll need to change the structure of our command script a bit for this, but that's not a problem.

We'll start with the command description. Since it will do more than just give out a wand, we'll explain its usage. Replacing it with a *Main Cuboid Tool command. Proper usage is /ct wand/create/remove (name).* should work.

Now we need to change the script section. We'll start by adding a check before the give command that throws an error and stops the queue if the command has no arguments. That's possible checking the size of the argument list, which can be obtained from a context tag. Our condition would be *<context.arguments* with a *.size>* modifier and then compared to *1*.

This new check will look like this:

```
  - if <context.arguments.size> < 1:
    - tell <player> "Proper command usage is /ct wand/create/remove
(name)"
    - stop
```

Now we need to execute different blocks of script depending on the first argument specified, and `- choose` is the perfect command for that. We'll use this **first argument**, which can be accessed with the `.get[1]>` modifier, as the choose value. Then we can just add cases with the expected values, as well as a default case that throws an error. Remember to place the give command inside the proper wand case.

Putting the choose together, it should look like this:

```
  - choose <context.arguments.get[1]>:
    - case wand:
      - give <player>
"<item[blaze_rod].with[display_name:<texts.for_input[text:Cuboid
Tool|color:blue]>].with_flags[Cuboid_Tool:true]>"
    - case create:
    - case remove:
    - default:
      - tell <player> "Unknown argument specified! Proper command usage
is /ct wand/create/remove (name)"
```

Now it's time to develop the *create* and *remove* cases. We'll start by making sure that the command contains a *name* argument, as the proper usage explains. We can do that by checking if the argument list size is less than 2 and throwing an error (and stopping the queue) if that's the case.

This new checks would look like the following:

```
    - case create:
      - if <context.arguments.size> < 2:
        - tell <player> "No name specified! Proper command usage is /ct
create (name)"
        - stop
```

```
    - case remove:
      - if <context.arguments.size> < 2:
        - tell <player> "No name specified! Proper command usage is /ct
remove (name)"
        - stop
```

And after all these checks, we just lack the actual functionality. For the *create* subcommand, we'll flag the server with the cuboid name and the selected cuboid. So we start with *- flag server*, then add *Cuboids.<context.arguments.get[2]>*, which is the name argument, and then create the cuboid after a *:*.

Denizen2Sponge has a cuboid wrapping tool which can be accessed through the *<server* base tag. For creating a cuboid from two locations, all we have to do is *<server.cuboid_wrapping[Location1|Location2]>*. These two locations can be obtained from the flags we set earlier on the player thanks to the *<player.flag[FlagName]>* tag.

Putting it together, our flag command is ready:

```
      - flag server
Cuboids.<context.arguments.get[2]>:<server.cuboid_wrapping[<player.flag[
Cuboid_Tool.Primary_Block]>|<player.flag[Cuboid_Tool.Secondary_Block]>]>
```

For our remove subcommand, removing the cuboid is much easier. We just have to use the *- unflag* command with the proper flag key.

This command looks just like this:

```
      - unflag server Cuboids.<context.arguments.get[2]>
```

We can also add some *- tell* commands to all three subcommands to give information about what happened.

Our command script should now be ready for some intensive testing and looks like this:

```
Cuboid_Tool_Command:
  type: command
  debug: full
  name:
  - cuboidtool
```

```
  - ct
  description: Gives yourself a cuboid wand.
  script:
  - if <context.source> != player:
    - echo "This is a player command only!"
    - stop
  - if <player.gamemode> != creative:
    - tell <player> "You don't have permission to use this command!"
    - stop
  - if <context.arguments.size> < 1:
    - tell <player> "Proper command usage is /ct wand/create/remove
(name)"
    - stop
  - choose <context.arguments.get[1]>:
    - case wand:
      - give <player> Cuboid_Tool_Item
    - case create:
      - if <context.arguments.size> < 2:
        - tell <player> "No name specified! Proper command usage is /ct
create (name)"
        - stop
      - flag server
Cuboids.<context.arguments.get[2]>:<server.cuboid_wrapping[<player.flag[
Cuboid_Tool.Primary_Block]>|<player.flag[Cuboid_Tool.Secondary_Block]>]>
    - case remove:
      - if <context.arguments.size> < 2:
        - tell <player> "No name specified! Proper command usage is /ct
remove (name)"
        - stop
      - unflag server Cuboids.<context.arguments.get[2]>
    - default:
      - tell <player> "Unknown argument specified! Proper command usage
is /ct wand/create/remove (name)"
```

While this might look like a monster script, it's pretty well organized and we've written it step by step, so it's not a big deal.

Testing

So we can already create a cuboid, but what now? We can't see it at all, so we better make sure it exists. Let's give ourselves a wand with `/ct wand`, and then left and right click two different blocks. Now we can do `/ct create mycuboid` to create it, and should receive a message in chat confirming the creation.

Now it's time to prove how useful cuboids can be. We can mimic WorldEdit and fill the whole cuboid with a block type. Let's do that directly ingame through the `/ex` command. We just need to use the command `- setblock` a list of locations, which can be obtained through the tag modifier for cuboids `.block_locations>`.

In our cause, we can retrieve the cuboid from the server flag, making sure we convert it to a cuboid type tag (flags usually lose their type when being retrieved). This can be done by nesting the flag value inside a `<cuboid[]` base tag. The resulting cuboid tag would in the end be `<cuboid[<server.flag[Cuboids.mycuboid]>]`

Our block of choice is just `stone`, though any block type can be used. The whole command would be: `/ex setblock <cuboid[<server.flag[Cuboids.mycuboid]>].block_locations> stone`.

Did that work? Assuming you are not stuck inside a stone cuboid, you can now see your creation. Great!

Additional Checks

Our script already works as intended and does the job, but some problems might arise when a user without experience tries to use it. For example, we should probably check if a cuboid name `mycuboid` already exists before creating it, as well as check if a cuboid exists before removing it.

This shouldn't be too complex. Let's scroll to our create case and place a new `- if` before the flag command. We want to check if the cuboid already exists, so we can use the `<server.has_flag[FlagName]>` tag.

Our new if will take the following shape:

```
  - if <server.has_flag[Cuboids.<context.arguments.get[2]>]>:
    - tell <player> "A cuboid with name <context.arguments.get[2]>
already exists!"
    - stop
```

Now for the remove subcommand, we have to invert the condition with a *.not*> tag modifier.

This if block would look like this:

```
  - if <server.has_flag[Cuboids.<context.arguments.get[2]>].not>:
    - tell <player> "The cuboid with name <context.arguments.get[2]>
doesn't exist!"
    - stop
```

This should be it for now. Now we have a handy Cuboid Tool, which will be used a lot in other tutorials. Thanks for putting the time in following this tutorial and **happy scripting**!