Digital Twin Toolkit

# Three.js

Three.js is a JavaScript library for displaying 3D models on webpages. We used the library to create models of our physical factory. You will need to be able to develop in JavaScript to follow this section of the guide.

## Three.js Documentation

Site with all of the resources

<https://threejs.org/>

Documentation on the site – Text based

<https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene>

Documentation with examples and videos

<https://threejsfundamentals.org/threejs/lessons/threejs-responsive.html>

Book guide by one of the develops:

<https://discoverthreejs.com/>

Book written by Jos Dirksen

<https://www.packtpub.com/product/learn-three-js-third-edition/9781788833288>

## How we Used Three.js

We created models using the Three.js editor(<https://threejs.org/editor/>) of the various assets of our factory.

Here is a video demonstration:

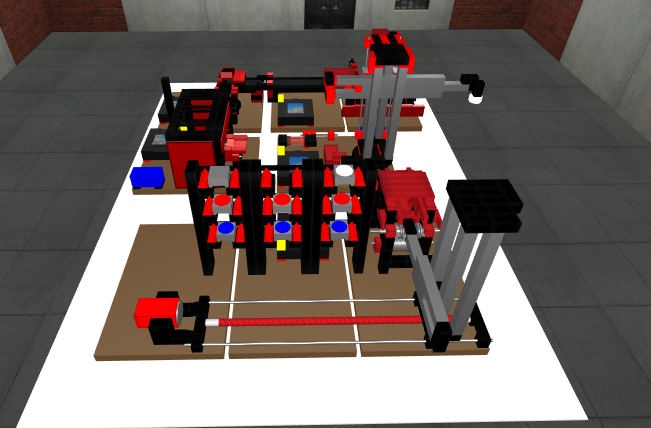


If you already have assets created here are the file types that three.js allows you to import.

Supported model import/export types:

* DAE
* DRC
* GLB
* GLTF
* OBJ
* PLY(Binary/normal)
* STL(Binary/normal)

## Our Finished Model





## Animating Models

We animated the models using Three.js again and used code such as:

Rotating

function craneRotateTo(rotation) {

var crane = glScene.getObjectByName('crane');

if (crane.rotation.y < rotation ){

crane.rotation.y += 0.02;

return false;

}

crane.rotation.y = rotation;

We increase the rotation by 0.02 each time the function is called until it reaches the goal rotation value. We do this with increments so the viewer can see the rotation on screen, else it would just jump to the end rotation.

Moving

function rollTextCoord(value,incr) {

value+=incr;

if (value>1)

value = 0;

if (value<0)

value = 1;

return value

}

var myObject = glScene.getObjectByName( "TrackPlaneA2" );

x = myObject.material.map.offset.x;

y = myObject.material.map.offset.y;

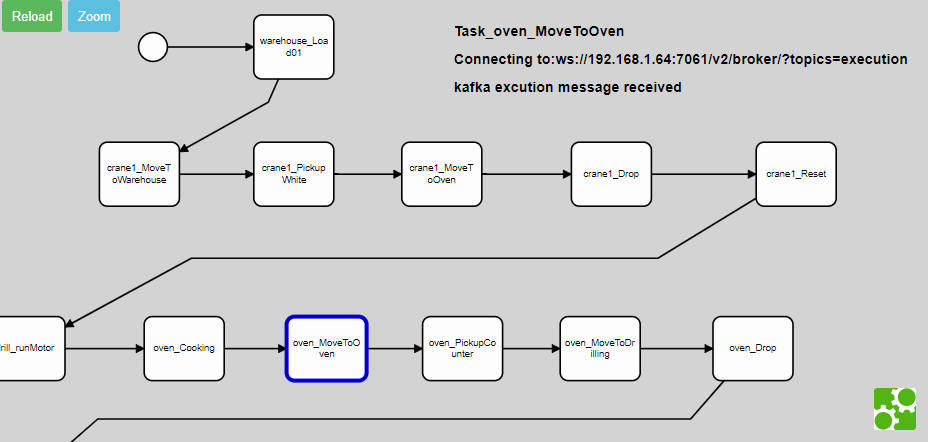
y = rollTextCoord(y, 0.1);

myObject.material.map.offset.set(x,y);

So here we move the texture for the conveyor belt to make it look like it’s moving the object across the top of it, once the value is over 1 we reset it back to 0 and when the value is under 0 we reset it to 1, this depends on if it is increasing or decreasing I.E. Moving backwards or forwards.

# BPMN Process

We created a BPMN process which has the various tasks of our factory.



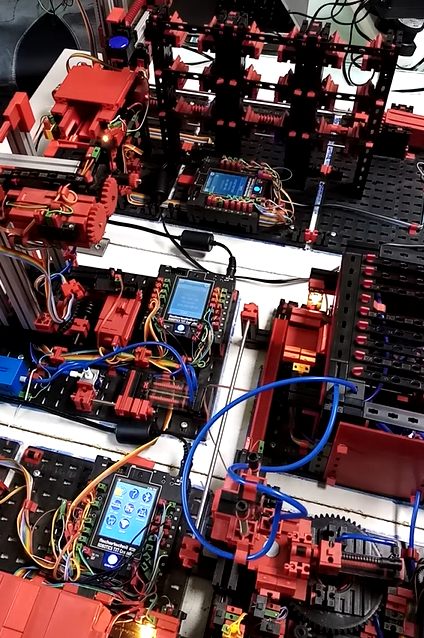
This view is from the process viewer on our live web page which tracks the current stage of the process by listening to the Kafka topic.

## Process Engine

We created a backend Process Engine which when the user clicks start from the webpage, sends the BPMN file to the Process Engine. The Engine then gets the tasks from that process and put’s them into a simple data format in order. The engine sends a Kafka message to start the first task, then waits for a done message on the same Kafka topic before moving on to the next task.

# Physical Factory

We have a physical Fischertechnik factory which we have based our model from:



<https://www.fischertechnik.de/en>

We have created a factory controller which listens to the kafka topic to receive the start event from the engine. This then triggers our factory to complete the task based on the message from kafka. Once the physical factory has completed the task, a done message is sent back on the kafka topic to the Process Engine. Which then moves on to the next task and repeats the process until out of tasks for the process. The process viewer shown previously in a screen shot also listens to this kafka topic to highlight the current task on the webpage.

## Without Physical Asset

If you didn’t have a physical model which can be controlled it’s possible to call your animate 3D model functions with buttons on the webpage. An alternative would be if you wanted to use the Process Engine and a BPMN process, a button or application could create those Kafka messages on the topic which the Process Engine is listening too.

# Analytics

We created a Grafana dashboard to see the analytics from the factory process.

## 

Another piece of software which we created is the MySQL Factory Consumer (MFC). The MFC consumes the Kafka topic as well and timestamps the messages into a MySQL database. As it reads the start and end time for each task it calculates the time taken to complete the task and inserts this into the database. It also calculates the time taken for the entire process by listening to the start and end events from Kafka.

This allows the MFC to:

* Calculate average times for each task
* Calculate Overall Equipment Efficiency (OEE) for the process
* Detect failures by monitoring the time taken to complete the task
* Display the time taken to complete the tasks in tables/graphs
* Display an alert if it took longer than usual

# Architecture

Here is an architecture diagram of our finished system.

