

Events

Events that get fired during game to control recipes, world, etc.

- [List of all events](#)
- [Custom Items](#)
- [EventJS](#)
- [Custom Blocks](#)
- [CommandEventJS](#)
- [TagEventJS](#)
- [Loot Table Modification](#)
- [RecipeEventJS](#)
- [Item Modification](#)
- [WorldgenAddEventJS \(1.16\)](#)
- [Block Modification](#)
- [JEI Integration](#)
- [WorldgenRemoveEventJS \(1.16\)](#)
- [REI Integration](#)
- [ItemTooltipEventJS](#)
- [Worldgen Events](#)
- [Chat Event](#)
- [Custom Fluids](#)
- [Command Registry](#)
- [Datapack Load Events](#)

List of all events

This is a list of all events. It's possible that not all events are listed here, but this list will be updated regularly.

Click on event ID to open it's class and see information, fields and methods.

Type descriptions:

- Startup: Scripts go in kubejs/startup_scripts folder.
- Server: Scripts go in kubejs/server_scripts folder. Will be reloaded when you run /reload command.
- Server Startup: Same as Server, and the event will be fired at least once when server loads.
- Client: Scripts go in kubejs/client_scripts folder. Currently only reloaded if you have KubeJS UI installed in you run Ctrl+F5 in a menu.
- Client Startup: Same as Client, and the event will be fired at least once when client loads.

ID	Cancellable	Type	Note
init	No	Startup	
postinit	No	Startup	
loaded	No	Startup	
command.registry	No	Server	
command.run	Yes	Server	
client.init	No	Client	
client.debug_info.left	No	Client	
client.debug_info.right	No	Client	
client.generate_assets	No	Client	
client.logged_in	No	Client	

ID	Cancellable	Type	Note
client.logged_out	No	Client	
client.tick	No	Client	
server.load	No	Server	
server.unload	No	Server	
server.tick	No	Server	
server.datapack.first	No	Server	
server.datapack.last	No	Server	
recipes	No	Server	
recipes.after_load	No	Server	Does not work 1.18+
level.load	No	Server	Replace <code>level</code> with <code>world</code> in 1.16
level.unload	No	Server	Replace <code>level</code> with <code>world</code> in 1.16
level.tick	No	Server	Replace <code>level</code> with <code>world</code> in 1.16
level.explosion.pre	Yes	Server	Replace <code>level</code> with <code>world</code> in 1.16
level.explosion.post	No	Server	Replace <code>level</code> with <code>world</code> in 1.16
player.logged_in	No	Server	
player.logged_out	No	Server	
player.tick	No	Server	
player.data_from_server.	Yes	Client	
player.data_from_client.	Yes	Server	
player.chat	Yes	Server	
player.advancement	No	Server	

ID	Cancellable	Type	Note
player.inventory.opened	No	Server	
player.inventory.closed	No	Server	
player.inventory.changed	No	Server	
player.chest.opened	No	Server	
player.chest.closed	No	Server	
entity.death	Yes	Server	
entity.attack	Yes	Server	
entity.drops	Yes	Server	
entity.check_spawn	Yes	Server	
entity.spawned	Yes	Server	
block.registry	No	Startup	
block.missing_mappings	No	Server	
block.tags	No	Server	
block.right_click	Yes	Server	
block.left_click	Yes	Server	
block.place	Yes	Server	
block.break	Yes	Server	
block.drops	No	Server	
item.registry	No	Startup	
item.missing_mappings	No	Server	
item.tags	No	Server	

ID	Cancellable	Type	Note
item.right_click	Yes	Server	
item.right_click_empty	No	Server	
item.left_click	No	Server	
item.entity_interact	Yes	Server	
item.modification	No	Startup	
item.pickup	Yes	Server	
item.tooltip	No	Client	
item.toss	Yes	Server	
item.crafted	No	Server	
item.smelted	No	Server	
fluid.registry	No	Startup	
fluid.tags	No	Server	
entity_type.tags	No	Server	
worldgen.add	No	Startup	
worldgen.remove	No	Startup	

Custom Items

This is a startup_scripts/ event

```
// Listen to item registry event
onEvent('item.registry', event => {

    // The texture for this item has to be placed in kubejs/assets/kubejs/textures/item/test_item.png
    // If you want a custom item model, you can create one in Blockbench and put it in
    kubejs/assets/kubejs/models/item/test_item.json
    event.create('test_item')

    // You can chain builder methods as much as you like
    event.create('test_item_2').maxStackSize(16).glow(true)

    // You can specify item type as 2nd argument in create(), some types have different available methods
    event.create('custom_sword', 'sword').tier('diamond').attackDamageBaseline(10.0)
})
```

Valid item types:

- "basic"
 - default
- "sword"
- "pickaxe"
- "axe"
- "shovel"
- "hoe"
- "helmet"
- "chestplate"
- "leggings"
- "boots"

Other methods item builder supports:

You can chain these methods after create()

Anything with a ??? may not be completely accurate

Physical Properties

- `maxStackSize(size)`
- `unstackable()`
 - Identical to `maxStackSize(1)`
- `maxDamage(damage)`
 - ie max durability of the item
- `burnTime(ticks)`
 - In a furnace
- `fireResistant(true/false)`

Non-Model Visual Stuff

- `rarity('rarity')`
 - Options are:
 - "common"
 - "uncommon"
 - "rare"
 - "epic"
- `glow(true/false)`
- `tooltip(text...)`
 - The text under the item name to provide details about it
- `color(index, colorHex)`
 - If you do not have a custom model, index is 0
 - If you do have a custom model, then index is the layer that you want to affect
 - [there is an example below](#)
- `color((item, number) => {...})`
 - any code you want
 - must return a color
 - [there is an example below](#)
 - ???
- `displayName(name)`
- `name(item => {...})`
 - you can put whatever code in there you want
 - must return a string
 - ???
- `translationKey(key)`
 - ???
 - You don't need this unless you know what you are doing

Model Editing

[There is an example below](#)

- `textureJson(json)`

- for example {layer0:"minecraft:item/sand",layer1:"minecraft:item/paper"}
- The contents of the texture part of item model
- ???
- modelJson(json)
 - the entire json that you would put for a item model, you can just put in here
 - ???
- parentModel(modelName)
 - Set the "parent" property of this items model to modelName
- texture(customTexturePath)
 - for example "minecraft:item/feather"
- texture(key, customTexturePath)
 - if key is "layer0", then its the same as texture(customTexturePath)
 - ???

Bar

[There an example farther below](#)

- barColor((item) => {...})
 - must return a color
 - any code you want
 - ???
- barWidth(width)
 - ???

Custom Uses

[The is a section below for an example](#)

- useAnimation(animation)
 - Can be:
 - "spear"
 - trident
 - "crossbow"
 - "eat"
 - food
 - "spyglass"
 - "block"
 - "none"
 - "bow"
 - "drink"
 - ???
- useDuration(itemstack => {...})
 - any code you want
 - for example useDuration(itemstack => 60)
 - three seconds

- must return a whole number
- if you want something that does not end on its own, then use something like 72000 (an hour)
- ???
- use((level, player, hand) => {...})
 - for example use(() => true)
 - any code you want
 - item is usable if it is true
 - must return a boolean
 - ???
- finishUsing((itemstack, level, entity) => {...})
 - any code you want
 - when the duration completes
 - ???
- releaseUsing((itemstack, level, entity, tick) => {...})
 - any code you want
 - when released before the duration completes
 - ???

Miscellaneous

- type(type)
 - for 1.16
- tag(resourceLocation)
 - ???
- tool(type, level)
 - for 1.16
- modifyAttribute(attribute, identifier, d, operation)
 - ???
- group(group_id)
 - Creative mode tab
 - Vanilla tabs are:
 - "search"
 - "buildingBlocks"
 - "decorations"
 - "redstone"
 - "transportation"
 - "misc"
 - "food"
 - "tools"
 - "combat"
 - "brewing"
- containerItem(id)
 - A item to reference properties of
 - ???
- subtypes(item => {...})

- must return a itemstack collection
- This is for making JEI or creative menu have the same item multiple times with different NBT
- any code you want
- ???
- food(foodBuilder => {...})
 - [There is an example farther down](#)

Tool

Methods available if you use 'sword', 'pickaxe', 'axe', 'shovel' or 'hoe' type:

- tier(toolTier)
 - Can be:
 - "wood"
 - "stone"
 - "iron"
 - "gold"
 - "diamond"
 - "netherite"
- modifyTier(tier => ...)
- Same syntax as custom tool tier, see below
- attackDamageBaseline(damage)
 - You only want to modify this if you are creating a custom weapon such as Spear, Battleaxe, etc.
- attackDamageBonus(damage)
- speedBaseline(speed)
 - Same as attackDamageBaseline, only modify for custom weapon types
- speed(speed)

Armor

Methods available if you use 'helmet', 'chestplate', 'leggings' or 'boots' type:

- tier('armorTier')
 - Can be:
 - "leather"
 - "chainmail"
 - "iron"
 - "gold"
 - "diamond"
 - "turtle"
 - "netherite"
- modifyTier(tier => ...) // Same syntax as custom armor tier, see below

Creating custom tool and armor tiers

All values are optional and by default are based on iron tier

```
onEvent('item.registry.tool_tiers', event => {
  event.add('tier_id', tier => {
    tier.uses = 250
    tier.speed = 6.0
    tier.attackDamageBonus = 2.0
    tier.level = 2
    tier.enchantmentValue = 14
    tier.repairIngredient = '#forge:ingots/iron'
  })
})
```

```
onEvent('item.registry.armor_tiers', event => {
  // Slot indices are [FEET, LEGS, BODY, HEAD]
  event.add('tier_id', tier => {
    tier.durabilityMultiplier = 15 // Each slot will be multiplied with [13, 15, 16, 11]
    tier.slotProtections = [2, 5, 6, 2]
    tier.enchantmentValue = 9
    tier.equipSound = 'minecraft:item.armor.equip_iron'
    tier.repairIngredient = '#forge:ingots/iron'
    tier.toughness = 0.0 // diamond has 2.0, netherite 3.0
    tier.knockbackResistance = 0.0
  })
})
```

Examples:

Custom Foods

These methods are each optional, and you may include as many or as few as you like.

```
onEvent('item.registry', event => {
  event.create('magic_steak').food(food => {
    food
    food.hunger(6)
    food.saturation(6)//This value does not directly translate to saturation points gained
    //The real value can be assumed to be:
    //min(hunger * saturation * 2 + saturation, foodAmountAfterEating)
    food.effect('speed', 600, 0, 1)
```

```

    [].removeEffect('poison')
    [].alwaysEdible()//Like golden apples
    [].fastToEat()//Like dried kelp
    [].meat()//Dogs are willing to eat it
    []eaten(ctx => { //runs code upon consumption
        []ctx.player.tell('Yummy Yummy!')
        []//If you want to modify this code then you need to restart the game.
        []//However, if you make this code call a global startup function
        []//and place the function *outside* of an 'onEvent'
        []//then you may use the command:
        []// /kubejs reload startup_scripts
        []//to reload the function instantly.
    })
  })
})
})

```

Custom Uses

```

onEvent("item.registry", event => {
  event.create("nuke_soda", "basic")
    .tooltip("§5Taste of Explosion!")
    .tooltip("§c...Inappropriate intake may cause disastrous result.")
  /**
   * The use animation of the item, can be "spear" (trident),
   * "crossbow", "eat" (food), "spyglass", "block", "none", "bow", "drink"
   * When using certain animations, corresponding sound will be played.
   */
  .useAnimation("drink")
  /**
   * The duration before the item finishes its using,
   * if you need something like hold-and-charge time (like bow),
   * consider set this to 72000 (1h) or more.
   * A returned value of 0 or lower will render the item not usable.
   */
  .useDuration((itemstack) => 64)
  /**
   * When item is about to be used.
   * If true, item will starts it use animation if duration > 0.
   */
  .use((level, player, hand) => true)

```

```

/**
 * When the item use duration expires.
 */
.finishUsing((itemstack, level, entity) => {
    let effects = entity.potionEffects;
    effects.add("haste", 120 * 20)
    itemstack.itemStack.shrink(1)
    if (entity.player) {
        entity.minecraftPlayer.addItem(Item.of("minecraft:glass_bottle").itemStack)
    }
    return itemstack;
})
/**
 * When the duration is not expired yet, but
 * players release their right button.
 * Tick is how many ticks remained for player to finish using the item.
 */
.releaseUsing((itemstack, level, entity, tick) => {
    itemstack.itemStack.shrink(1)
    level.createExplosion(entity.x, entity.y, entity.z).explode()
})
})

```

Bar

```

event.create("hammer")
    //Determine how long the bar is, should be an integer between 0 (empty) and 13 (full)
    //If the value is below 0, it will be treated as 0.
    //The value is capped at 13, any value over 13 will be considered "full", thus making it not shown
    .barWidth(i => i.nbt.contains("hit_count") ? i.nbt.getInt("hit_count") / 13.0 : 0)
    //Determine what color should the bar be.
    .barColor(i => Color.AQUA)

```

Dynamic Tinting and Model Stuff

```

onEvent("item.registry", (event) => {
    /**
     * Old style with just setting color by index still works!
     */
    event

```

```

    .create("old_color_by_index")
    .textureJson({
      layer0: "minecraft:item/paper",
      layer1: "minecraft:item/ghast_tear",
    })
    .color(0, "#70F00F")
    .color(1, "#00FFF0");

```

event

```

.create("cooler_sword", "sword")
.displayName("Test Cooler Sword")
.texture("minecraft:item/iron_sword")
.color((itemstack) => {
  /**
   * Example by storing the color in the nbt of the itemstack
   * You have to return -1 to apply no tint.
   *
   * U can test this through: /give @p kubejs:cooler_sword{color:"#ff0000"}
   */
  if (itemstack.nbt && itemstack.nbt.color) {
    return itemstack.nbt.color;
  }

  return -1;
});

```

event

```

.create("test_item")
.displayName("Test Item")
.textureJson({
  layer0: "minecraft:item/beef",
  layer1: "minecraft:item/ghast_tear",
})
.color((itemstack, tintIndex) => {
  /**
   * If you want to apply the color to a specific layer, you can use the tintIndex
   * tintIndex is the texture layer index from the model: layer0 -> 0, layer1 -> 1, etc.
   * U can use the `Color` wrapper for some default colors
   *
   * This example will apply the color to the ghast_tear texture.

```

```

    return */
    if (tintIndex == 1) {
        return Color.BLUE;
    }
    return -1;
    });

/**
 * Set a texture for a specific layer
 */
event.create("test_sword", "sword").displayName("Test Sword").texture("layer0", "minecraft:item/bell");

/**
 * Directly set your custom model json
 */
event.create("test_something").displayName("Test something").modelJson({
    parent: "minecraft:block/anvil",
});
});

```

EventJS

This event is the most basic event class, parent of all other events.

Parent class

[Object](#)

Can be cancelled

No

Variables and Functions

Name	Return Type	Info
cancel()	void	Cancels event. If the event can't be cancelled, it won't do anything.

Custom Blocks

This is a startup script.

```
onEvent('block.registry', event => {
  event.create('test_block')
    .material('glass')
    .hardness(0.5)
    .displayName('Test Block') // No longer required in 1.18.2+
    .tagBlock('minecraft:mineable/shovel') // Make it mine faster using a shovel in 1.18.2+
    .tagBlock('minecraft:needs_iron_tool') // Make it require an iron or higher level tool on 1.18.2+
    .requiresTool(true) // Make it require a tool to drop any loot

  // Block with custom type (see below for list of types for 1.18 (use .type for 1.16))
  event.create('test_block_slab', 'slab').material('glass').hardness(0.5)

  //uses a combo of properties (things you might consider blockstate) and random tick to make the block
  eventually change to test_block, but only progresses if waterlogged

  event.create('test_block_2').material('glass').hardness(0.2).property(BlockProperties.WATERLOGGED).property(BlockProperties.AGE_7).randomTick(tick => {
    const block = tick.block
    const properties = block.properties
    const age = Number(properties.age)
    if (properties.waterlogged == 'false') return
    if (age == 7) {
      block.set('kubejs:test_block')
    } else {
      block.set('kubejs:test_block_2', {waterlogged: 'true', age: `${age+1}`})
    }
  })
})
```

The texture for this block has to be placed in `kubejs/assets/kubejs/textures/block/test_block.png`.
If you want a custom block model, you can create one in Blockbench and put it in

kubejs/assets/kubejs/models/block/test_block.json .

List of available materials - to change break/walk sounds and to *change some properties*.

Materials (1.18.2)
air
wood
stone
metal
grass
dirt
water
lava
leaves
plant
sponge
wool
sand
glass
explosive
ice
snow

Materials (1.18.2)

clay

vegetable

dragon_egg

portal

cake

web

slime

honey

berry_bush

lantern

Other methods block builder supports:

- `displayName('name')`
 - Not required for 1.18.2+
- `material('material')`
 - See list above
- `type('basic')`
 - See available types below.
 - Do not use for 1.18.2, use the syntax in the second example above
- `hardness(float)`
 - ≥ 0.0
- `resistance(float)`
 - ≥ 0.0
- `unbreakable()`
 - Sets the resistance to MAX_VALUE and hardness to -1, like bedrock
- `lightLevel(int)`
 - 0.0 - 1.0

- harvestTool('tool', level)
 - Available tools: pickaxe, axe, hoe, shovel
 - level >= 0
 - Not used in 1.18.2+, see tag in example above
- opaque(boolean)
- fullBlock(boolean)
- requiresTool(boolean)
- renderType('type')
 - Available types: solid, cutout, translucent
 - cutout required for blocks with texture like glass
 - translucent required for blocks like stained glass
- color(tintindex, color)
- textureAll('texturepath')
- texture('side', 'texturepath')
- model('modelpath')
- noItem()
- box(x0, y0, z0, x1, y1, z1, true)
 - 0.0 - 16.0
 - default is (0,0,0,16,16,16, true)
- box(x0, y0, z0, x1, y1, z1, false)
 - Same as above, but in 0.0 - 1.0 scale
 - default is (0,0,0,1,1,1, false)
- noCollision()
- notSolid()
- waterlogged()
- noDrops()
- slipperiness(float)
- speedFactor(float)
- jumpFactor(float)
- randomTick(randomTickEvent => {})
- see below
- item([itemBuilder](#) => {})
- setLootTableJson(json)
- setBlockstateJson(json)
- setModelJson(json)
- noValidSpawns(boolean)
- suffocating(boolean)
- viewBlocking(boolean)
- redstoneConductor(boolean)
- transparent(boolean)
- defaultCutout()
 - batches a bunch of methods to make blocks such as glass
- defaultTranslucent()
 - similar to defaultCutout() but using translucent layer instead
- tagBlock('forge:something')

- adds a block tag
- `tagItem('forge:something_better')`
 - adds an item tag
- `tagBoth('forge:something')`
 - adds both block and item tag
- `property(BlockProperty)`
 - See example above, but adds in more "blockstates" to the block
 - Example: `BlockProperties.WATERLOGGED`
 - You can add as many or few as you desire

RandomTickEvent callback properties

- `BlockContainerJS` block
- `Random` random
- `LevelJS` level
- `ServerJS` server

Block Properties

The default 1.18 properties are:

- `"MAX_RESPAWN_ANCHOR_CHARGES"`
- `"BAMBOO_LEAVES"`
- `"HANGING"`
- `"WEST_WALL"`
- `"BOTTOM"`
- `"EYE"`
- `"HALF"`
- `"DRAG"`
- `"MAX_ROTATIONS_16"`
- `"SOUTH"`
- `"MIN_RESPAWN_ANCHOR_CHARGES"`
- `"DISTANCE"`
- `"LOCKED"`
- `"EXTENDED"`
- `"SCULK_SENSOR_PHASE"`
- `"LEVEL"`
- `"DOOR_HINGE"`
- `"STAIRS_SHAPE"`
- `"EGGS"`
- `"LAYERS"`
- `"CONDITIONAL"`
- `"EAST_WALL"`
- `"HATCH"`
- `"ORIENTATION"`
- `"LEVEL_CAULDRON"`

- "RAIL_SHAPE_STRAIGHT"
- "SIGNAL_FIRE"
- "STRUCTUREBLOCK_MODE"
- "PISTON_TYPE"
- "MIN_LEVEL"
- "HAS_BOOK"
- "ATTACH_FACE"
- "WATERLOGGED"
- "FALLING"
- "AGE_25"
- "TRIGGERED"
- "MAX_LEVEL_8"
- "UNSTABLE"
- "CHEST_TYPE"
- "AGE_5"
- "SOUTH_WALL"
- "AGE_7"
- "STABILITY_MAX_DISTANCE"
- "BELL_ATTACHMENT"
- "AGE_1"
- "MAX_LEVEL_3"
- "ATTACHED"
- "AGE_3"
- "STAGE"
- "AGE_2"
- "POWER"
- "MAX_DISTANCE"
- "HAS_BOTTLE_1"
- "HAS_BOTTLE_0"
- "PICKLES"
- "HAS_BOTTLE_2"
- "OPEN"
- "DRIPSTONE_THICKNESS"
- "AGE_15"
- "LEVEL_HONEY"
- "CANDLES"
- "LEVEL_COMPOSTER"
- "LIT"
- "EAST_REDSTONE"
- "OCCUPIED"
- "MODE_COMPARATOR"
- "NORTH_REDSTONE"
- "IN_WALL"
- "SNOWY"
- "DOWN"
- "WEST"

- "NORTH_WALL"
- "MIN_LEVEL_CAULDRON"
- "BED_PART"
- "NORTH"
- "LEVEL_FLOWING"
- "TILT"
- "UP"
- "SOUTH_REDSTONE"
- "MAX_AGE_15"
- "HORIZONTAL_FACING"
- "BITES"
- "SLAB_TYPE"
- "MAX_AGE_2"
- "MAX_AGE_1"
- "ROTATION_16"
- "MAX_AGE_7"
- "STABILITY_DISTANCE"
- "MAX_AGE_5"
- "MAX_AGE_3"
- "MAX_AGE_25"
- "DELAY"
- "AXIS"
- "MAX_LEVEL_15"
- "HORIZONTAL_AXIS"
- "RAIL_SHAPE"
- "MOISTURE"
- "VERTICAL_DIRECTION"
- "DOUBLE_BLOCK_HALF"
- "NOTE"
- "BERRIES"
- "RESPAWN_ANCHOR_CHARGES"
- "EAST"
- "PERSISTENT"
- "HAS_RECORD"
- "FACING_HOPPER"
- "NOTEBLOCK_INSTRUMENT"
- "POWERED"
- "SHORT"
- "VINE_END"
- "WEST_REDSTONE"
- "ENABLED"
- "INVERTED"
- "FACING"
- "DISARMED"

You can make your own also using the following example:

```
const $BooleanProperty = Java.loadClass('net.minecraft.world.level.block.state.properties.BooleanProperty')
const $IntegerProperty = Java.loadClass('net.minecraft.world.level.block.state.properties.IntegerProperty')

onEvent('block.registry', event => {
  event.create('my_block').property($IntegerProperty.create("uses", 0,
2)).property($BooleanProperty.create("empty"))
})
```

Types

- basic
- detector
- slab
- stairs
- fence
- fence_gate
- wall
- wooden_pressure_plate
- stone_pressure_plate
- wooden_button
- stone_button
- falling
- crop

Detector Block Types

The detector block type can be used to run code when the block is powered with redstone signal.

Startup script code:

```
onEvent('block.registry', event => {
  event.create('test_block', 'detector').detectorId('myDetector')
})
```

Server script code:

```
onEvent('block.detector.myDetector.unpowered', event => { // you can also use powered and changed instead
of upowered
  event.block.set('tnt')
})
```


CommandEventJS

This event needs cleanup! Using it is not recommended.

Information

This event is fired when a command is executed on server side.

Parent class

[EventJS](#)

Can be cancelled

Yes

Variables and Functions

Name	Type	Info
parseResults	ParseResults <CommandSource>	Command params
exception	Exception	Error, set if something went wrong

TagEventJS

This event is fired when a tag collection is loaded, to modify it with script. You can add and remove tags for items, blocks, fluids and entity types.

This goes into server scripts.

Tags are per item/block/fluid/entity type and as such cannot be added based on things like NBT data!

Parent class

[EventJS](#)

Can be cancelled

No

Variables and Functions

Name	Type	Info
<u>type</u>	String	Tag collection type.
get(String tag)	TagWrapper	Returns specific tag container which you can use to add or remove objects to. tag parameter can be something like 'forge:ingots/copper'. If tag doesn't exist, it will create a new one.
add(String tag, String [/Regex ids])	TagWrapper	Shortcut method for event.get(tag).add(ids).
remove(String tag, String [/Regex ids])	TagWrapper	Shortcut method for event.get(tag).remove(ids).
removeAll(String tag)	TagWrapper	Shortcut method for event.get(tag).removeAll().
removeAllTagsFrom(String[] ids)	void	Removes all tags from object

TagWrapper class

Variables and Functions

Name	Type	Info
add(String [/Regex ids])	TagWrapper (itself)	Adds an object to this tag. If string starts with # then it will add all objects from the second tag. It can be either single string, regex (/regex/flags) or array of either.
remove(String [/Regex ids])	TagWrapper (itself)	Removes an object from tag, works the same as add().
removeAll()	TagWrapper (itself)	Removes all entries from tag.
getObjectIds()	Collection<ResourceLocation>	Returns a list of all entries in a tag. Will resolve any sub-tags.

Examples

```
// Listen to item tag event
onEvent('item.tags', event => {
  // Get the #forge:cobblestone tag collection and add Diamond Ore to it
  event.add('forge:cobblestone', 'minecraft:diamond_ore')

  // Get the #forge:cobblestone tag collection and remove Mossy Cobblestone from it
  event.remove('forge:cobblestone', 'minecraft:mossy_cobblestone')

  // Get #forge:ingots/copper tag and remove all entries from it
  event.removeAll('forge:ingots/copper')

  // Required for FTB Quests to check item NBT
  event.add('itemfilters:check_nbt', 'some_item:that_has_nbt_types')

  // You can create new tags the same way you add to existing, just give it a name
  event.add('forge:completely_new_tag', 'minecraft:clay_ball')

  // Removes all tags from this entry
  event.removeAllTagsFrom('minecraft:stick')

  // Add all items from the forge:stone tag to the c:stone tag, unless the id contains diorite
  const stones = event.get('forge:stone').getObjectIds()
  const blacklist = Ingredient.of(/.*diorite.*/)
  stones.forEach(stone => {
```

```
if (!blacklist.test(stone)) {  
    event.add('c:stone', stone)  
}  
})  
})
```

Recipes use item tags, not block or fluid tags, even if items representing those are blocks. Like `minecraft:cobblestone` even if it's a block, it will still be an item tag for recipes.

`tags.blocks` and `tags.fluids` are for adding tags to block and fluid types, they work the same way. You can find existing block and fluid tags if you look at a block with F3 mode enabled, on side. These are mostly only used for technical reasons, and like mentioned above, if its for recipes/inventory, you will want to use `tags.items` even for blocks.

Loot Table Modification

```
onEvent('block.loot_tables', event => {  
  event.addSimpleBlock('minecraft:dirt', 'minecraft:red_sand')  
})
```

```
onEvent('block.loot_tables', event => {  
  event.addSimpleBlock('minecraft:dirt') // To drop itself (fix broken blocks)  
  event.addSimpleBlock(/minecraft:.*_ore/, 'minecraft:red_sand') // To drop a different item  
})
```

```
onEvent('block.loot_tables', event => {  
  event.addBlock('minecraft:dirt', table => { // Build loot table manually  
    table.addPool(pool => {  
      pool.rolls = 1 // fixed  
      // pool.rolls = [4, 6] // or {min: 4, max: 6} // uniform  
      // pool.rolls = {n: 4, p: 0.3} // binominal  
      pool.survivesExplosion()  
      pool.addItem('minecraft:dirt')  
      pool.addItem('minecraft:dirt', 40) // 40 = weight  
      pool.addItem('minecraft:dirt', 40, [4, 8]) // [4-8] = count modifier, uses same syntax as rolls  
      // pool.addCondition({json condition, see vanilla wiki})  
      // pool.addEntry({json entry, see vanilla wiki for non-items})  
    })  
  })  
})
```

Example from Factorial: (adds 1-3 leaves dropped from all Leaves blocks, 4-8 logs from all log and wood blocks and 4-8 stone from Stone, Cobblestone, Andesite, Diorite and Granite)

```
onEvent('block.loot_tables', event => {  
  event.addBlock(/minecraft:.*_leaves/, table => {  
    table.addPool(pool => {  
      pool.survivesExplosion()  
      pool.addItem('factorial:leaf', 1, [1, 3])  
    })  
  })  
})
```

```

event.addBlock(/minecraft:.*(log|wood)/, table => {
  table.addPool(pool => {
    pool.survivesExplosion()
    pool.addItem('factorial:wood', 1, [4, 8])
  })
})

event.addBlock([
  'minecraft:stone',
  'minecraft:cobblestone',
  'minecraft:andesite',
  'minecraft:diorite',
  'minecraft:granite'
], table => {
  table.addPool(pool => {
    pool.rolls = [4, 8] // Roll the pool instead of individual items
    pool.survivesExplosion()
    pool.addItem('factorial:stone', 1)
  })
})
})

```

You can also modify existing loot tables to add items to them:

```

onEvent('block.loot_tables', event => {
  // all dirt blocks have a 50% chance to drop an enchanted diamond sword named "test"
  event.modifyBlock(/^minecraft:.*dirt/, table => {
    table.addPool(pool => {
      pool.addItem('minecraft:diamond_sword').randomChance(0.5).enchantWithLevels(1,
true).name(Text.of('Test').blue())
    })
  })
})

```

Other loot table types work too:

```

onEvent('entity.loot_tables', event => {
  // Add a loot table for the zombie that will drop 5 of either carrot (25% chance) or apple (75% chance)
  // Because the zombie already has a loot table, this will override the current one

```

```

event.addEntity('minecraft:zombie', table => {
  table.addPool(pool => {
    pool.rolls = 5
    pool.addItem('minecraft:carrot', 1)
    pool.addItem('minecraft:apple', 3)
  })
})

event.modifyEntity('minecraft:pig', table => {
  table.addPool(pool => {
    // Modify pig loot table to *also* drop dirt on top of its regular drops
    pool.addItem('minecraft:dirt')
  })
})
})

```

Supported table types:

Event ID	Override method name	Modify method name
generic.loot_tables	addGeneric	modify
block.loot_tables	addBlock	modifyBlock
entity.loot_tables	addEntity	modifyEntity
gift.loot_tables	addGift	modify
fishing.loot_tables	addFishing	modify
chest.loot_tables	addChest	modify

RecipeEventJS

Examples

The most basic script to add a single recipe:

```
onEvent('recipes', event => {
  event.shaped('3x minecraft:stone', [
    'SAS',
    'S S',
    'SAS'
  ], {
    S: 'minecraft:sponge',
    A: 'minecraft:apple'
  })
})
```

The most basic script to remove a recipe:

```
onEvent('recipes', event => {
  event.remove({output: 'minecraft:stick'})
})
```

Example recipe script:

```
// kubejs/server_scripts/example.js
// This is just an example script to show off multiple types of recipes and removal methods
// Supports /reload

// Listen to server recipe event
onEvent('recipes', event => {
  // Remove broken recipes from vanilla and other mods
  // This is on by default, so you don't need this line
  //event.removeBrokenRecipes = true

  event.remove({}) // Removes all recipes (nuke option, usually not recommended)
  event.remove({output: 'minecraft:stone_pickaxe'}) // Removes all recipes where output is stone pickaxe
  event.remove({output: '#minecraft:wool'}) // Removes all recipes where output is Wool tag
```



```

event.remove({input: '#forge:dusts/redstone'}) // Removes all recipes where input is Redstone Dust tag
event.remove({mod: 'quartzchests'}) // Remove all recipes from Quartz Chests mod
event.remove({type: 'minecraft:campfire_cooking'}) // Remove all campfire cooking recipes
event.remove({id: 'minecraft:glowstone'}) // Removes recipe by ID. in this case,
data/minecraft/recipes/glowstone.json
event.remove({output: 'minecraft:cooked_chicken', type: 'minecraft:campfire_cooking'}) // You can combine
filters, to create ANDk logic

// You can use 'mod:id' syntax for 1 sized items. For 2+ you need to use '2x mod:id' or Item.of('mod:id', count)
syntax. If you want NBT or chance, 2nd is required

// Add shaped recipe for 3 Stone from 8 Sponge in chest shape
// (Shortcut for event.recipes.minecraft.crafting_shaped)
// If you want to use Extended Crafting, replace event.shapeless with
event.recipes.extendedcrafting.shapeless_table
event.shaped('3x minecraft:stone', [
    'SAS',
    'S S',
    'SAS'
], {
    S: 'minecraft:sponge',
    A: 'minecraft:apple'
})

// Add shapeless recipe for 4 Cobblestone from 1 Stone and 1 Glowstone
// (Shortcut for event.recipes.minecraft.crafting_shapeless)
// If you want to use Extended Crafting, replace event.shapeless with
event.recipes.extendedcrafting.shaped_table
event.shapeless('4x minecraft:cobblestone', ['minecraft:stone', '#forge:dusts/glowstone'])

// Add Stonecutter recipe for Golden Apple to 4 Apples
event.stonecutting('4x minecraft:apple', 'minecraft:golden_apple')
// Add Stonecutter recipe for Golden Apple to 2 Carrots
event.stonecutting('2x minecraft:carrot', 'minecraft:golden_apple')

// Add Furnace recipe for Golden Apple to 3 Carrots
// (Shortcut for event.recipes.minecraft.smelting)
event.smelting('2x minecraft:carrot', 'minecraft:golden_apple')
// Similar recipe to above but this time it has a custom, static ID - normally IDs are auto-generated and will
change. Useful for Patchouli

```

```
event.smelting('minecraft:golden_apple', 'minecraft:carrot').id('mymodpack:my_recipe_id')
```

```
// Add similar recipes for Blast Furnace, Smoker and Campfire
```

```
event.blasting('3x minecraft:apple', 'minecraft:golden_apple')
```

```
event.smoking('5x minecraft:apple', 'minecraft:golden_apple')
```

```
event.campfireCooking('8x minecraft:apple', 'minecraft:golden_apple')
```

```
// You can also add .xp(1.0) at end of any smelting recipe to change given XP
```

```
// Add a smithing recipe that combines 2 items into one (in this case apple and gold ingot into golden apple)
```

```
event.smithing('minecraft:golden_apple', 'minecraft:apple', 'minecraft:gold_ingot')
```

```
// Create a function and use that to make things shorter. You can combine multiple actions
```

```
let multiSmelt = (output, input, includeBlasting) => {
```

```
  event.smelting(output, input)
```

```
  if (includeBlasting) {
```

```
    event.blasting(output, input)
```

```
  }
```

```
}
```

```
multiSmelt('minecraft:blue_dye', '#forge:gems/lapis', true)
```

```
multiSmelt('minecraft:black_dye', 'minecraft:ink_sac', true)
```

```
multiSmelt('minecraft:white_dye', 'minecraft:bone_meal', false)
```

```
// If you use custom({json}) it will be using vanilla Json/datapack syntax. Must include "type": "mod:recipe_id"!
```

```
// You can add recipe to any recipe handler that uses vanilla recipe system or isn't supported by KubeJS
```

```
// You can copy-paste the json directly, but you can also make more javascript-y by removing quotation marks  
from keys
```

```
// You can replace {item: 'x', count: 4} in result fields with Item.of('x', 4).toResultJson()
```

```
// You can replace {item: 'x'} / {tag: 'x'} with Ingredient.of('x').toJson() or Ingredient.of('#x').toJson()
```

```
// In this case, add Create's crushing recipe, Oak Sapling to Apple + 50% Carrot
```

```
// Important! Create has integration already, so you don't need to use this. This is just an example for datapack  
recipes!
```

```
// Note that not all mods format their jsons the same, often the key names ('ingredients', 'results', ect) are  
different.
```

```
// You should check inside the mod jar (mod.jar/data/modid/recipes/) for examples
```

```
event.custom({
```

```
  type: 'create:crushing',
```

```
  ingredients: [
```

```

    Ingredient.of('minecraft:oak_sapling').toJson()
  ],
  results: [
    Item.of('minecraft:apple').toResultJson(),
    Item.of('minecraft:carrot').withChance(0.5).toResultJson()
  ],
  processingTime: 100
})

```

// Example of using items with NBT in a recipe

```

event.shaped('minecraft:book', [
  'CCC',
  'WGL',
  'CCC'
], {
  C: '#forge:cobblestone',
  // Item.of('id', '{key: value}'), it's recommended to use /kubejs hand
  // If you want to add a count its Item.of('id', count, '{key: value}'). This won't work here though as crafting
  // table recipes to do accept stacked items
  L: Item.of('minecraft:enchanted_book', '{StoredEnchantments:[{lvl:1,id:"minecraft:sweeping"}]}'),
  // Same principle, but if its an enchantment, there's a helper method
  W: Item.of('minecraft:enchanted_book').enchant('minecraft:respiration', 2),
  G: '#forge:glass'
})

```

// In all shapeless crafting recipes, replace any planks with Gold Nugget in input items

```

event.replaceInput({type: 'minecraft:crafting_shapeless'}, '#minecraft:planks', 'minecraft:gold_nugget')

```

// In all recipes, replace Stick with Oak Sapling in output items

```

event.replaceOutput({}, 'minecraft:stick', 'minecraft:oak_sapling')

```

// By default KubeJS will mirror and shrink recipes, which makes things like UU-Matter crafting (from ic2) harder to do as you have less shapes.

// You can use noMirror() and noShrink() to stop this behaviour.

```

event.shaped('9x minecraft:emerald', [
  ' D ',
  'D  ',
  '  '
], {
  D: 'minecraft:diamond'
})

```

```
}).noMirror().noShrink()  
})
```

Possible settings you can change for recipes. It's recommended that you put this in it's own server scripts file, like `settings.js`

```
// priority: 5  
  
// Enable recipe logging, off by default  
settings.logAddedRecipes = true  
settings.logRemovedRecipes = true  
// Enable skipped recipe logging, off by default  
settings.logSkippedRecipes = true  
// Enable erroring recipe logging, on by default, recommended to be kept to true  
settings.logErroringRecipes = false
```

As mentioned before, you can add any recipe from any mod with JSON syntax (see `event.custom({})`) but these mods are supported as addons with special syntax:

- [KubeJS Mekanism](#)
- [KubeJS Immersive Engineering](#)
- [KubeJS Thermal](#)
- [KubeJS Blood Magic](#)
- [KubeJS Create](#)

Ingredient Actions

Poorly documented things below!

You can transform ingredients in shaped and shapeless recipes by adding these functions at end of it:

- `.damageIngredient(IngredientFilter filter, int damage?)` // Will damage item when you craft with it
- `.replaceIngredient(IngredientFilter filter, ItemStackJS item)` // Will replace item with another (like bucket)
- `.keepIngredient(IngredientFilter filter)` // Will keep item without doing anything to it
- `.customIngredientAction(IngredientFilter filter, String customId)` // Custom action that has to be registered in startup script

IngredientFilter can be either

- ItemStackJS ('minecraft:dirt', Item.of('minecraft:diamond_sword').ignoreNBT(), etc)
- Integer index of item in crafting table (0, 1, etc)
- Object with item and/or index ({item: 'something', index: 0}, etc)

Examples:

```
onEvent('recipes', event => {
  event.shapeless('9x minecraft:melon_slice', [ // Craft 9 watermelon slices
    Item.of('minecraft:diamond_sword').ignoreNBT(), // Diamond sword that ignores damage
    'minecraft:melon' // Watermelon block
  ]).damageIngredient(Item.of('minecraft:diamond_sword').ignoreNBT()) // Damage the sword (also has to ignore
  damage or only 0 damage will work)

  // Craft example block from 2 diamond swords and 2 dirt. After crafting first diamond sword is damaged (index
  0) and 2nd sword is kept without changes.
  event.shaped('kubejs:example_block', [
    'SD ',
    'D S'
  ], {
    S: Item.of('minecraft:diamond_sword').ignoreNBT(),
    D: 'minecraft:dirt'
  }).damageIngredient(0).keepIngredient('minecraft:diamond_sword')

  // Craft example block from 2 diamond swords and 2 stones. After crafting, diamond sword is replaced with
  stone sword
  event.shapeless('kubejs:example_block', [
    Item.of('minecraft:diamond_sword').ignoreNBT(),
    'minecraft:stone',
    Item.of('minecraft:diamond_sword').ignoreNBT(),
    'minecraft:stone'
  ]).replaceIngredient('minecraft:diamond_sword', 'minecraft:stone_sword')

  // Craft clay from sand, bone meal, dirt and water bottle. After crafting, glass bottle is left in place of water
  bottle
  event.shapeless('minecraft:clay', [
    'minecraft:sand',
    'minecraft:bone_meal',
    'minecraft:dirt',
    Item.of('minecraft:potion', {Potion: "minecraft:water"})
  ])
```

```
    ).replaceIngredient({item: Item.of('minecraft:potion', {Potion: "minecraft:water"})}, 'minecraft:glass_bottle')
```

// Register a customIngredientAction, and recipe that uses it

// This one takes the nbt from an enchanted book and applies it to a tool in the crafting table, for no cost.

// Thanks to Prunoideae for providing it!

```
Ingredient.registerCustomIngredientAction("apply_enchantment", (itemstack, index, inventory) => {
    let enchantment = inventory.get(inventory.find(Item.of("minecraft:enchanted_book").ignoreNBT())).nbt;
    if (itemstack.nbt == null)
        itemstack.nbt = {}
    itemstack.nbt = itemstack.nbt.merge({ Enchantments: enchantment.get("StoredEnchantments") })
    return itemstack;
})

event.shapeless("minecraft:book", ["#forge:tools", Item.of("minecraft:enchanted_book").ignoreNBT()])
    .customIngredientAction("#forge:tools", "apply_enchantment")
})
```

Item Modification

`item.modification` event is a startup script event that allows you to change properties of existing items

```
onEvent('item.modification', event => {  
  event.modify('minecraft:ender_pearl', item => {  
    item.maxStackSize = 64  
    item.fireResistant = true  
  })  
})
```

All available properties:

- int maxStackSize
- int maxDamage
- int burnTime
- String craftingReminder
- boolean fireResistant
- Rarity rarity
- tier = tierOptions => {
 - int uses
 - float speed
 - float attackDamageBonus
 - int level
 - int enchantmentValue
 - Ingredient repairIngredient
- }
- foodProperties = food => { // note: uses functions instead of a = b
 - hunger(int)
 - saturation(float)
 - meat(boolean)
 - alwaysEdible(boolean)
 - fastToEat(boolean)
 - effect(String effectId, int duration, int amplifier, float probability)
 - removeEffect(String effectId)
- }

WorldgenAddEventJS (1.16)

This event isn't complete yet and can only do basic things. Adding dimension-specific features also isn't possible yet, but is planned.

Example script: (kubejs/startup_scripts/worldgen.js)

```
onEvent('worldgen.add', event => {
  event.addLake(lake => { // Create new lake feature
    lake.block = 'minecraft:diamond_block' // Block ID (Use [] syntax for properties)
    lake.chance = 3 // Spawns every ~3 chunks
  })

  event.addOre(ore => {
    ore.block = 'minecraft:glowstone' // Block ID (Use [] syntax for properties)
    ore.spawnsIn.blacklist = false // Inverts spawn whitelist
    ore.spawnsIn.values = [ // List of valid block IDs or tags that the ore can spawn in
      '#minecraft:base_stone_overworld' // Default behavior - ores spawn in all stone types
    ]

    ore.biomes.blacklist = true // Inverts biome whitelist
    ore.biomes.values = [ // Biomes this ore can spawn in
      'minecraft:plains', // Biome ID
      '#nether' // OR #category, see list of categories below
    ]

    ore.clusterMinSize = 5 // Min blocks per cluster (currently ignored, will be implemented later, it's always 1)
    ore.clusterMaxSize = 9 // Max blocks per cluster
    ore.clusterCount = 30 // Clusters per chunk
    ore.minHeight = 0 // Min Y ore spawns in
    ore.maxHeight = 64 // Max Y ore spawns in
    ore.squared = true // Adds random value to X and Z between 0 and 16. Recommended to be true
    // ore.chance = 4 // Spawns the ore every ~4 chunks. You usually combine this with clusterCount = 1 for rare
    ores
  })

  event.addSpawn(spawn => { // Create new entity spawn
```



```
spawn.category = 'creature' // Category, can be one of 'creature', 'monster', 'ambient', 'water_creature' or  
'water_ambient'  
spawn.entity = 'minecraft:pig' // Entity ID  
spawn.weight = 10 // Weight  
spawn.minCount = 4 // Min entities per group  
spawn.maxCount = 4 // Max entities per group  
})  
})
```

All values are optional. All feature types have `biomes` field like `addOre` example

Valid biome categories ('#category'):

- taiga
- extreme_hills
- jungle
- mesa
- plains
- savanna
- icy
- the_end
- beach
- forest
- ocean
- desert
- river
- swamp
- mushroom
- nether

You can also use ('\$type' (case doesn't matter)) on Forge's BiomeDictionary:

- hot
- cold
- wet
- dry
- sparse
- dense
- spooky
- dead
- lush
- etc.... see [BiomeDictionary](#) for more

This is the order vanilla worldgen happens:

1. raw_generation
2. lakes
3. local_modifications
4. underground_structures
5. surface_structures
6. strongholds
7. underground_ores
8. underground_decoration
9. vegetal_decoration
10. top_layer_modification

It's possible you may not be able to generate some things in their layer, like ores in dirt, because dirt hasn't spawned yet. So you may have to change the layer by calling `ore.worldgenLayer = 'top_layer_modification'`. But this is not recommended.

If you want to remove things, see [this event](#).

Block Modification

`block.modification` event is a startup script event that allows you to change properties of existing blocks

```
onEvent('block.modification', event => {  
  event.modify('minecraft:stone', block => {  
    block.destroySpeed = 0.1  
    block.hasCollision = false  
  })  
})
```

All available properties:

- String material
- boolean hasCollision
- float destroySpeed
- float explosionResistance
- boolean randomlyTicking
- String soundType
- float friction
- float speedFactor
- float jumpFactor
- int lightEmission
- boolean requiredTool

JEI Integration

All JEI events are client sided and so go in the client_scripts folder

Sub-types

```
onEvent('jei.subtypes', event => {  
  event.useNBT('example:item')  
  event.useNBTKey('example:item', 'type')  
})
```

Hide Items & Fluids

```
onEvent('jei.hide.items', event => {  
  event.hide('example:ingredient')  
})  
  
onEvent('jei.hide.fluids', event => {  
  event.hide('example:fluid')  
})
```

Add Items & Fluids

```
onEvent('jei.add.items', event => {  
  event.add(Item.of('example:item', {test: 123}))  
})  
  
onEvent('jei.add.fluids', event => {  
  event.add('example:fluid')  
})
```

Add Information

```
onEvent('jei.information', event => {  
  event.add('example:ingredient', ['Line 1', 'Line 2'])  
})
```

Hide categories

```
onEvent('jei.remove.categories', event => {  
  console.log(event.getCategoryIds()) //log a list of all category ids to logs/kubejs/client.txt  
  
  event.remove('create:compacting')  
})
```

WorldgenRemoveEventJS

(1.16)

For more information on `biomes` field, see [worldgen.add](#) event page.

```
onEvent('worldgen.remove', event => {
  event.removeOres(ores => {
    ores.blocks = [ 'minecraft:coal_ore', 'minecraft:iron_ore' ] // Removes coal and iron ore
    ores.biomes.values = [ 'minecraft:plains' ] // Removes it only from plains biomes
  })

  event.removeSpawnsByID(spawns => {
    spawns.entities.values = [
      'minecraft:cow',
      'minecraft:chicken',
      'minecraft:pig',
      'minecraft:zombie'
    ]
  })

  event.removeSpawnsByCategory(spawns => {
    spawns.biomes.values = [
      'minecraft:plains'
    ]
    spawns.categories.values = [
      'monster'
    ]
  })
})
```

If something isn't removing, you may try to remove it "manually" by first printing all features (this will spam your console a lot, I suggest reading logs/kubejs/startup.txt) and then removing them by ID where possible.

```
onEvent('worldgen.remove', event => {  
  // May be one of the decoration types/levels described in worldgen.add docs  
  // But ores are *most likely* to be generated in this one  
  event.printFeatures('underground_ores')  
})
```

```
onEvent('worldgen.remove', event => {  
  event.removeFeatureById('underground_ores', 'mekanism:ore_copper')  
})
```

REI Integration

Note: REI integration only works on Fabric in 1.16. In 1.18+, it works on both Forge and Fabric!

All REI events are client sided and so go in the `client_scripts` folder

For 1.19+, see below (this is a temporary page!)

Hide Items

```
onEvent('rei.hide.items', event => {  
    event.hide('example:ingredient')  
})
```

Add Items

```
onEvent('rei.add.items', event => {  
    event.add(Item.of('example:item', { test: 123 }))  
})
```

Add Information

```
onEvent('rei.information', event => {  
    event.add('example:ingredient', 'Title', ['Line 1', 'Line 2'])  
})
```

Yeet categories

```
onEvent('rei.remove.categories', event => {  
    console.log(event.getCategoryIds()) //log a list of all category ids to logs/kubejs/client.txt  
  
    //event.remove works too, but yeeting is so much more fun ☹️  
    event.yeet('create:compacting')  
})
```

Grouping / Collapsible Entries (1.18.2+)


```

onEvent('rei.group', event => {
    // This event allows you to add custom entry groups to REI, which can be used to clean up the entry list
    // significantly.
    // As a simple example, we can add a 'Swords' group which will contain all (vanilla) swords
    // Note that each group will need an id (ResourceLocation) and a display name (Component / String)
    event.groupItems('kubejs:rei_groups/swords', 'Swords', [
        'minecraft:wooden_sword',
        'minecraft:stone_sword',
        'minecraft:iron_sword',
        'minecraft:diamond_sword',
        'minecraft:golden_sword',
        'minecraft:netherite_sword'
    ])

    // An easy use case for grouping stuff together could be using tags:
    // In this case, we want all the Hanging Signs and Sign Posts from Supplementaries to be grouped together
    event.groupItemsByTag('supplementaries:rei_groups/hanging_signs', 'Hanging Signs',
'supplementaries:hanging_signs')
    event.groupItemsByTag('supplementaries:rei_groups/sign_posts', 'Sign Posts', 'supplementaries:sign_posts')

    // Another example: We want all of these items to be grouped together ignoring NBT,
    // so you don't have a bajillion potions and enchanted books cluttering up REI anymore
    const useNbt = ['potion', 'enchanted_book', 'splash_potion', 'tipped_arrow', 'lingering_potion']

    useNbt.forEach(id => {
        const item = Item.of(id)
        const { namespace, path } = Utils.id(item.id)
        event.groupSameItem(`kubejs:rei_groups/${namespace}/${path}`, item.name, item)
    })

    // Items can also be grouped using anything that can be expressed as an IngredientJS,
    // including for example regular expressions or lists of ingredients
    event.groupItems('kubejs:rei_groups/spawn_eggs', 'Spawn Eggs', [
        '/spawn_egg/',
        '/^ars_nouveau:.*_se$/',
        'supplementaries:red_merchant_spawn_egg'
    ])

    // you can even use custom predicates for grouping, like so:
    event.groupItemsIf('kubejs:rei_groups/looting_stuff', 'Stuff with Looting I', item =>

```

```
// this would group together all items that have the Looting I enchantment on them
item.hasEnchantment('minecraft:looting', 1)
)

// you can also group fluids in much the same way as you can group items, for instance:
event.groupFluidsByTag('kubejs:rei_groups/fluid_tagged_as_water', '\Water\' (yeah right lmao)',
'minecraft:water')
})
```

This below code is meant for 1.19+

Hide Items

```
REIEvents.hide('item', event => {
  event.hide('example:ingredient')
})
```

Add Items

```
REIEvents.add('item', event => {
  event.add(Item.of('example:item', { test: 123 }))
})
```

Add Information

```
REIEvents.information(event => {
  event.addItem('example:ingredient', 'Title', ['Line 1', 'Line 2'])
})
```

Yeet categories

```
REIEvents.removeCategories(event => {
  console.log(event.getCategoryIds()) //log a list of all category ids to logs/kubejs/client.txt

  //event.remove works too, but yeeting is so much more fun ☹️
  event.yeet('create:compacting')
})
```

Grouping / Collapsible Entries (1.18.2+)

```

REIEvents.groupEntries(event => {
    // This event allows you to add custom entry groups to REI, which can be used to clean up the entry list
    // significantly.
    // As a simple example, we can add a 'Swords' group which will contain all (vanilla) swords
    // Note that each group will need an id (ResourceLocation) and a display name (Component / String)
    event.groupItems('kubejs:rei_groups/swords', 'Swords', [
        'minecraft:wooden_sword',
        'minecraft:stone_sword',
        'minecraft:iron_sword',
        'minecraft:diamond_sword',
        'minecraft:golden_sword',
        'minecraft:netherite_sword'
    ])

    // An easy use case for grouping stuff together could be using tags:
    // In this case, we want all the Hanging Signs and Sign Posts from Supplementaries to be grouped together
    event.groupItemsByTag('supplementaries:rei_groups/hanging_signs', 'Hanging Signs',
'supplementaries:hanging_signs')
    event.groupItemsByTag('supplementaries:rei_groups/sign_posts', 'Sign Posts', 'supplementaries:sign_posts')

    // Another example: We want all of these items to be grouped together ignoring NBT,
    // so you don't have a bajillion potions and enchanted books cluttering up REI anymore
    const useNbt = ['potion', 'enchanted_book', 'splash_potion', 'tipped_arrow', 'lingering_potion']

    useNbt.forEach(id => {
        const item = Item.of(id)
        const { namespace, path } = Utils.id(item.id)
        event.groupSameItem(`kubejs:rei_groups/${namespace}/${path}`, item.name, item)
    })

    // Items can also be grouped using anything that can be expressed as an IngredientJS,
    // including for example regular expressions or lists of ingredients
    event.groupItems('kubejs:rei_groups/spawn_eggs', 'Spawn Eggs', [
        '/spawn_egg/',
        '/^ars_nouveau:.*_se$/,'
        'supplementaries:red_merchant_spawn_egg'
    ])

    // you can even use custom predicates for grouping, like so:
    event.groupItemsIf('kubejs:rei_groups/looting_stuff', 'Stuff with Looting I', item =>

```

```
// this would group together all items that have the Looting I enchantment on them
```

```
item.hasEnchantment('minecraft:looting', 1)
```

```
)
```

```
// you can also group fluids in much the same way as you can group items, for instance:
```

```
event.groupFluidsByTag('kubejs:rei_groups/fluid_tagged_as_water', '\Water\' (yeah right lmao)',  
'minecraft:water')
```

```
})
```

ItemTooltipEventJS

A client event that allows adding tooltips to any item!

```
onEvent('item.tooltip', tooltip => {
  // Add tooltip to all of these items
  tooltip.add(['quark:backpack', 'quark:magnet', 'quark:crate'], 'Added by Quark Oddities')
  // You can also use any ingredient except #tag (due to tags loading much later than client scripts)
  tooltip.add(/refinedstorage:red_/, 'Can be any color')
  // Multiple lines with an array []. You can also escape ' by using other type of quotation marks
  tooltip.add('thermal:latex_bucket', ["Not equivalent to Industrial Foregoing's Latex", 'Line 2 text would go here'])

  // Use some data from the client to decorate this tooltip. name returns a component so we just append that.
  tooltip.add('minecraft:skeleton_skull', Text.of('This used to be ').append(Client.player.name).append("'s head"))

  tooltip.addAdvanced('thermal:latex_bucket', (item, advanced, text) => {
    text.add(0, Text.of('Hello')) // Adds text in first line, pushing the items name down a line. If you want the line below the item name, the index must be 1
  })

  tooltip.addAdvanced('minecraft:beacon', (item, advanced, text) => {
    // shift, alt and ctrl are all keys you can check!
    if (!tooltip.shift) {
      text.add(1, [Text.of('Hold ').gold(), Text.of('Shift ').yellow(), Text.of('to see more info.').gold()])
    } else {
      text.add(1, Text.green('Gives positive effects to players in a range').bold(true))
      text.add(2, Text.red('Requires a base built out of precious metals or gems to function!'))
      text.add(3, [Text.white('Iron, '), Text.aqua('Diamonds, '), Text.gold('Gold '), Text.white('or even '), Text.green('Emeralds '), Text.white('are valid base blocks!')])
    }
  })

  // Neat utility to display NBT in the tooltip
  tooltip.addAdvanced(Ingredient.all, (item, advanced, text) => {
    if (tooltip.alt && item.nbt) {
      text.add(Text.of('NBT: ').append(Text.prettyPrintNbt(item.nbt)))
    }
  })
})
```

```
}  
})  
  
// Show the name of the player who owns the skull in a skulls tooltip  
tooltip.addAdvanced('minecraft:player_head', (item, advanced, text) => {  
  let playername = item.nbt?.SkullOwner?.Name  
  if (playername) {  
    text.add(Text.red(`The head of ${playername}`))  
  }  
})  
})
```

Worldgen Events

These following examples will only work on **1.18+!** If you need examples for 1.16, you can look [here](#) if you want to add new features to world generation and [here](#) if you want to remove features from it.

General Notes

Biome Filters:

Biome filters work similarly to *recipe filters*, and can be used to create complex and exact filters to fine tune exactly where your features may and may not spawn in the world. They are used for the `biomes` field of a feature and may look something like this:

```
onEvent('worldgen.add', event => {
  event.addOre(ore => {
    // let's look at all of the 'simple' filters first
    ore.biomes = 'minecraft:plains' [ ] // only spawn in exactly this biome
    ore.biomes = /^minecraft:.*[ ] // spawn in all biomes that match the given pattern (here: anything that starts
    with minecraft:)

    ore.biomes = '#minecraft:is_forest' [ ] // [1.19+] spawn in all biomes tagged as 'minecraft:is_forest'
    ore.biomes = '^nether' [ ] // [1.18 only!] spawn in all biomes in the 'nether' category (see Biome Categories)
    ore.biomes = '$hot'[ ] // [Forge 1.18 only!] spawn in all biomes that have the 'hot' biome type (see Biome
    Dictionary)

    // filters can be arbitrarily combined using AND, OR and NOT logic
    ore.biomes = { } [ ] // empty AND filter, always true
    ore.biomes = [ ] [ ] // empty OR filter, also always true
    ore.biomes = {
      not: 'minecraft:ocean'[ ] // spawn in all biomes that are NOT 'minecaraft:ocean'
    }

    // since AND filters are expressed as maps and expect string keys,
    // all of the 'primitive' filters can also be expressed as such
    ore.biomes = { [ ] // see above for an explanation of these filters
      id: 'minecraft:plains',
      id: /^minecraft:.*[ ] // regex (also technically an id filter)
```

```

tag: '#minecraft:is_forest',
category: '^nether',
biome_type: '$hot',
}
// note all of the above syntax may be mixed and matched individually
// for example, this will spawn the feature in any biome that is
// either plains, or a hot biome that is not in the nether or savanna categories
ore.biomes = [
'minecraft:plains', {
biome_type: '$hot',
not: [
'#nether',
{ category: 'savanna' }
]
},
]
})
})

```

Rule Tests and Targets:

In 1.18, Minecraft worldgen has changed to a "target"-based replacement system, meaning you can specify specific blocks to be replaced with specific other blocks within the same feature configuration. (This is used to replace stone with the normal ore and deepslate with the deepslate ore variant, for example).

Each target gets a "rule test" as input (something that checks if a given block state should be replaced or not) and produces a specific output block state. On the KubeJS script side, both of these concepts are expressed as the same class: *BlockStatePredicate*.

Syntax-wise, BlockStatePredicate is pretty similar to biome filters as they too can be combined using AND or OR filters (which is why we will not be repeating that step here), and can be used to match one of three different things fundamentally:

1. Blocks (these are simply parsed as strings, so for example `"minecraft:stone"` to match Stone)
2. Block States (these are parsed as the block id followed by an array of properties, so you would use something like `"minecraft:furnace[lit=true]"` to match only furnace blocks that are active (lit). You can use F3 to figure out a block's properties, as well as possible values through using the debug stick.
3. Block Tags (as you might expect, these are parsed in the "familiar" tag syntax, so you could for example use `"#minecraft:base_stone_overworld"` to match all types of stone that can

be found generating in the ground in the overworld. Note that these are **block** tags, not **item** tags, so they may (and probably will) be different! (F3 is your friend!)

You can also use regular expressions with block filters, so `/^mekanism:.*_ore$/` would match any block from Mekanism whose id ends with "_ore". Keep in mind this will *not* match block state properties!

When a RuleTest is required instead of a BlockStatePredicate, you can also supply that rule test directly in the form of a JavaScript object (it will then be parsed the same as vanilla would parse JSON or NBT objects). This can be useful if you want rule tests that have a random chance to match, for example!

More examples on how targets work can be found in the example script down below.

Height Providers:

Another system that may appear a bit confusing at first is the system of "height providers", which are used to determine at what Y level a given ore should spawn and with what frequency. Used in tandem with this feature are the so-called "vertical anchors", which may be used to get the height of something relative to a specific anchor point (for example the top or bottom of the world)

In KubeJS, this system has been simplified a bit to make it easier to use for script developers: To use the two most common types of ore placement, *uniform* (the feature has the same chance to spawn anywhere in between the two anchors) and *triangle* (the feature is more likely to spawn in the center of the two anchors than it is to spawn further outwards), you can use the methods `uniformHeight` and `triangleHeight` in `AddOreProperties`, respectively. Vertical anchors have also been simplified, as you can use the `aboveBottom / belowTop` helper methods in `AddOreProperties`, or, in newer KubeJS versions, the builtin class wrapper for `VerticalAnchor` (Note that the former has been deprecated in favour of the latter), as well as if you want to specify absolute heights as simple numbers, instead.

Once again, see the example script for more information!

(1.18 only!) Biome Categories:

Biome categories are a vanilla system that can be used to roughly sort biomes into predefined categories, which are noted below. Note that other mods *may* add more categories through extending the enum, however since there is no way for us to know this we will only provide you with the vanilla IDs here:

- taiga
- extreme_hills
- jungle
- mesa

- plains
- savanna
- icy
- the_end
- beach
- forest
- ocean
- desert
- river
- swamp
- mushroom
- nether

(1.18 and Forge only!) Biome Dictionary:

Much like Vanilla biome categories, Forge uses a "Biome Dictionary" to sort biomes based on their properties. Note that this system is *designed* to be extended by mods, so there is no way for us to give a complete list of all categories to you, however some of the ones you might commonly find yourself using are listed here:

- hot
- cold
- wet
- dry
- sparse
- dense
- spooky
- dead
- lush
- etc.... see [BiomeDictionary](#) for more

In 1.19, both of these systems have been removed **with no replacement** in favour of biome tags!

Example script: (kubejs/startup_scripts/worldgen.js)

```
onEvent('worldgen.add', event => {
  // use the anchors helper from the event (NOTE: this requires newer versions of KubeJS)
  // if you are using an older version of KubeJS, you can use the methods in the ore properties class
  const { anchors } = event

  event.addOre(ore => {
    ore.id = 'kubejs:glowstone_test_lmao' // (optional, but recommended) custom id for the feature
```

```

ore.biomes = {
  not: '^savanna' // biome filter, see above (technically also optional)
}

// examples on how to use targets
ore.addTarget('#minecraft:stone_ore_replaceables', 'minecraft:glowstone') // replace anything in the vanilla
stone_ore_replaceables tag with Glowstone
ore.addTarget('minecraft:deepslate', 'minecraft:nether_wart_block') // replace Deepslate with Nether Wart
Blocks
ore.addTarget([
  'minecraft:gravel', // replace gravel...
  '/minecraft:(.*)_dirt/' // or any kind of dirt (including coarse, rooted, etc.)...
], 'minecraft:tnt') // with TNT (trust me, it'll be funny)

ore.count([15, 50]) // generate between 15 and 50 veins (chosen at random), you could use a single
number here for a fixed amount of blocks
.squared() // randomly spreads the ores out across the chunk, instead of generating them in a
column
.triangleHeight(16) // generate the ore with a triangular distribution, this means it will be more likely to be
placed closer to the center of the anchors
anchors.aboveBottom(32), // the lower bound should be 32 blocks above the bottom of the world, so in
this case, Y = -32 since minY = -64
anchors.absolute(96) // the upper bound, meanwhile is set to be just exactly at Y = 96
) // in total, the ore can be found between Y levels -32 and 96, and will be most likely at Y = (9
32) / 2 = 32

// more, optional parameters (default values are shown here)
ore.size = 9 // max. vein size
ore.noSurface = 0.5 // chance to discard if the ore would be exposed to air
ore.worldGenLayer = 'underground_ores' // what generation step the ores should be generated in (see below)
ore.chance = 0 // if != 0 and count is unset, the ore has a 1/n chance to generate per chunk
})

// oh yeah, and did I mention lakes exist, too?
// (for now at least, Vanilla is likely to remove them in the future)
event.addLake(lake => {
  lake.id = 'kubejs:funny_lake' // BlockStatePredicate
  lake.chance = 4
  lake.fluid = 'minecraft:lava'
  lake.barrier = 'minecraft:diamond_block'

```

```

    })
  })

onEvent('worldgen.remove', event => {
  console.info('HELP')
  //console.debugEnabled = true;

  // print all features for a given biome filter
  event.printFeatures('', 'minecraft:plains')

  event.removeOres(props => {
    // much like ADDING ores, this supports filtering by worldgen layer...
    props.worldgenLayer = 'underground_ores'
    // ...and by biome
    props.biomes = [
      { category: 'icy' },
      { category: 'savanna' },
      { category: 'mesa' }
    ]

    // BlockStatePredicate to remove iron and copper ores from the given biomes
    // Note tags may NOT be used here, unfortunately...
    props.blocks = ['minecraft:iron_ore', 'minecraft:copper_ore']
  })

  // remove features by their id (first argument is a generation step)
  event.removeFeatureById('underground_ores', ['minecraft:ore_coal_upper', 'minecraft:ore_coal_lower'])
})

```

Generation Steps

1. raw_generation
2. lakes
3. local_modifications
4. underground_structures
5. surface_structures
6. strongholds
7. underground_ores
8. underground_decoration
9. vegetal_decoration
10. top_layer_modification

It's possible you may not be able to generate some things in their layer, like ores in dirt, because dirt hasn't spawned yet. So you may have to change the layer to one of the above generation steps by calling `ore.worldgenLayer = 'top_layer_modification'`. This is, however, not recommended.

Chat Event

This script is peak of human evolution. Whenever someone says "Creeper" in chat, it replies with "Aw man".

```
onEvent('player.chat', (event) => {  
  // Check if message equals creeper, ignoring case  
  if (event.message.trim().equalsIgnoreCase('creeper')) {  
    // Schedule task in 1 tick, because if you reply immediately, it will print before player's message  
    event.server.scheduleInTicks(1, event.server, (callback) => {  
      // Tell everyone Aw man, colored green. Callback data is the server  
      callback.data.tell(Text.green('Aw man'))  
    })  
  }  
})
```

Another example, cancelling the chat event. No need to schedule anything now, because player's message won't be printed,

```
onEvent('player.chat', (event) => {  
  // Check if message equals creeper, ignoring case  
  if (event.message.startsWith('!some_command')) {  
    event.player.tell('Hi!')  
    event.cancel()  
  }  
})
```

Custom Fluids

Supported by Forge on all versions, and Fabric on 1.18.2+

```
// Startup script
onEvent('fluid.registry', event => {
  // These first examples are 1.16.5 and 1.18.2 syntax

  // Basic "thick" (looks like lava) fluid with red tint
  event.create('thick_fluid')
    .thickTexture(0xFF0000)
    .bucketColor(0xFF0000)
    .displayName('Thick Fluid')

  // Basic "thin" (looks like water) fluid with cyan tint, has no bucket and is not placeable
  event.create('thin_fluid')
    .thinTexture(0x00FFFF)
    .bucketColor(0x00FFFF)
    .displayName('Thin Fluid')
  fluid.noBucket() // both these methods are 1.18.2+ only
  fluid.noBlock()

  // Fluid with custom textures
  event.create('strawberry_cream')
    .displayName('Strawberry Cream')
    .stillTexture('kubejs:block/strawberry_still')
    .flowingTexture('kubejs:block/strawberry_flow')
    .bucketColor(0xFF33FF)

  // For 1.18.1 the syntax is slightly different
  event.create('thick_fluid', fluid => {
    fluid.textureThick(0xFF0000) // the texture method names are different in 1.18.1 and below, textureXyz
    // instead of xyzTexture
    fluid.bucketColor(0xFF0000)
    fluid.displayName('Thick Fluid')
  })
})
```

In 1.18.1, 1.17 and 1.16 the texture method names are swapped, so textureStill and textureThin instead of stillTexture and thinTexture

Methods that you can use after the event.create('name')

- displayName(name)
- color(color)
- bucketColor(color)
- builtinTextures()
 - same as thinTexture(0xFFFFFFFF)
- stillTexture(path)
 - path is the path to texture is for example maybe "minecraft:block/sand"
 - this texture is recommended to be 16x16, or if animated with a mcmeta file then 16x48 for 3 frames or 16x80 for 5 or 16x240 for 15
 - Frame counts of 3, 5, 15, 6, 10, or 30 will make your life easier, because the flowing animation need to be a multiple of 15 to look good
- flowingTexture(path)
 - path is the path to texture is for example maybe "minecraft:block/sand"
 - this texture is recommended to be 32x480 and animated with a mcmeta file
 - each frame is recommended to be 32x32 (recommended to be the same 16x16 texture tiled)
 - then each of these frames are shifted one pixel vertically from the previous, so it looks like its moving
 - If you are going to be making your own flowing fluid texture it is *highly recommended* to not make these by hand (It is hours of suffering), and instead write a some program, or setup something with blender nodes to make it.
- noBucket()
- noBlock()
- gaseous()
 - It is now a gas
- rarity(value)
 - Can be:
 - "common"
 - "uncommon"
 - "rare"
 - "epic"

The following can also be used but have no effect unless a mod adds a purpose:

- luminosity(value)
 - default 0
- density(value)
 - default 1000
- temperature(value)
 - default 300
- viscosity(value)

- default 1000

There is a good chance the following does not work at all

You can use `.bucketItem` to get the bucket item builder.

If you one want to use it then you can place it at the end of the other methods then use the its methods instead.

```
// notice this script has not been tested
onEvent('fluid.registry', event => {
  event.create('taco_suace')
    .thickTexture(0xFF0000)
    .bucketColor(0xFF0000)
    .bucketItem
    .group("food")
})
```

Some amount of the methods in these builders will not work or cause problems

- `.bucketItem`
 - Any method that you can use on an [item builder](#) might work

Command Registry

This page is unfinished and only provides basic information

Example:

The following code has not been completely tested on 1.18 and not at all on 1.16

```
onEvent("command.registry", event => { //command registry event
  const { commands: Commands, arguments: Arguments } = event;
  event.register( //register a new command
    Commands.literal("myCommand") //the command is called myCommand
    [] .requires(src => src.hasPermission(2)) //2 is op. This line is optional, but you can also instead of just one value,
    wrap it in {}s and use return to write a more complex requirement checks
    [] .then(Commands.argument('arg1', Arguments.STRING.create(event))) //takes argument string called arg1. You
    can have as many (or none) as you want.
    [] .then(Commands.argument('arg2', Arguments.FLOAT.create(event))) //takes argument float called arg2. The oth
    type you can use can be found with ProbeJS
    [] .executes(ctx => { //run the command
      [] [] [] const arg1 = Arguments.STRING.getResult(ctx, "arg1"); //get recipe
      [] [] [] const arg2 = Arguments.FLOAT.getResult(ctx, "arg2"); //get the value
      //your code goes here
      [] [] [] if(arg1 == "example")
        [] return 0 //return 0 means command did not work
        let level = ctx.source.level.asKJS()
        let position = ctx.source.position
        //hurt entities in a around a area of where the command was run
        let i = 0
        level.getEntitiesWithin(AABB.of(position.x()-2,position.y()-2,position.z()-
        2,position.x()+2,position.y()+2,position.z()+2)).forEach(entity => {
          [] if (entity.living) {
            entity.attack(arg2)
            i++
            if (entity.type == "minecraft:player") entity.tell(arg1) //tell players that got hurt the message
            that is arg1
          }
        })
      return i
    })
  }
```

```
}
```

```
})
```

```
return i // always return something
```

```
})
```

```
// every then requires a ')' so dont forget them
```

```
//but requires does not
```

```
)
```

```
})
```

Datapack Load Events

You can load json datapack files programmatically!

```
onEvent('server.datapack.first', event => {  
  event.addJson(name, json)  
})
```

`resourceLocation` could be `minecraft:loot_tables/entities/villager.json`

`json` could be for example:

```
{  
  type: "entity",  
  pools: [  
    {  
      rolls: 2,  
      bonus_rolls: 1,  
      entries: [  
        {  
          type: "item",  
          weight: 3,  
          name: "minecraft:emerald"  
        },  
        {  
          type: "empty",  
          weight: 2  
        }  
      ]  
    }  
  ]  
}
```

Note: Practically everything in vanilla has a way better to programmatically load it, so it is recommended to use this mostly for loading thing for other mods

There are different timing that you can make the file get loaded too!

- `server.datapack.first`
- `server.datapack.last`

This event is useful, because instead of needing to write multiple json files, you can write one then change the values passed to it.

Example

Adds multiple advancements for using different items that reward experience:

```
onEvent('server.datapack.first', event => {
  const items = ['bow', 'golden_hoe', 'flint_and_steel', 'spyglass']
  items.forEach(item => {
    event.addJson(`kubejs:advancements/${item}`, {
      criteria: {
        requirement: {
          trigger: "minecraft:using_item",
          conditions: {
            item: {
              items: [`minecraft:${item}`]
            }
          }
        }
      },
      rewards: {
        experience: 20
      }
    })
  })
})
```

In the `custom_machinery` mod, the packdev needs to make a massive json file for each machine they wish to create. This script will make 16 machines, and is shorter then even a single on of the original json files would have been.

```
onEvent('server.datapack.first', event => {
  let json
  //create 16 custom machines with 6 inputs and 1 output
  for (let machineNumber = 0; machineNumber < 16; machineNumber++) {

    json = {
      name: {
```

```

    text: `${machineNumber}`
  },
  appearance: {},
  components: [
    {
      type: "custommachinery:item",
      id: "out",
      mode: "output"
    },
    {
      gui: [
        {
          type: "custommachinery:progress",
          x: 70,
          y: 41,
          width: 18,
          height: 18
        }, {
          type: "custommachinery:slot",
          x: 88,
          y: 41,
          id: "out"
        }, {
          type: "custommachinery:text",
          text: `Machine ${machineNumber}`, //string builder to make the name match the machine number
          x: 16,
          y: 16
        }, {
          type: "custommachinery:texture",
          x: 0,
          y: 0,
          texture: "custommachinery:textures/gui/base_background.png",
          priority: 1000
        }, {
          type: "custommachinery:player_inventory",
          x: 16,
          y: 68
        }
      ]
    }
  ]
}

```

```

//add the input slots and corresponding componets
let slotNumber = 0

const xValues = [16,34,52]
const yValues = [32,50]

xValues.forEach(x => {
  yValues.forEach(y => {

    json.components.push({
      type: "custommachinery:item",
      id: `input${slotNumber}`,
      mode: "input"
    })

    json.gui.push({
      type: "custommachinery:slot",
      x: x,
      y: y,
      id: `input${slotNumber}`
    })

    slotNumber++
  })
})

//add the json
event.addJson(`kubejs:machines/${machineNumber}`,json)
}
})

```