Global

Constants, classes and functions

- Components, KubeJS and you!
- Item and Ingredient

Components, KubeJS and you!

In 1.18.2 and beyond KubeJS uses Components in a lot of places. It returns them for entity names, item names and accepts them for everything from tooltips to sending messages to players.

> All examples use `event.player.tell` from the `player.chat` event to output their example, but they will with anywhere that accepts a Component!

Making your own Components starts from the ComponentWrapper class, invokable with just `Component` or `Text` from anywhere. The examples all use `Component` but `Text` works just the same.

ComponentWrapper methods:

Name	Return Type	Info
of(Object o)	MutableComponent	Returns a component based on what was input. Accepts strings, primitives like numbers, snbt/nbt format of Components and a couple others.
clickEventOf(Object o)	ClickEvent	Returns a ClickEvent based on what was input. See examples below
prettyPrintNbt(Tag tag)	Component	Returns a component with a prettified version of the input NBT.
join(MutableComponent seperator, Iterable<? extends Component> texts)	MutableComponent	Returns the result of looping through `texts` and joining them, separating each one with `seperator`.
string(String text)	MutableComponent	Returns a basic unformatted TextComponent with just the input text
translate(String key)	MutableComponent	Returns a basic unformatted TranslatableComponent with the input key.
translate(String key, Object... objects)	MutableComponent	Returns an unformatted TranslatableComponent with `objects` as the replacements for %s in the translation output.

Name	Return Type	Info
keybind(String keybind)	MutableComponent	Returns a basic unformatted KeybindComponent with the specified keybind.
<color>(Object text)	MutableComponent	Returns a basic Component with the specified color for text coloring. Valid colors are in the list below. Do not include the <> brackets.

A list of colors accepted in various places:

- black
- darkBlue
- darkGreen
- darkAqua
- darkRed
- darkPurple
- gold
- gray
- darkGray
- blue
- green
- aqua
- red
- lightPurple
- yellow
- white

Basic examples:

```
onEvent('player.chat', event => {
  // Tell the player a normal message
  event.player.tell(Component.string('Hello world'))
  // Now in black
  event.player.tell(Component.black('Welcome to the dark side, we have cookies!'))
  // Tell them the diamond item, in whatever language they have set
  event.player.tell(Component.translate('item.minecraft.diamond'))
  // Now tell them whatever key they have crouching set to
  event.player.tell(Component.keybind('key.sneak'))
  // And finally show them the nbt data of the item they are holding
  event.player.tell(Component.prettyPrintNbt(event.player.mainHandItem.nbt))
})
```

MutableComponent

These are methods you can call on any MutableComponent. This includes ComponentKJS, which is a KubeJS extension for vanilla's components and is injected into vanillas code on runtime. All methods from ComponentKJS are included, but only relevant ones from vanilla are included.

Name	Return Type	Info
iterator()	Iterator<Component>	Returns an Iterator for the components contained in this component, useful for when multiple have been joined or appended. From ComponentKJS.
self()	MutableComponent	Returns the component you ran it on. From ComponentKJS.
toJson()	JsonElement	Returns the Json representation of this Component. From ComponentKJS.
<color>()	MutableComponent	Modifies the Component with the specified color applied as formatting, and returns itself. Do not include the <> brackets. From ComponentKJS.
color(Color c)	MutableComponent	Modifies the Component to have the input Color, and returns itself. (Color is a Rhino color). From ComponentKJS.
noColor()	MutableComponent	Modifies the Component to have no color, and returns itself. From ComponentKJS.
bold() italic() underlined() strikethrough() obfuscated()	MutableComponent	Modifies the Component to have said formatting and returns itself. From ComponentKJS.
bold(@Nullable Boolean value) italic(@Nullable Boolean value) underlined(@Nullable Boolean value) strikethrough(@Nullable Boolean value) obfuscated(@Nullable Boolean value)	MutableComponent	Modifies the Component to have said formatting and returns itself. From ComponentKJS.

Name	Return Type	Info
insertion(@Nullable String s)	MutableComponent	Makes the Component insert the specified string into the players chat box when shift clicked (does not send it) and returns itself. From ComponentKJS.
font(@Nullable ResourceLocation s)	MutableComponent	Changes the Components font to the specified font and returns itself. For more information on adding fonts see the Minecraft Wiki's Resource packs page. From ComponentKJS.
click(@Nullable ClickEvent s)	MutableComponent	Sets this components ClickEvent to the specified ClickEvent. From ComponentKJS.
hover(@Nullable Component s)	MutableComponent	Sets the hover tooltip for this Component to the input Component. From ComponentKJS.
setStyle(Style style)	MutableComponent	Sets the style to the input Style (net.minecraft.network.chat.Style) and returns itself. Not recommended for use, use the specific methods added by CompontentKJS instead.
append(String string)	MutableComponent	Appends the input string as a basic TextComponent to this Component then returns itself.
append(Component component)	MutableComponent	Appends the input Component to this Component then returns itself.
withStyle(Style style)	MutableComponent	Merges the input style with the current style, preffering properties from the new style if a conflict exists.
getStyle()	Style	Returns this Components current Style.
getContents()	MutableComponent	Returns this Components contents. Will return the text for TextComponents, the pattern for SelectorComponents and an empty string for all other Components.
getSiblings()	List<Component>	Returns a list of all Components which have been append()ed to this Component
plainCopy()	BaseComponent	Returns a basic copy of this, preserving only the contents and not the style or siblings.
copy()	MutableComponent	Returns a full copy of this Component, preserving style and siblings

Name	Return Type	Info
getString()	String	Returns this components text as a String. Will return a blank string for any non-text component

More complex examples:

```
// First a prefix, like a rank. This won't be changing so we can just declare it up here.
const prefix = Component.darkRed('[Admin]').underlined()

onEvent('player.chat', event => {

  // First cancel the event because we are going to be sending the message ourselves
  event.cancel()

  // The main Component we will be adding stuff to. It is just a copy of the prefix component for now
  let component = prefix.copy() // If we didn't copy it all the modifications we made to it would be applied to the original as well!

  // Make a component of the players name and then surround with < > and make it white again. Then append it our main copmponent.
  // A component will inherit any styiling it doesnt have from whatever it has been .append()ed to, so you need to apply formatting rather liberally some times!
  let playerName = Component.string(event.getUsername())
  // Doing it this way means we only have to apply the white formatting and no underline once to the name
  let nameComponent = Component.white(' <').underlined(false).append(playerName).append('> ')
  component.append(nameComponent)

  // Finnally add the message (obfuscated, of course) and send it!
  // We make sure to set its color and underline though, otherwise it will end up inheriting the red and underline from the prefix!
  component.append(Component.string(event.message).obfuscated().white().underlined(false))
  event.server.tell(component)

})
```

Item and Ingredient

When making recipes you can specify items in many ways, the most common is just to use `'namspace:id'`, like `'minecraft:diamond'`, however you can also use `Item#of` and `Ingredient#of` for advanced additions, such as NBT or count.

Note that Item and Ingredient are **not** the same! They may work similarly but there are differences. Item can only ever represent a single item type whereas Ingredient can represent multiple item types (and multiple instances of the same item type with different properties such as NBT data). For most cases Ingredient should be preferred over Item.

Item/ItemWrapper

Its Java class name is ItemWrapper but it is bound to Item in JS.

Name	Return Type	Info
of(ItemStackJS in)	ItemStackJS	Returns an ItemStackJS based on what was input. Note that this relies mostly on Rhinos type wrapping to function, see paragraph below about ItemStackJS#of for more info
of(ItemStackJS in, int count)	ItemStackJS	See above. count will override any other count set from the first parameter.
of(ItemStackJS in, CompoundTag tag)	ItemStackJS	See above. NBT is merged, with the input NBT taking priority over existing NBT.
of(ItemStackJS in, int count, CompoundTag nbt)	ItemStackJS	Combines the functionality of the above two.
withNBT(ItemStackJS in, CompoundTag nbt)	ItemStackJS	Same as the corresponding #of.
withChance(ItemStackJS in, double chance)	ItemStackJS	Same as #of, chance will override currently set chance.
getList()	ListJS	Returns a list of ItemStackJS, one per registered item.
getTypeList()	ListJS	Returns a list of String, one per registered item.

Name	Return Type	Info
getEmpty()	ItemStackJS	Returns ItemSTackJS.EMPTY
clearListCache()	void	Clears the caches used for #getList and #getTypeList
fireworks(Map<String, Object> properties)	FireworkJS	Returns a FireworkJS based on the input map of propeties. See FireworkJS#of on the FireworkJS page for more information <TODO: Make and link FireworkJS page>
getItem(ResourceLocation id)	Item	Returns the instance of the Item class associated with the item id passed in.
@Nullable findGroup(String id)	CreativeModTab	Returns the Creative tab associated with the id passed in, returns null if none found.
exists(ResourceLocation id)	boolean	Returns if the item id passed in exists or not.
isItem(@Nullable Object o)	boolean	Just does an instanceof ItemStackJS check on the object passed in.

Item#of relies on Rhinos type wrapping to function, which calls ItemStackJS#of. This tries its best to turn the input into an ItemStackJS. If no match is found ItemStackJS.EMPTY is returned. Valid inputs:

- null/ItemStack.EMPTY/Items.EMPTY/ItemStackJS.EMPTY - will return ItemStackJS.EMPTY
- ItemStackJS - will return the same object passed in.
- FluidStackJS - will return a new DummyFluidItemStackJS
- IngredientJS - will return the first item in the Ingredient
- ItemStack - will return a new ItemStackJS wrapping the ItemStack passed in
- ResourceLocation - will lookup this ResourceLocation in the item registry and return it if found. If not found will return ItemStackJS.EMPTY, and throw an error if RecipeJS.itemErrors is true
- ItemLike - will return a new ItemStackJS of the input
- JsonObject - will return an item based on properties in the Json. `item` will be used as the item id, or `tag` if item does not exist. `count`, `chance` and `nbt` all set their respective properties
- RegEx - will return a new ItemStackJS of the first item id that matches this regex.
- String (CharSequence) - will parse it and return a new ItemStackJS based on the input item id. Prefix with `nx` to change the count (where n is any number between 1 and 64). Put `#` before the item id to parse it as a tag instead. Put `@` before the item id to parse it as a modid instead. Prefix with `%` to parse it as a creative menu tab group. Surround in `/` to parse as a RegEx. NOTE: will only be the first item in any of the groups mentioned above!

- Map/JS Object - uses the same rules as a JsonObject.

Ingredient/IngredientWrapper

Its Java class name is IngredientWrapper but it is bound to Ingredient in JS. All static methods.

Name	Return Type	Info
getNone()	IngredientJS	Returns ItemStack.EMPTY
getAll()	IngredientJS	Returns an IngredientJS of every single item in game. All of them.
of(Object object)	IngredientJS	Works exactly the same as Item#of except it recognises Ingredient and forge json ingredient syntax.
of(Object object, int count)	IngredientJS	Same as above. The count passed in will override any from the first parameter.
custom(Predicate<ItemStackJS> predicate)	IngredientJS	Takes the arrow function or anonymous function passed in and makes an IngredientJS with that as IngredientJS#test. Return true from the function if the ItemStackJS passed should match as an ingredient.
custom(IngredientJS in, Predicate< ItemStackJS> predicate)	IngredientJS	Same as above except it must match the IngredientJS passed in as the first parameter before the custom function is called.
customNBT(IngredientJS in, Predicate <CompoundTag> predicate)	IngredientJS	Same as above except the Predicate is passed the items NBT instead of the full ItemStackJS. Useful for advanced NBT matching.
matchAny(Object objects)	IngredientJS	Adds the passed in object to an ingredient. If it is a list then it adds all items in the list. All objects are passed through #of before adding.
registerCustomIngredientAction(String id, CustomIngredientActionCallback callback)	void	Registers a custom ingredient action. See the recipe page for more information.
isIngredient(@Nullable Object o)	boolean	Just does an instanceof IngredientJS check on the object passed in.

> Remember that Item and Ingredient are not equivalent!

Examples

```
<TODO: examples>
```

ItemStackJS

A wrapper class for vanilla's ItemStack. All methods listed here are instance methods, all useful static methods are wrapped in ItemWrapper. Implements IngredientJS and overrides most of its default methods.

Name	Return Type	Info
getItem()	Item	Returns the instance of the Item class associated with this ItemStackJS.
getItemStack()	ItemStack	Returns the vanilla ItemStack that this wraps.
getId()	String	Returns the item id associated with this ItemStackJS in the form mod_name:item_name
getTags()	Colletion<ResourceLocation>	Returns all item tags the item has. (NOT NBT tags).
hasTag(ResourceLocation tag)	boolean	Returns if the item has the input tag or not.
copy()	ItemStackJS	Returns a copy of this ItemStackJS.
setCount(int count)	void	Sets the count on this ItemStackJS.
getCount()	int	Gets the count.
withCount()	ItemStackJS	Returns a copy of this ItemStackJS with a different count.
isEmpty()	boolean	Returns if this is an empty item or not.
isInvalidRecipeIngredient()	boolean	Returns if this is a valid recipe ingredient.
isBlock()	boolean	Returns if this item is a BlockItem, that is it can be placed and form a block.
@Nullable getNbt()	CompoundTag	Gets this items NBT data.
setNbt(@Nullable CompoundTag tag)	void	Sets this items NBT data

Name	Return Type	Info
hasNBT()	boolean	Returns if this item has NBT data.
getNbtString()	String	Returns this items NBT data as a string. If you want to display it to the player see Text#prettyPrintNbt.
removeNBT()	ItemStackJS	Returns a copy with no NBT data.
withNBT(CompoundTag nbt)	ItemStackJS	Returns a copy with the specified NBT data. Any tags from the original NBT are kept if not overwritten by the NBT passed in.
hasChance()	boolean	Returns if the ItemStackJS has a chance.
removeChance()	void	Removes the chance from this ItemStackJS.
setChance(double c)	void	Sets the chance for this ItemStackJS.
getChance()	double	Returns the chance.
withChance(double c)	ItemStackJS	Returns a copy with the chance passed in, unless the chance passed in is the same as the current chance, in which case it returns this.
getName()	Components	Returns this items name. Probably a Translateable Component unless its been overridden by something else (ie method below).
withName(@Nullable Component displayName)	ItemStackJS	Returns a copy with a different display name set.
toString()	String	Returns a string representing this ItemStackJS. The same method used for the `/kubejs hand` command.
test(ItemStackJS other)	boolean	Returns if this ItemStackJS equals another one. Tests for item type and NBT data.
testVanilla(ItemStack other)	boolean	Returns if this ItemStackJS equals the passed in ItemStack. Tests for item type and NBT data.

Name	Return Type	Info
testVanillaItem(Item item)	boolean	Returns if the Item passed in is the same as this ItemStackJS's Item. Basically checks they are the same item type.
getStacks()	Set<ItemStackJS>	Returns this ItemStackJS as the only entry in a Set.
getVanillaItems()	Set<Item>	Returns this ItemStackJS associated Item as the only entry in a Set.
getFirst()	ItemStackJS	Retuns a copy of this ItemStackJS
hashCode()	int	Returns a hash code of the Item and NBT data.
equals(Object o)	boolean	Returns if this is equal to the input object.
strongEquals(Object o)	boolean	Returns if this is equal to the input object. Checks count as well.
getEnchantments()	MapJS	Returns a MapJS of this itemStackJS enchament id's to their level.
hasEnchantment(Enchantment enchantment, int level)	boolean	Returns if this ItemStackJS is enchanted with a minimum of the passed in enchantment level.
enchant(MapJS enchantments)	ItemStackJS	Enchants a copy of this ItemStackJS with the MapJS passed in (it should be a map of enchantment ids to levels), then returns the copy.
enchant(Enchantment enchantment, int level)	ItemStackJS	Enchants a copy of this item with the passed in Enchantment at the specified level, then returns the copy.
getMod()	String	Returns the mod id of the mod this item is from.
ignoreNBT()	IngredientJS	Returns a new IgnoreNBTIngredientJS of this item.
weakNBT()	IngredientJS	Returns a new WeakNBTIngredientJS of this item.

Name	Return Type	Info
areItemsEqual(ItemStackJS other)	boolean	Returns if this item type is equal to the item type of the passed in ItemStackJS
areItemsEqual(ItemStack other)	boolean	Returns if this item type is equal to the item type of the passed in ItemStack
isNBTEqual(ItemStackJS other)	boolean	Returns if the NBT of this ItemStackJS is equal to the NBT of the ItemStackJS passed in.
isNBTEqual(ItemStack other)	boolean	Returns if the NBT of this ItemStackJS is equal to the NBT of the ItemStack passed in.
getHarvestSpeed(@Nullable BlockContainerJS block)	float	Returns the mining speed of this ItemStackJS if used to mine the passed in BlockContainerJS
getHarvestSpeed()	float	Returns this items default mining speed
toJson() toResultJson() toRawResultJson()	JsonElement	Returns a Json representation of this ItemStackJS. They all appear to work almost identically.
toNBT()	CompoundTag	Returns an NBT representation of this ItemStackJS, the same sort that vanilla uses to store items in blocks.
onChanged(@Nullable Tag o)	void	Sets the items NBT data to the tag passed in, only if it is a CompoundTag or null.
getItemGroup()	String	Returns the name of the creative tab this item belongs in. An empty string if it does not exist in the creative tabs (like a jigsaw block).
getItemIds()	Set<String>	Returns a set with this items id as the only entry.
getFluidStack()	FluidStackJS	Returns null, by default. Overriden by some superclasses to return the FluidStackJS that this item represents.
getTypeData()	CompoundTag	Unknown purpose.

```
<TODO: Examples>
```