Iowa State University

FactBoard API Manual

Digital Manufacturing and Design Innovation Institute (DMDII)

Table of Contents

Introduction

FactBoard is a shop floor decision support system that converts thousands of existing real-time transactional data inputs from logistics and production systems into a collection of visual dashboards. These mobile decision support displays are organized around a set of personas defined according to the information needs of manufacturing management.

A key innovation of FactBoard is its ability to utilize existing transactional data within the enterprise and dynamically respond to increases, or even temporary decreases, in the quantity and quality of real-time inputs. This means companies no longer have to be in a position to make major upfront investments in shop floor data collection as FactBoard can utilize the available information effectively to ensure an increase in the quality of decision-making as additional data sources become available in the future.

Another key innovation of FactBoard is its ability to map engineering production life-cycle management (PLM) data sets with ERP-generated build schedules and real-time transactional production and logistics data to create a series of information-rich and visually effective views designed around the needs of shop floor decision makers (personas). FactBoard's decision support engine provides persona-specific calculations and probabilistic recommendations to facilitate inter-persona communications enabling effective factory-wide decision-making. The dashboard system informs decision makers of the consequences of their decisions not only for their local objective but also for global objectives at the facility level.

FactBoard fully supports pull-driven production/logistical methodologies, such as Kanban and Kitting, and provides decision traceability. The resulting information can be post-processed by an inventory reconciler engine for collaboration with ERP systems to enhance inventory and supply chain accuracy. This document describes how to get started with FactBoard and explains in-depth all the features and functionality of the current software version.

Getting Started

This document is a developing guide which will help the user develop and modify the plant layout and different views. The first section helps the user to get started with the APIs and then general information will be given on the structure of plant, dashboardview and buildlist; moreover, an example will be discussed in the last section.
The successfully created actual shop floor views will be timely updated at website: http://factboard.azurewebsites.net/

Web APIs

The Factboard application server backend application program interface (API) is based on REST, JSON and in future will support OAuth2 authentication. API's are used to define methods of communication between various software components or resources.
All the API URL's follow a common pattern as shown below.

{VERB} {base_url} /api/{controller}

Where
{VERB}: Can be "Get","Post"or"Delete", see "Postman" below for more information
{base_url}: Is the url of website.
{controller}: Indicates the controller of api used. See "Controllers of WebAPI's" below for a complete list.
For example, if I want to access EVIR controller on factboard website the URL should be:

---

http://factboard.azurewebsites.net//api/EVIR

Controllers of WebAPI's

Table 1 listed the current resources supported by Factboard.

Table 1 List of resources supported by Factboard

Name	Description
Plant	Used for maintaining the plant related information like the Assembly Lines, Stations and other layout information
DashboardView	Used for creating & editing Dashboard Views
BuildList	Used for uploading & Viewing the Build List i.e, the sequenced list of Units to be assembled on each line.
EVIR	Transactions related to Unit SignOn and SignOFF
Issue	Transactions for filing Quality & Logistics Issues
Station	Viewing and Managing station list / information

Plant, DashboardView and BuildList will be discussed later in this document, as they are all resources that can be manipulated. EVIR and Issues are generated via transactions (sign on/sign off logs, Q-notes, M@POU, etc) entered by workers.

Postman

Postman is a Google Chrome app used for interacting with HTTP API's for testing purposes. In this case, we use Postman to send information (examples: plant layout, quality issues, sign on/signoff transactions, etc.) to Factboard to simulate real time transactions expected from the client. These transactions are intended to test the current Factboard environment and identify any current bugs or development issues. Any JSON Web API compatible plugins/tools can be used for testing the web service API. In this document, Postman is used for all developing steps' demonstration.

Postman can be downloaded at https://www.getpostman.com/postman.

Request builder

Under the **Builder** tab (Figure 1), the request builder lets you create any kind of HTTP request quickly.

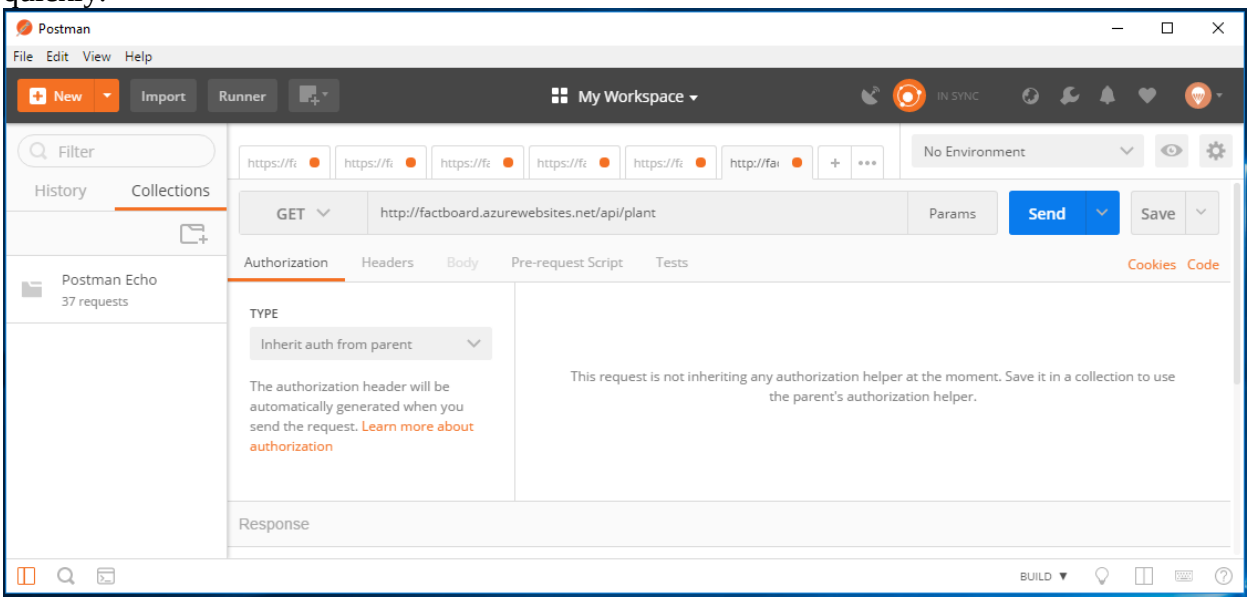


Figure 1 Request builder in Postman

The URL is the first thing that you would be setting for a request. The URL used will be determined by the type of information sent. See "Web API's" for a full list of currently supported URL's. The URL input field stores previously-used URLs and will show an autocomplete dropdown as the URL is entered.

Changing the request method is straightforward, using the control dropdown (Figure 2). **Get** and **Post** are two methods that are used here.

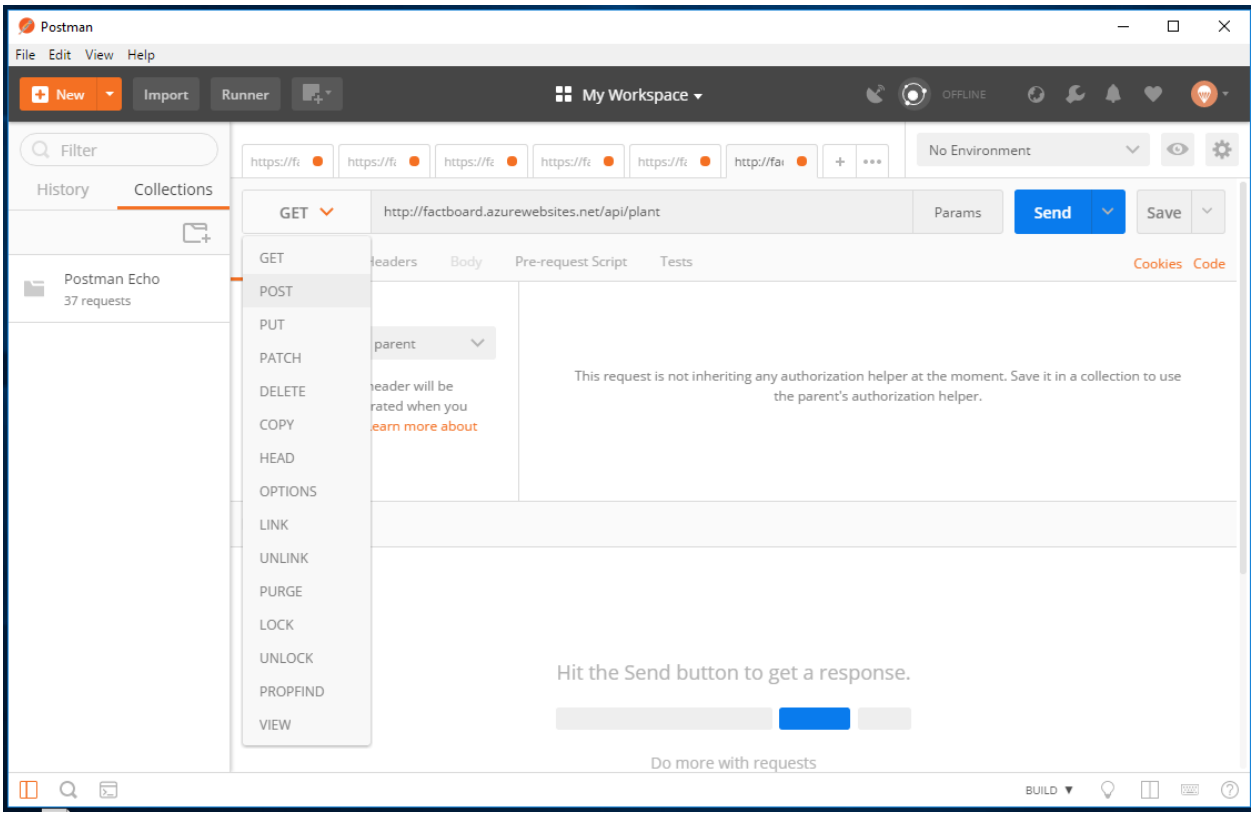


Figure 2 Methods in Postman

Get

Change the method to Get and insert the URL. Click the blue "Send" button to retrieve the information. This will return all the information/transaction logs that have been successfully received by the specified URL.

Post

Post will all a user to send information/transactions log to the desired API. Change the method to Post and insert the URL. In the Body section, choose raw and set it on JSON (application/json). Click the blue "Send" button to post the information. See Figure 3. For factboard, post formats are provided for different operations in manual.

Delete

Delete request is used to delete information , usually you will need a unique identifier to run delete command. On Factboard, Delete is only allowed on Dashboard view controller. For other controllers you can modify data by using POST request on an existing Id.

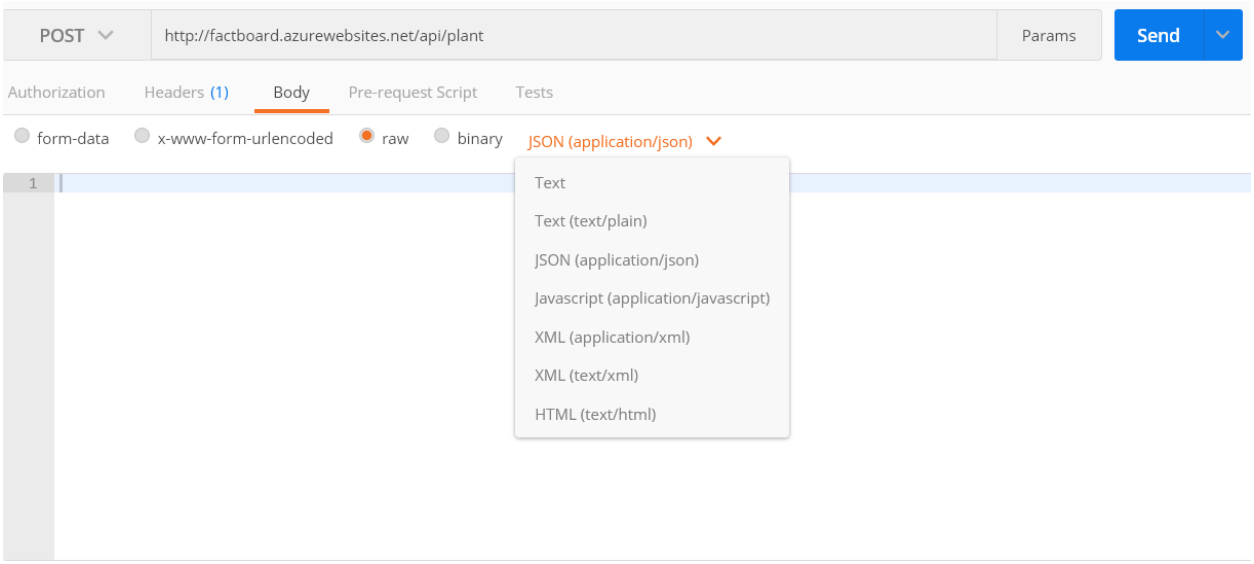


Figure 3 Posting a request

Common HTTP Status Codes

Once the information/transaction logs have been "Posted", the system displays one of the following HTTP codes. This will allow the user to determine if the transaction was successfully completed or if an error occurred. See in Table 2 below for common responses.

Table 2 Common HTTP status code

HTTP Code	Meaning
200	OK
4xx	Bad request (clients fault)
401	Unauthorized request
404	Resource not found (check the URL)
5xx	Failed request(servers fault)
500	Internal Server Error (occurs when some exception happens when processing the request)
501	Not Implemented (may occur when trying out beta APIs not fully implemented)
503	Server overloaded

HTTP Status Code Examples

The following are some examples of HTTP Status code errors

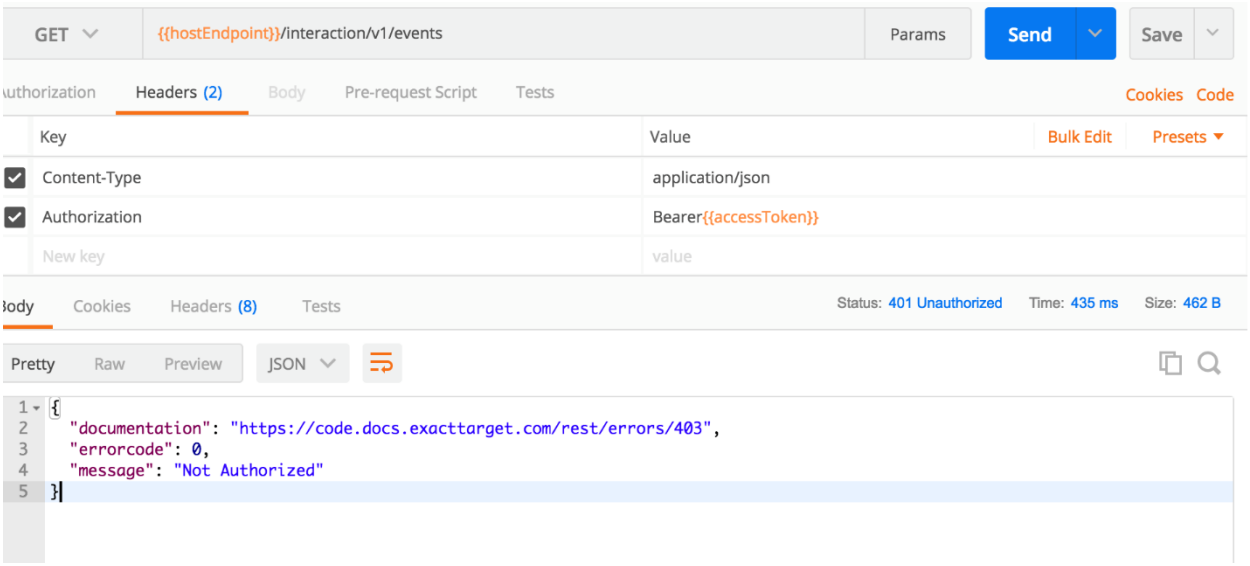


Figure 4 401 error example

401 error occurs when a request is unauthorized. This is caused by entering a URL that's not authorized, the user needs to have username and password to be able to make requests to the URL. In the current version of Factboard, an authorization is not needed.

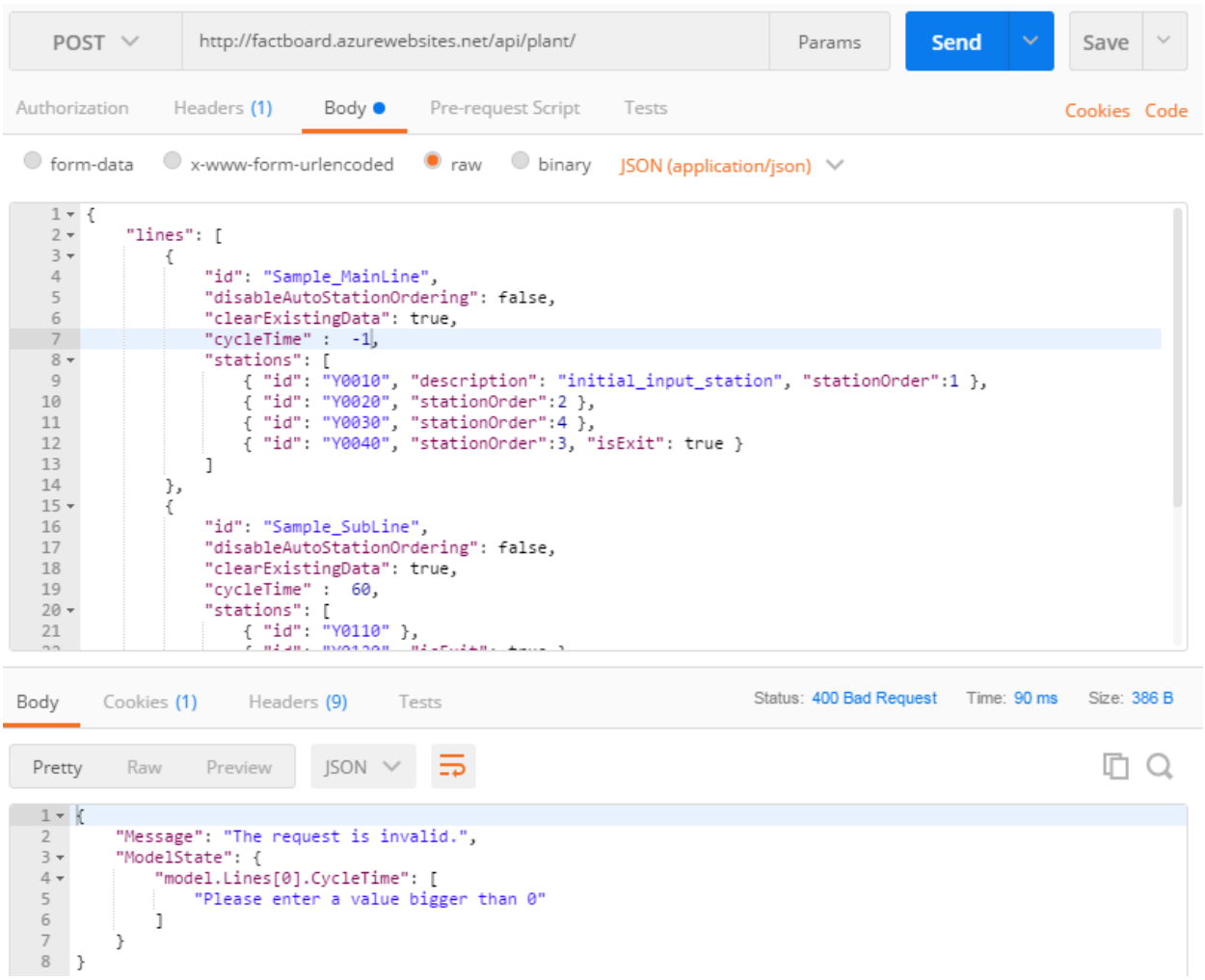


Figure 5 400 error example

400 error comes from a bad request from the client (client's fault). Corrections can be made according the error message, in this example, this is caused by entering a cycle time less than 0.

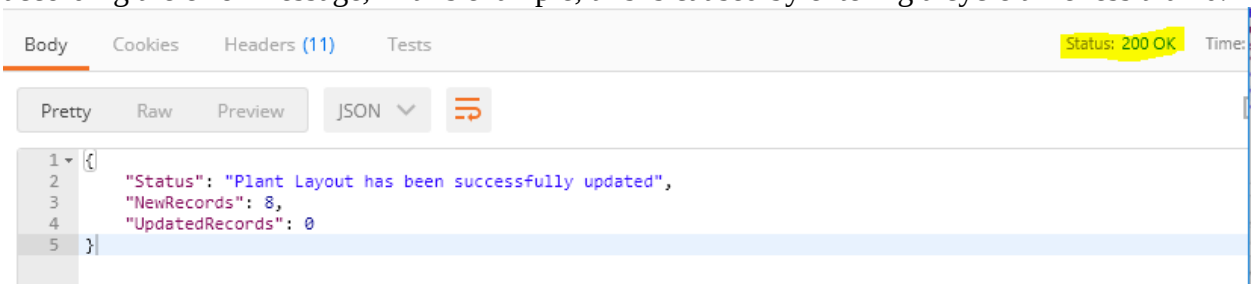


Figure 6 A success operation with 200 OK

Success message: every successful operation will get http status code "200 OK".
When you get a status "200 OK" as in Figure 6 A success operation with 200 OK, a new plant layout is successfully updated, and the number of records updated and added are also shown.
In the remaining of the document, all examples are performed with successful get and post operations with Postman.

Plant

Plant layout information

In a plant layout, all components corresponding to a shop floor layout are defined. The plant layout must be defined before a dashboard is created because it is the basis on which a dashboard view will be created. The main objects are assembly lines, including both main assembly line and any sublines supporting the mainline. Within each line, objects (mainly stations and buffers) are defined with necessary amount of information.

Get the current plant information

GET	{base_url}/api/plant

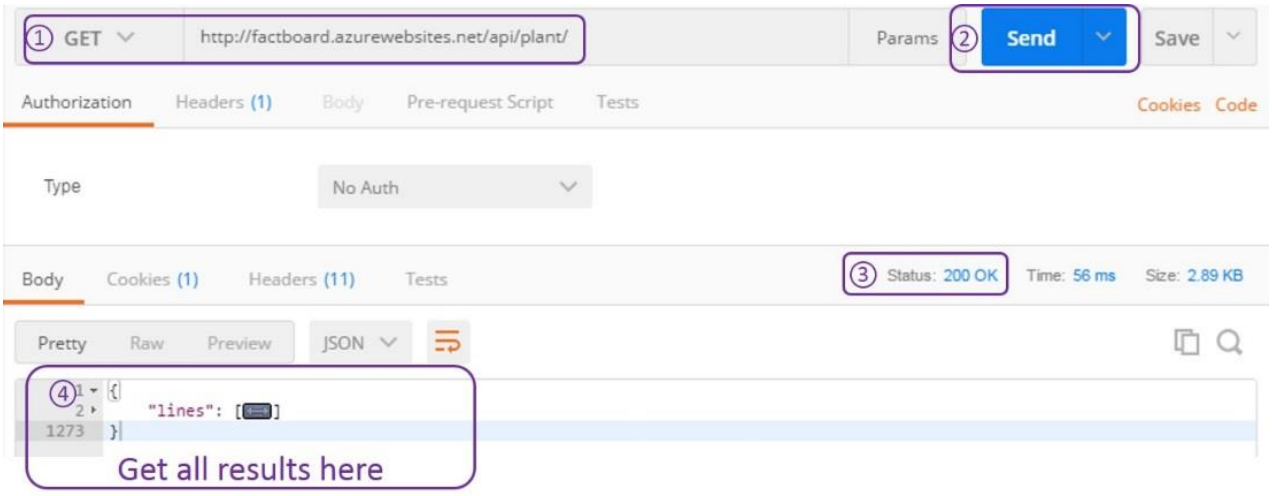


Figure 7 Steps to get the current plant layout JSON code

Follow the steps in 7 to successfully preform a GET request. With this operation, we receive all plant info with regards to all existing views.

Add or update the plant information

POST	{base_url}/api/plant

With this operation, we can add or update plant layout information with uploaded JSON script.

Plant format for post request:

```
{
  "lines": [
    {
      "id": "string",
      "clearExistingData": true,
      "disableAutoStationOrdering": true,
      "cycleTime": 0,
      "stations": [
        {
          "id": "string",
          "description": "string",
          "stationOrder": 0,
          "isExitStation": true,
          "cycleTime": 0,

        }
      ],
      "buffers": [
        {
          "id": "string",
          "description": "string",
          "fromStation": "string",
          "toStation": "string",
          "maxQty": 0,
          "minQty": 0,
          "qty": 0,

        }
      ]
    }
  ]
}
```

The format defines how the POST request should be formatted for adding/updating one line. The parameters are explained in table below. Note: At a minimum, the user must define all required parameters ("Required" = yes) for a successful transaction. All other parameters may be included but are not necessary.

Table 3 Line object

Parameter	Required	Notes
lines	YES	A collection of assembly lines. Each line is an object containing its own properties

Parameters for line object are listed in Table 4.

Table 4 Parameters for line object

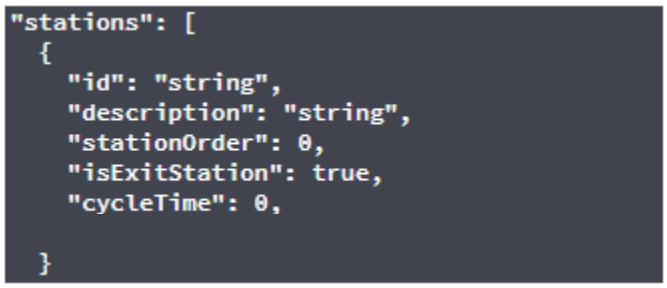
Parameter	Required	Notes
id	YES	A unique name for the assembly line.
disableAutoStationOrdering	**YES**	This will disable the automatic order assignment for stations. Default: false

clearExistingData	**YES**	This will delete all existing data for the assembly line instead of merging it, which is the default behavior.
cycleTime	NO	Specify the cycle time for the assembly line. Exceeding cycle time to a certain extend will reflects on the dashboard. The time is in minutes.
stations	**YES**	Define a list of station objects
buffers	NO	Define a list of buffers

You do not need to do a separate post for stations and buffers, they are included in plant request. They are mentioned here for explanation.

1. Station object – This is used to define individual stations within the line. Note: At a minimum, the user must define all required parameters ("Required" = yes) for a successful transaction. All other parameters may be included, but are not necessary. '

Station Layout for post request:



```
"stations": [
  {
    "id": "string",
    "description": "string",
    "stationOrder": 0,
    "isExitStation": true,
    "cycleTime": 0,

  }
```

Table 5 Parameters for station object

Parameter	Required	Notes
id	**YES**	A unique name for the station
description	NO	A short descriptive name for the station.
cycleTime	NO	An override to the cycle time specified at the line level. This is to enable specifying localized cycle times for the station. Exceeding cycle time to a certain extend will reflects on the dashboard.
isExitStation	NO	Indicate if this station is the last station on the line. Crossing this status will case the unit to be completed and will update the Line Metrics in the Andon board. Default: false.
stationOrder	NO	A number higher than 0, which indicates the order of the station. Use this if the disableAutoStationOrdering is set to False

Note:

a) **The station id should be unique**, no matter in the same or different line object. Violating this an error message "Message": "Error: Could not perform update. Validation Errors found." will appear when trying to send a POST request (HTTP Status: 400 Bad Request).
b) Each line object should have only one exit station, or {"isExit": true}.
c) stationOrder defined within station object will be used if the line object it belongs to has {"disableAutoStationOrdering": true}.

2. Buffer object – Some sub-assembly lines will allow units to build a queue between the end of the sub-assembly and the insertion into the main line. Buffer's allow users to view the units currently waiting in said queues.

Buffer Layout for post request:

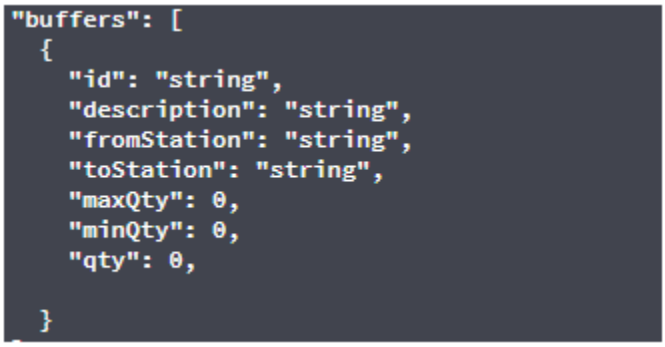


Table 6 Parameters for buffer object

Parameter	Required	Notes
id	**YES**	A unique name for the buffer station
description	NO	A short descriptive name for the buffer station.
fromStation	**YES**	The station from which the units will come into this buffer station. Usually this is the exit station of the line.
toStation	NO	Indicate if the units travel from this buffer to another downstream line. The unit count will be decremented when a unit is pulled from the buffer to the toStation.
minQty	**YES**	A number of 0 or higher, that indicates the minimum quantity of units that needs to be at the buffer at any given point. This value is used for alerts in the dashboard.
maxQty	**YES**	A number greater than 1, which indicates the maximum quantity of units that is allowed / should be at a buffer. Exceeding this value, would show alerts in the dashboard.
qty	NO	Sets the current quantity of units at a buffer. Used only when setting up an initial state. Otherwise should not be used.

Dashboard View

Dashboard View Information

A dashboard view makes use of the plant information previously uploaded to Factboard and defines the actual layout of the stations, buffers, and linkages. After a dashboard view is created or updated, it can be accessed at http://factboard.azurewebsites.net/dashboard. See user guide for further information on accessing dashboard views.

Note: The dashboard view creation or update is based on all elements needed are already existed in plant layout environment. If a plant layout is not defined than an error will occur.

Get all available dashboard views

These steps are similar to the "GET" and "POST" transactions previously described, the only difference is the API resource called. For the next example use the following URL.

GET	{base_url}/api/dashboardview/

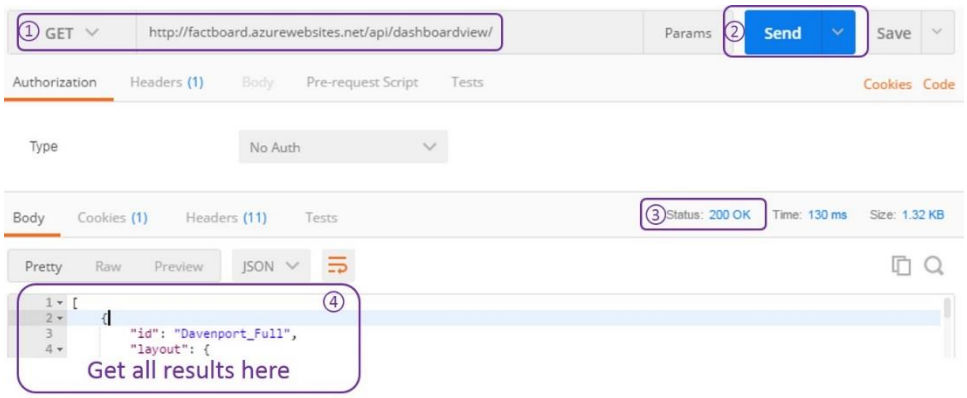


Figure 8 Steps to get the current dashboard view JSON code

Similar to GET in the previous example, using this command and the dashboard URL (factboard.azure.websites.net/api/dashboard) we receive all existing dashboard view info.

Get all dashboard view names in system

To get all view names existing in Dashboard you can send a get request to the Dashboard View controller with /getviewnames option.

GET	**{base_url}/api/DashboardView/getviewnames**

Get a specific view by ID

To get information about a specific view replace view_name by view id you want to use.

GET	**{base_url}/api/DashboardView/byViewId/{view_name}**

Add or update a view

POST	{base_url}/api/dashboardview/

Like POST in the previous example, using this command and the dashboard URL will allow the user to update or create new dashboard views with the uploaded JSON script. In this section, detailed plant layout parameter arguments are introduced.

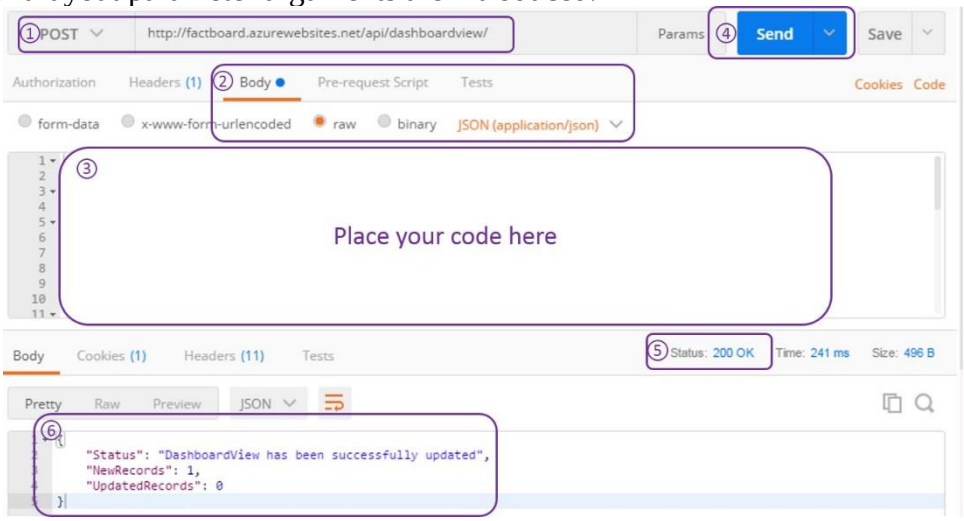


Figure 9 Steps to add/update plant layout

1. Dashboard view

To create a dashboard view, it needs a unique id, and a layout object that specifies details of the view. Note: At a minimum, the user must define all required parameters ("Required" = yes) for a successful transaction. All other parameters may be included, but are not necessary.

View Layout for post request:

```
{
  "id": "string",
  "layout": {
    "lanes": [
      "string"
    ],
    "lines": [
      {
        "id": "string",
        "lane": "string",
        "isVerticalLayout": true,
        "disableAutoStationLinking": true
      }
    ],
    "stations": [
      "string"
    ],
    "buffers": [
      "string"
    ],
    "links": [
      {
        "from": "string",
        "to": "string"
      }
    ],
    "options": {
      "showAndonBoard": true,
      "showLegend": true,
      "andonBoardLines": [
        "string"
      ],
      "useSmallNodes": true,
      "paddingTop": 0,
      "paddingBottom": 0,
      "paddingLeft": 0,
      "paddingRight": 0
    }
  }
}
```


Table 7 Dashboard elements

Parameter	Required	Notes
id	**YES**	A unique name for the dashboard. If an existing name is specified it will update that view.
Layout	**YES**	The Dashboard layout

2. Layout

Table 8 Parameters for layout

Parameter	Required	Notes
lanes	**YES**	The list of lanes, with names "L1", "L2"…, representing pieces of presenting areas on dashboard from top to bottom.
lines	**YES**	A collection of line objects that specify the id and lane on which this assembly line should be drawn. Parameters: **id**: unique id, consistent with plant layout info **lane**: lane to appear on dashboard **disableAutoStationLinking**: default true If set to false then the links section is used to define the layout. If it is true then station order is used for linking stations. **isVerticalLayout**: default false
stations	**YES**	A list of station IDs that needs to be shown in this view. Just the IDs is enough since the station information from the plant layout will be inherited.
buffers	NO	An optional list of buffer ids to use from the plant layout.
links	NO	An optional list of links to connect sub line exit station to main line stations. Parameters: from: linkage from station to: linkage to station
options	NO	See below.

Options:

Show AndOnBoard:	Toggle the view for AndOnBoard on right top corner. AndOnBoard display lines with total and completed unit count.
Show Legend	Toggle the view for Legends at bottom of screen.
AndOnBoard Lines	Provide a comma separated list of assembly lines you want to include in AndOnBoard.
Use Small Nodes	Recommended if your view contain many stations. Will use smaller nodes than usual increasing visibility.
Padding Top Padding Bottom Padding Left Padding Right	Optional padding controls.

Delete a view

To delete a view, you need to run DELETE request with an id on postman. Replace view_name with name of view you want to delete.

DELETE {base_url}/api/dashboardview?viewid=view_name

EVIR

EVIR information.

The EVIR controller is used to assign units to stations. You can use it to signon and signoff units. EVIR logs also populate the drop down list in Product structure dashboard, enabling you to see all stations in which the unit exist .

Get all the EVIR Information

To get information for all EVIR logs in the system, you can simply do get request to EVIR controller.

GET {base_url}/api/EVIR

Filter EVIR by the view Id

You can filter the results by doing a GET request with view id as shown below:

GET {base_url}/api/EVIR/byViewId/{view_id}

Filter EVIR Details for a station

You can supply a valid station id to filter EVIR information. You will also need to supply start and end date in your GET request. Replace station_id with an existing station id. The s_date and e_date tokens represent start date and end date respectively. The format for dates should be mm/dd/yyy. The end date should be greater than start date.

GET {base_url}/api/EVIR/details/{station_id}?startDateTime={s_date}&endDateTime={e_date}

Filter EVIR Details by Buffer ID

Similar to station ID you can also use a buffer ID to filter EVIR information. You will also need to supply start and end date for this request. You will also need to supply start and end date in your GET request. Replace buffer_id with an existing buffer id. The s_date and e_date tokens represent start date and end date respectively. The format for dates should be mm/dd/yyy. The end date should be greater than start date.

GET

{base_url}/api/EVIR/getbufferunits/{buffer_id}?startDateTime={s_date}&endDateTime={e_date}

Get all the Units serial numbers in a view

A view id can be used to pare down the list of EVIR logs.

GET {base_url}/api/EVIR/getevirunitlist?viewId=Default%20

Filter EVIR Information by Unit Serial Number

This request will filter EVIR information with unit serial numbers. Youcan also add view id to further narrow your search.

GET {base_url}/api/EVIR/unit/{unit_id}?viewId={view_id}

Add/ Modify EVIR

To add a EVIR record you need to do POST request to EVIR controller with appropriate data.

POST {base_url}/api/evir

EVIR layout for POST Request.

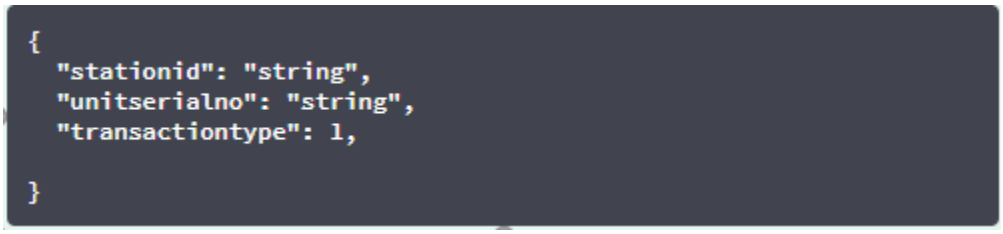


Table 8: Parameters for EVIR request

Parameter	Required	Description
Station ID	**Yes**	The station ID for which you want to add the EVIR.
Unit Serial No	**Yes**	A unique unit id used to locate unit.
Transaction type	**Yes**	

Build List

Build List Information:

Is used to upload orders and model information. The models are used to color code stations in dashboard and orders are used at sequencer.

Get all build list information:

To get information about all existing orders you can send a get request to buildlist controller.

GET {base_url} /api/buildList

Get build list information for a specific unit:

To filter information, you can use a unit serial number. Send a GET request to buildlist controller as mentioned below. Replace {unit_serial_numebr} with an existing unit serial number.

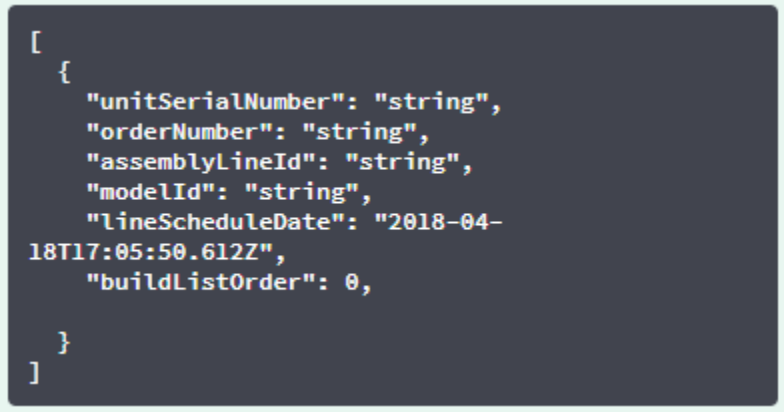
GET	{base_url} /api/buildList/{unit_serial_number}

Post a new item on build list:

You can use a POST request on buildlist controller to add or update order and model information for a existing unit.

POST	{base_url}/api/buildList

Build List Post format:



```
[
  {
    "unitSerialNumber": "string",
    "orderNumber": "string",
    "assemblyLineId": "string",
    "modelId": "string",
    "lineScheduleDate": "2018-04-
18T17:05:50.612Z",
    "buildListOrder": 0,

  }
]
```

Table 8: Build List elements

Field name	Required	Description
Unit serial Number	**Yes**	Unit serial number for which you want the order and model to be attached to. It needs to be a existing unit serial number for changes to be visible.
Order Number	**Yes**	A unique id to locate the order.
Assembly Line Id	**Yes**	The assembly line at which the unit exist.
Model Id	**Yes**	A Model ID to be associated with the unit. You can use this id later to color code stations.
Line Schedule Date	No	Optional schedule date. *It is required if you want to use AndOnBoard. This will update the Units scheduled for today in AndonBoard.*
Build List Order	No	

Issue

Issue controller is used to file Quality and Logistics issues for stations in fact board.

Get all dashboard issues

You can fetch a comprehensive list of all issues filled in the system by simply doing a GET request on Issue controller:

GET {base_url}/ api/Issue

Filter issue list

Using station id

You can modify the GET request as displayed below to get all issues for the specific stations. Replace {Station_ID} with a valid ID.

GET {base_url}/ api/Issue/details/{Station_ID}

Using Issue Type

You can further narrow down the list through Issue Type. The only supported issue types are 'Logistics' and 'Quality'. The station Id is required for this query too. The query needs to be formatted as follows:

GET {base_url}/ api/Issue/details/{Station_ID} ?issueType={Issue_type}

Replace Station Id with a valid station id and {Issue_type} by Logistics or Quality.

Using Dates

If you want to look at issues filled in a specific date range you can add a start and end date in the query as follows:

GET {base_url}/ api/Issue/details/{Station_ID} ?startDateTime={start_date} &endDateTime={end_date}

The format of dates need to be mm/dd/yyyy and end date should be greater than start date.

Add/ Update a dashboard issue

To add an issue to dashboard you will need to do a POST request. The fomat of post request needs to be of format:

```
{
  "stationid": "string",
  "unitserialnumber": "string",
  "partnumber": "string",
  "issueid": "string",
  "issuetype": "string",
  "description1": "string",
  "description2": "string",
  "severity": "string",
  "isopen": true,
  "createdby": "string"
}
```

The UID needs to be unique and is used to find a issue in database. If the same ID is used it will overwrite the old issue.
Station Id is required and

Table 9: Issue elements

Parameter	Required	Description
Stationid	Yes	Station where the issue need to be filled.
Partnumber	No	Optional part number.
Issueid	YES	A unique issue id
Issue type	Yes	Need to be one of two: “Logistics” and “Quality”
Description1	Yes	Description of issue
Description2	No	Can be provided to elaborate the issue.
Severity	Yes	Can be one of following options: “”
Isopen	Yes	Need to be true or false. Denote if issue is open or close. To close a previously opened issue, make sure to use same issue id.
Createdby	Yes	The name of person creating issue.

Inventory Reconciler

Get all assembly lines

To get all the assembly lines in the system you will need to do a get request to the Inventory Reconciler controller. The response will include a unique UID , the name of assembly line and description, if provided by the user.

GET	{base_url} /api/inventoryReconciler/assemblyLines

Get all storage areas

To get a list of all storage areas associated with an assembly line you can execute a get request inventory reconciler controller. You need to specify a UID (Not id) in request. Replace {assembly_line_uid} with the UID.

GET	{base_url}/api/inventoryReconciler/storageAreas/{assembly_line_uid}

Sequencer

Though Most of the tasks in sequencer can be accomplished by user interface in Sequencer , you can do some postman requests to Factboard API to get more filtered data.
Sequencer has following controllers:

Controller name	**Description**

Sequence	To get information regarding views and product groups.
Sequencer	To post new information or update old records related to orders.
Sequence Import	Can be used to modify more higher-level information including assembly lines, time slots and storage areas, etc.

Sequence

Get all the sequencer product groups

No parameters needed.
You can quickly get a list of all product groups in sequencer by doing a get request to sequence controller. The response will include UID, Name and description for groups.

GET	{base_url} /sequence/productGroups

Get all the views within a specific sequencer product group

Parameters needed: Product group uid.
Product group uid should be a valid id and exist in database. Please not that it is uid and is an integer not the text id.

GET	{base_url} /api/sequence/viewInfos/{view_uid}

Get sequence view information for a specific view filtered by dates

Parameters needed: Sequence view id, start date and end date.
Sequence view id should be valid id. Start date and end date should be in format yyyy/mm/dd and end date should be greater than start date. The request can be formatted as follows:

GET	{base_url} /api /sequence/view/{sequence_view_id}? startDate={start_date}&endDate={end_date}

Sequencer

Assign orders

To post assign orders to a view in sequencer you can do a post to Sequencer controller with assign orders. The request also returns what items were affected by the change.

POST	{base_url}/api/Sequencer/assignOrders

The post request should be like following:

```
{
  "viewUID": 0,
  "startDate": "2018-04-25T20:06:14.158Z",
  "endDate": "2018-04-25T20:06:14.158Z",
```

```
  "orderUIDList": [
    0
  ],
  "slotUIDList": [
    0
  ]
}
```

Un-assign order:

To un-assign orders within a specific view do a post request to Sequencer controller. The request will return all items that were affected by the change.

POST	{base_url}/ api/Sequencer/unassignOrders

The post request should be formatted as follows:

```
{
  "viewUID": 0,
  "startDate": "2018-04-25T20:36:38.542Z",
  "endDate": "2018-04-25T20:36:38.542Z",
  "orderUIDList": [
    0
  ]
}
```

Swap Slots:

You can also swap orders using POST request. The request should be on Sequener controller.

POST	{base_url}/ api/Sequencer/unassignOrders

You also need to know the slotUIDs for this post. The post should be formatted as follows:

```
{
  "viewUID": 0,
  "startDate": "2018-04-25T23:20:42.914Z",
  "endDate": "2018-04-25T23:20:42.914Z",
  "slotUID_1": 0,
  "slotUID_2": 0
}
```

Tutorial with Example

Example Plant layout:

First, we will start by adding our assembly lines in our plant. These lines will define stations and buffers with required attributes. We will add three lines named: Main Line, PowerMod and SUB1. For his we will do three POST requests to plant controller adding one line at a time. Note: **Station Ids has to be unique**, if you are using this example to follow modify the ids.

Post request for Main Line


```
{
 "lines": [
   {
                    "id": "Main Line",
                   "clearExistingData":"true",
                   "disableAutoStationOrdering":"true",
                   "cycletime":"100",
                   "stations":[
                              {
              "id": "DM_S1",
              "description": "",
              "stationOrder": 1,
              "isExitStation": false,
              "cycleTime": 60,
              "modifiedOn": "2017-08-04T23:50:14Z"
            },
            {
              "id": "DM_S2",
              "description": "Station 2",
              "stationOrder": 2,
              "isExitStation": false,
              "cycleTime": 60,
              "modifiedOn": "2017-08-04T23:50:14Z"
            },
            {
              "id": "DM_S3",
              "description": "Station 3",
              "stationOrder": 3,
              "isExitStation": false,
              "cycleTime": 30,
              "modifiedOn": "2017-08-04T23:50:14Z"
            },
            {
              "id": "DM_S4",
              "description": "Station 4",
              "stationOrder": 4,
              "isExitStation": false,
              "cycleTime": 60,
              "modifiedOn": "2017-08-04T23:50:14Z"
```

```
},
{
  "id": "DM_S5",
  "description": "Station 5",
  "stationOrder": 5,
  "isExitStation": true,
  "cycleTime": 60,
  "modifiedOn": "2017-08-04T23:50:14Z"
},
{
  "id": "DM_S6",
  "description": "Station6",
  "stationOrder": 6,
  "isExitStation": false,
  "cycleTime": 60,
  "modifiedOn": "2017-08-04T23:50:14Z"
},
{
  "id": "DM_S7",
  "description": "Station7",
  "stationOrder": 7,
  "isExitStation": false,
  "cycleTime": 60,
  "modifiedOn": "2017-08-04T23:50:14Z"
},
{
  "id": "DM_S8",
  "description": "Station8",
  "stationOrder": 8,
  "isExitStation": false,
  "cycleTime": 60,
  "modifiedOn": "2017-08-04T23:50:14Z"
},
{
  "id": "DM_S9",
  "description": "Station9",
  "stationOrder": 9,
  "isExitStation": false,
  "cycleTime": 60,
  "modifiedOn": "2017-08-04T23:50:14Z"
},
{
  "id": "DM_S10",
  "description": "Station10",
  "stationOrder": 10,
  "isExitStation": false,
```

```
    "cycleTime": 60,
    "modifiedOn": "2017-08-04T23:50:14Z"
  },
  {
    "id": "DM_S11",
    "description": "Station11",
    "stationOrder": 11,
    "isExitStation": false,
    "cycleTime": 60,
    "modifiedOn": "2017-08-04T23:50:14Z"
  },
  {
    "id": "DM_S12",
    "description": "Station12",
    "stationOrder": 12,
    "isExitStation": false,
    "cycleTime": 60,
    "modifiedOn": "2017-08-04T23:50:14Z"
  },
  {
    "id": "DM_S13",
    "description": "Station13",
    "stationOrder": 13,
    "isExitStation": false,
    "cycleTime": 60,
    "modifiedOn": "2017-08-04T23:50:14Z"
  },
  {
    "id": "DM_S14",
    "description": "Station14",
    "stationOrder": 14,
    "isExitStation": true,
    "cycleTime": 60,
    "modifiedOn": "2017-08-04T23:50:14Z"
  }
                ],
      "buffers":[
                {
    "id": "DM_B2",
    "description": "BUFFER2",
    "fromStation": "ML140",
    "toStation": "",
    "maxQty": 10,
    "minQty": 0,
    "qty": 0,
    "modifiedOn": "2017-08-04T23:50:14Z"
```

```
            }
                        ]
                }
                ]
}
```

POST for PowerMod

```
{
  "lines": [
    {
     "id":" PowerMod",
     "clearExistingData":"true",
     "disableAutoStationOrdering":"true",
     "cycletime":"80",
          "stations": [
             {
                "id": "D50Z010EN",
                "description": "Engine",
                "stationOrder": 1,
                "isExitStation": false,
                "cycleTime": 60
             },
             {
                "id": "D50Z010PM",
                "description": "Power Mod1",
                "stationOrder": 2,
                "isExitStation": false,
                "cycleTime": 60
             },
             {
                "id": "D50Z020PM",
                "description": "Power Mod2",
                "stationOrder": 3,
                "isExitStation": false,
                "cycleTime": 20
             },
             {
                "id": "D50Z030PM",
                "description": "Power Mod3",
                "stationOrder": 4,
                "isExitStation": true,
                "cycleTime": 60
             },
             {
                "id": "D50Z020EN",
                "description": "Heat Exchange",
                "stationOrder": 5,
```

```
                "isExitStation": false,
                "cycleTime": 60
            }
        ],
        "buffers": [
            {
                "id": "DM_B1",
                "description": "BUFFER1",
                "fromStation": "50Z030PM",
                "toStation": "ML090",
                "maxQty": 2,
                "minQty": 0,
                "qty": 0
            }
        ]
    }]
}
```

Post for SUB1

```
{
  "lines": [
    {
      "id": "SUB1",
      "clearExistingData":"true",
                    "disableAutoStationOrdering":"true",
                    "cycletime":"90",
          "stations": [
            {
                "id": "SD1",
                "description": "Station 1",
                "stationOrder": 1,
                "isExitStation": false,
                "cycleTime": 80
            },
            {
                "id": "SD2",
                "description": "Station 2",
                "stationOrder": 2,
                "isExitStation": false,
                "cycleTime": 80
            },
            {
                "id": "SD3",
                "description": "Station 3",
                "stationOrder": 3,
                "isExitStation": true,
```

```
                "cycleTime": 80
            }
        ],
        "buffers": [
            {
                "id": "DM_B3",
                "description": "BUFFER3",
                "fromStation": "S3",
                "toStation": "ML070",
                "maxQty": 10,
                "minQty": 0,
                "qty": 0
            }
        ]
    }]
}
```

Example View:

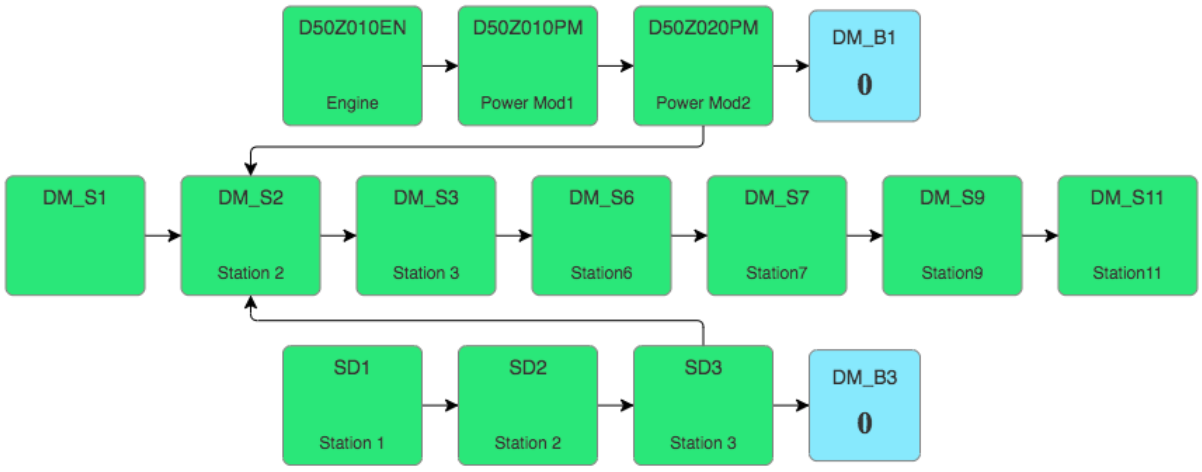
Now when we have defined our lines we need to define the view. In view you define which stations you want to see and how they should be linked. Here is a post request data for example view:

POST request for View:

```
{
  "id": "Example",
  "layout": {
        "lanes": [
            "LD1",
            "LD2",
            "LD3"
        ],
        "lines": [
            {
                "id": "Main Line",
                "lane": "LD2",
                "isVerticalLayout": false,
                "disableAutoStationLinking": false
            },
            {
                "id": "PowerMod",
                "lane": "LD1",
                "isVerticalLayout": false,
                "disableAutoStationLinking": false
            },
            {
                "id": "SUB1",
                "lane": "LD3",
                "isVerticalLayout": false,
```

```
            "disableAutoStationLinking": false
         }
      ],
      "stations": [
         "DM_S1",
         "DM_S2",
         "DM_S11",
         "DM_S9",
         "DM_S6",
         "DM_S7",
         "DM_S3",
         "D50Z010EN",
         "D50Z010PM",
         "D50Z020PM",
         "SD1",
         "SD2",
         "SD3"
      ],
      "buffers": [
         "DM_B1",
         "DM_B3"
      ],
      "links": [
         {
            "from": "D50Z020PM",
            "to": "DM_S2"
         },
         {
            "from": "SD3",
            "to": "DM_S2"
         }
      ],
      "options": {
         "showAndonBoard": true,
         "showLegend": true,
         "andonBoardLines": [
            "Main line"
         ]
      }
   }
}
```


At this point, you can go to your website and look at the view. The provided example will look like layout provided. You need to select your dashboard view with dashboard type as current and color code type: Logistic issues.

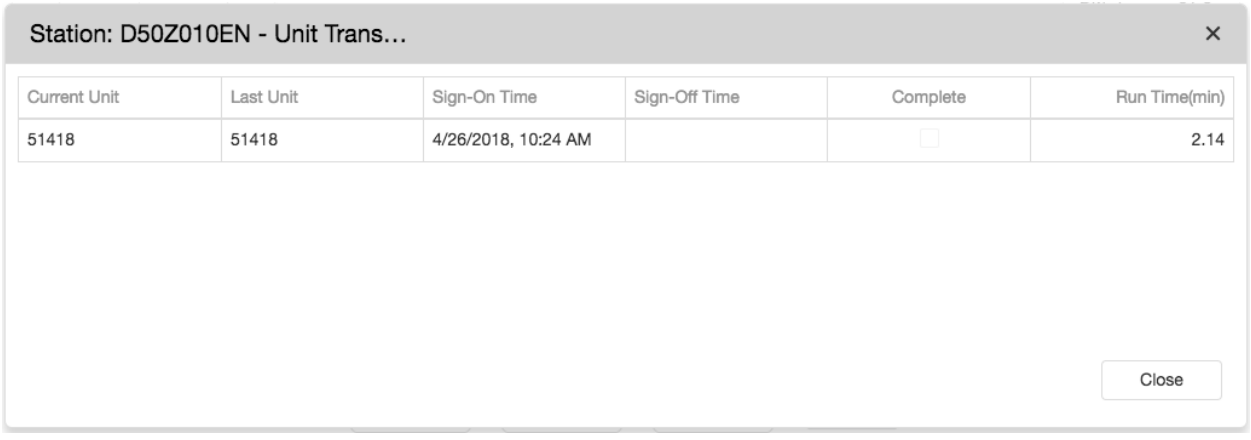


Example EVIR log:

The stations in the example are color coded according to station status. Station status depends on the units assigned to station and how much time they take to execute. If the execution time is more than selected upper limits, then stations are color coded as red. To attach a unit to station ou need to do post request to EVIR controller.

For example, if we want to add a unit with unit serial number as D51418 to station D50Z010EN the Post should look like below. After you successfully add the unit you can go back to website, refresh it and double click on the station to verify the unit is displayed in unit transactions.

```
 {
  "stationid": "D50Z010EN",
  "unitserialno": "D51418",
  "transactiontype": "signon"
}
```



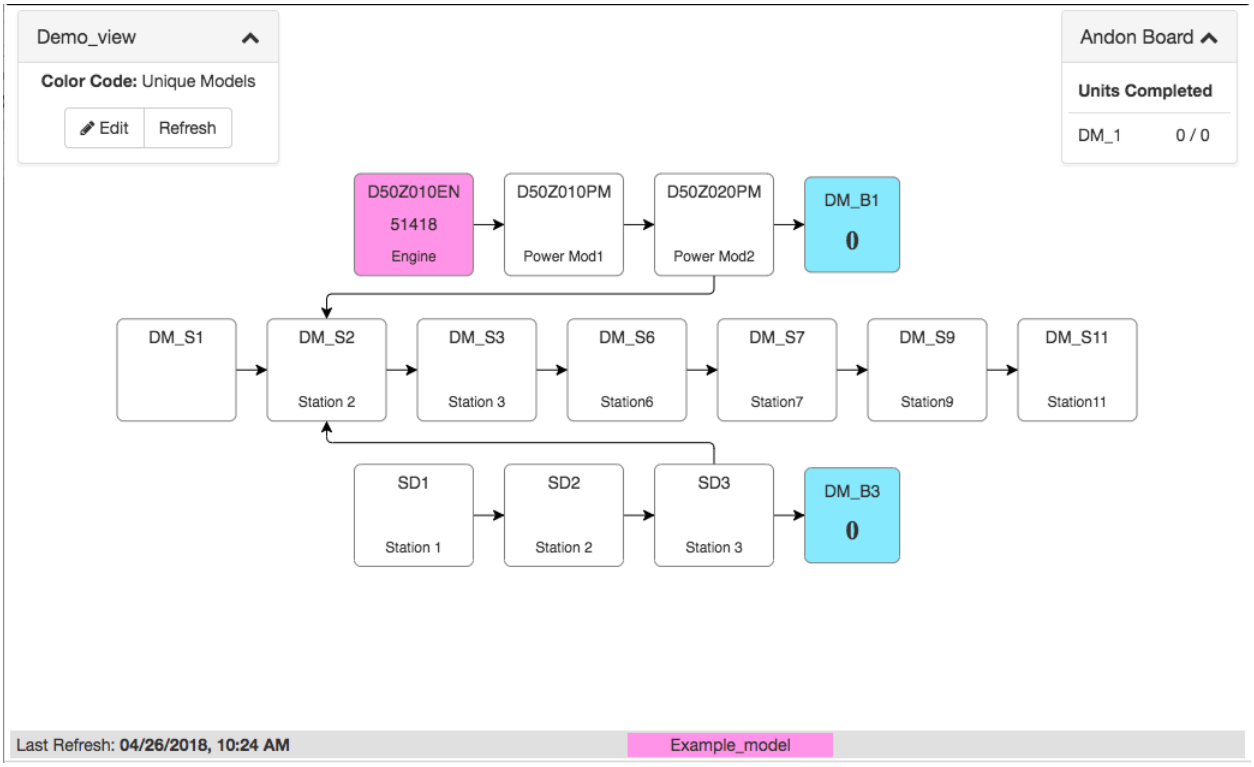
Example Build List:

Now when we have successfully added a unit, we can add order and model. To do this post data will look like:

```
[
  {
```

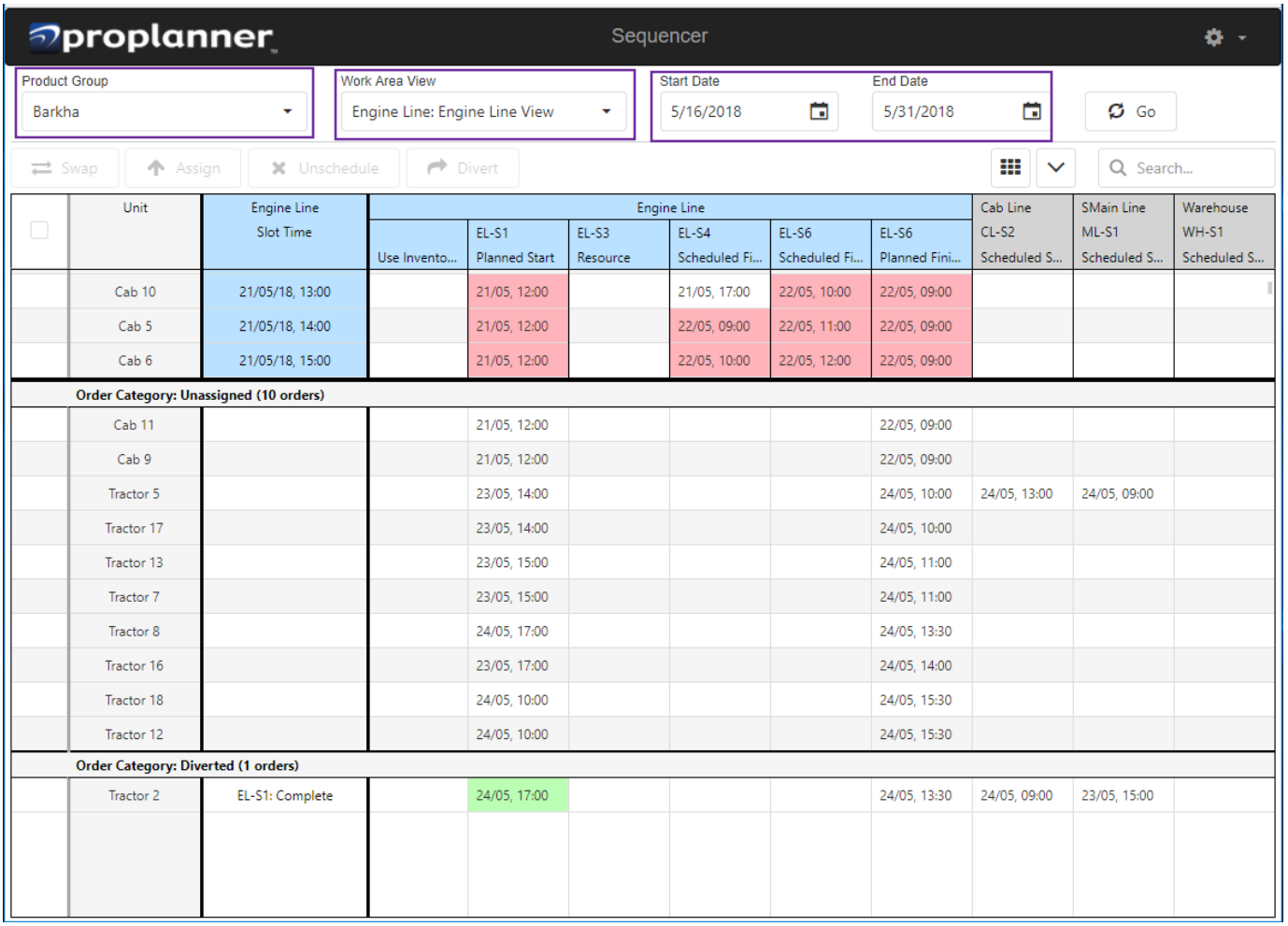
```
  "unitSerialNumber": "D51418",
  "orderNumber": "D1555",
  "assemblyLineId": "Main Line",
  "modelId": "Demo_model"
 }
]
```

To see your newly added model you can go back to website and select color code type as model. In the example, you will be able to see that Demo_Model has been associated station D50Z010EN because it contains unit D51418.



Example Sequencer File:

For Sequencer and Inventory Reconciler the data can be imported by uploading an excel file. This file is called a sequencer file and have number of sheets that we will discuss in detail. A typical sequencer tab looks as shown below:



Note that sections may vary depending on your view selected for the grid and orders. You can select to show or hide different panes in your grid options. The pane will only show up if you have a relevant order. For example, if you don't have any diverted order you will not see Order Category: Diverted even if you selected it in the grid option.