beJS

Download: CurseForge

> The custom BlockEntity event is a startup event.

Block Entities

Custom BlockEntities are created in a startup script. They cannot be reloaded without restarting the game. The event is not cancellable.

```
StartupEvents.registry('block', event => {
  event.create('example_block', 'entity' /*has to be here for the BE builder to work*/).displayName('Example Block')
  .entity(builder => { // adds a BlockEntity onto this block
    builder.ticker((level, pos, state, be) => { // a tick method, called on block entity tick
      if(!level.clientSide) { // ALWAYS check side, the tick method is called on both CLIENT and SERVER
        if(level.levelData.gameTime % 20 == 0) { // only .levelData.gameTime works for some reason??
          if(level.getBlockState(pos.above()) === Blocks.AIR.defaultBlockState()) {
            level.setBlock(pos.above(), Blocks.GLASS.defaultBlockState(), 3)
            be.persistentData.putBoolean("placed", true)
          } else {
            level.setBlock(pos.above(), Blocks.AIR.defaultBlockState(), 3)
            be.persistentData.putBoolean("placed", false)
          }
          console.info("placed: " + be.persistentData.getBoolean("placed"))
        }
      }
    }).defaultValues(tag => tag = { progress: 0, example_value_for_extra_saved_data: '0mG this iz Crazyyy'}) // adds a 'default' saved value, added on block entity creation (place etc)
                                                                                                              // [1st param: CompoundTag consumer]
    .addValidBlock('example_block') // adds a valid block this can attach to, useless in normal circumstances (except if you want to attach to multiple blocks or are building the BE separately)
    .itemHandler(27) // adds a basic item handler to this block entity, use something like PowerfulJS for more advanced functionality
```

```
                    // [1st param: slot count]
        .energyHandler(10000, 1000, 1000) // adds a basic FE handler, same as above
                                    // [1st param: max energy, 2nd param: max input, 3rd param: max output]
        .fluidHandler(1000, stack => true) // adds a basic fluid handler
            □                       // [1st param: max amount, 2nd param: fluid filter]
    })
})
```

alternatively, you can create the BlockEntity separately and attach it with `EntityBlockJS.Builder#entity('kubejs:be_id')`

```
StartupEvents.registry('block_entity_type', event => {
□event.create('example_block')
□.ticker((level, pos, state, be) => { // a tick method, called on block entity tick
        if(!level.clientSide) { // ALWAYS check side, the tick method is called on both CLIENT and SERVER
            if(level.levelData.gameTime % 20 == 0) { // only .levelData.gameTime works for some reason??
                if(level.getBlockState(pos.above()) === Blocks.AIR.defaultBlockState()) {
                    level.setBlock(pos.above(), Blocks.GLASS.defaultBlockState(), 3)
                } else {
                    level.setBlock(pos.above(), Blocks.AIR.defaultBlockState(), 3)
                }
            }
        }
    }).saveCallback((level, pos, be, tag) => { // called on BlockEntity save, don't see why you would ever need these tbf, but they're here
        tag.putInt("tagValueAa", be.getPersistentData().getInt('progress'))
    }).loadCallback((level, pos, be, tag) => { // called on BlockEntity load, same as above
         be.getPersistentData().putInt("progress", tag.getInt("tagValueAa"))
    }).defaultValues(tag => tag = { progress: 0, example_value_for_extra_saved_data: '0mG this iz Crazyyy'}) // adds a 'default' saved value, added on block entity creation (place etc)
                                                                                    // [1st param: CompoundTag consumer]
    .addValidBlock('example_block') // adds a valid block this can attach to, useless in normal circumstances (except if you want to attach to multible blocks)
    .hasGui() // if ScreenJS is installed, marks this blockentity as having a GUI, doesn't do anything otherwise
    .itemHandler(27) // adds a basic item handler to this block entity, use something like PowerfulJS for more advanced functionality
                    // [1st param: slot count]
    .energyHandler(10000, 1000, 1000) // adds a basic FE handler, same as above
                                // [1st param: max energy, 2nd param: max input, 3rd param: max output]
```

```
    .fluidHandler(1000, stack => true) // adds a basic fluid handler
                          □           // [1st param: max amount, 2nd param: fluid filter]
})
```

all valid methods available on all builders:

- `addValidBlock('block_id')`
- `ticker((level, pos, state, blockEntity) => ...)`
- `defaultValues(tag => ...)`
- `itemHandler(capacity)`
- `energyHandler(capacity, maxReceive, maxExtract)`
- `fluidHandler(capacity, fluidStack => isValid)`

Multiblocks

multiblock builder example:

```
StartupEvents.registry('block', event => {
    let CAP_PREDICATE = be => { // has *any* forge capability (item, energy, fluid)
        return be != null && (be.getCapability(ForgeCapabilities.ITEM_HANDLER).present ||
be.getCapability(ForgeCapabilities.FLUID_HANDLER).present ||
be.getCapability(ForgeCapabilities.ENERGY).present)
    }

□event.create('multi_block', 'multiblock').material('metal').hardness(5.0).displayName('Multiblock')
□    .entity(builder => {
□        builder.ticker((level, pos, state, be) => { // tick me here, but ONLY WHEN MULTIBLOCK IS FORMED!!

        □})
        □.pattern(() => { // ordering is: [aisle: z, aisle contents[]: y, single string: x]
        □    return BlockPatternBuilder.start()
        □        .aisle( 'BBB',
                         'ACA',
                         'AAA')
                 .aisle( 'BBB',
                         'AAA',
                         'AAA')
                 .aisle( 'BBB',
```

```
                    'AAA',
                    'AAA')
                .where('A', BlockInWorld.or(BlockInWorld.hasState(BlockPredicate.forBlock('minecraft:iron_block')),
BlockInWorld.hasBlockEntity(CAP_PREDICATE)))
                // ^ iron block OR any capability on a BE
                .where('C', BlockInWorld.hasState(BlockPredicate.forBlock('kubejs:multi_block')))
                // ^ controller block
                .where('B', BlockInWorld.hasState(BlockPredicate.forBlock('minecraft:copper_block')))
                // ^ self explanatory
        })
        })
        .property(BlockProperties.HORIZONTAL_FACING) // block builder stuff, facing direction
        .defaultState(state => {
            state.setValue(BlockProperties.HORIZONTAL_FACING, Direction.NORTH)
        })
        .placementState(state => {
            state.setValue(BlockProperties.HORIZONTAL_FACING, state.horizontalDirection.opposite)
        })
})
```

> currently only 1 input & 1 output per type are set as the multiblock's IO, and it's the last one found in the scan.

extra valid methods on `multiblock` builder:

- `pattern(builder => ...)`

available static methods in `BlockInWorld`:

- `hasState(predicate => ... return boolean)`
- `hasBlockEntity(predicate => ... return boolean)`
- `or(predicate1, predicate2)`
- `and(predicate1, predicate2)`

more advanced example: link

multiblock (and recipe type) example: link

Recipe Types

beJS can create custom recipe types for your block entities to use!

```
StartupEvents.registry('recipe_type', event => {
    event.create('name_here')
        .assembler((recipe, container) => { // optional, but very much suggested
            let results = recipe.results
            for (let i = 0; i < results.size() && i < container.containerSize; ++i) {
                container.setItem(i, results.get(i))
            }
        })
        .maxInputs(2) // required
        .maxOutputs(4) // required
        .toastSymbol('kubejs:block_id_here') // optional
})
```

valid methods on all RecipeType builders:

- `assembler((recipe, container) => ...)`
- `maxInputs(count)`
- `maxOutputs(count)`
- `toastSymbol(stack)`

Item/Fluid Handlers

beJS has multiple custom handlers that have extra functionality:

IMultipleItemHandler

IMultipleItemHandler is an item handler with multiple slots. valid methods listed below:

- `getAllContainers() : List<IItemHandlerModifiable>`
- `getContainer(index) : IItemHandlerModifiable`
- `getStackInSlot(container, slot) : ItemStack`
- `insertItem(container, slot, stack, simulate) : ItemStack`
- `extractItem(container, slot, amount, simulate) : ItemStack`
- `getSlotLimit(container, slot) : int`
- `isItemValid(container, slot, stack) : boolean`
- `setStackInSlot(container, slot, stack)`

IMultipleFluidHandler

IMultipleItemHandler is a fluid handler with multiple slots. valid methods listed below:

- default forge IFluidHandler methods (not listed here)
- `fill(tank, fluidStack, action) : int`
- `drain(tank, fluidStack, action) : FluidStack`
- `drain(tank, maxDrain, action) : FluidStack`

Revision #22
Created 15 February 2023 15:14:50 by Lat
Updated 28 February 2023 10:41:47 by Screret