ezzing3DLayout

PV planning tool

API Documentation

ezzingsolar®

16 of August, 2016

Table of Contents

Introduction

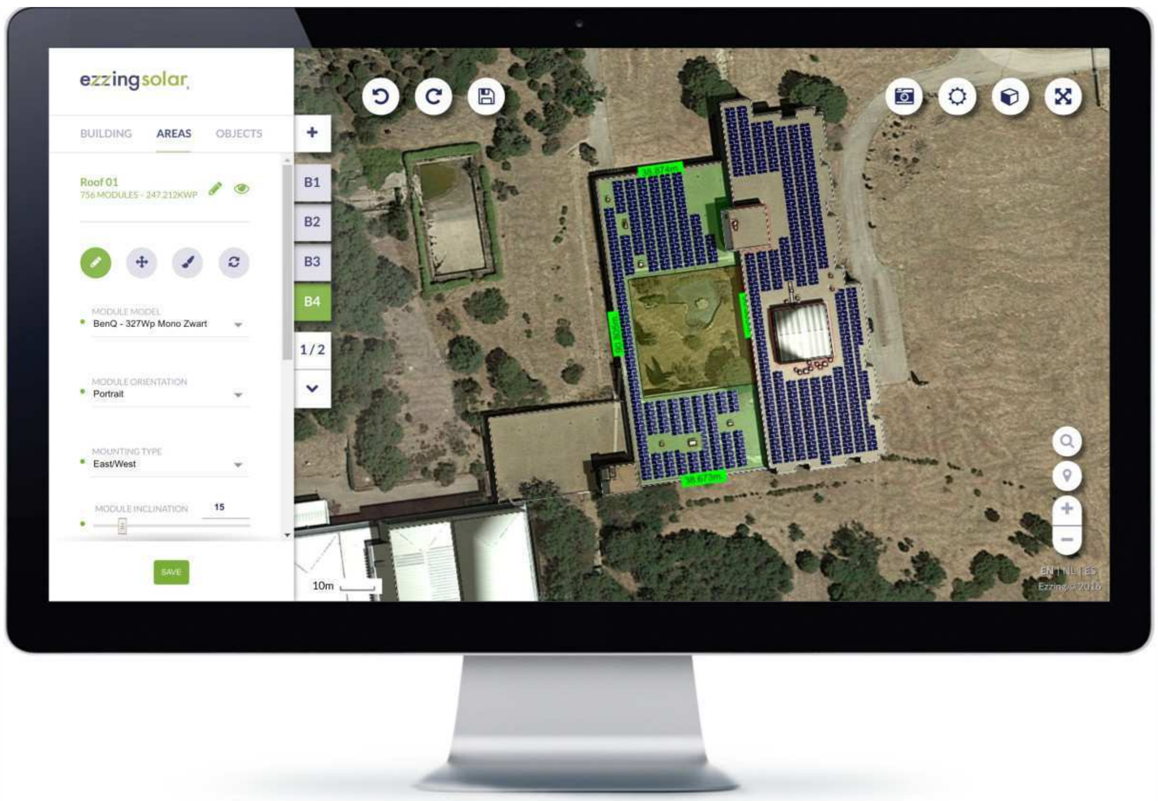
Ezzing 3DLayout is a PV planning tool that allows you to generate a 3d model of a building based on a satellite image. You can model any number of buildings, select between up to five different type of roofs, define keepouts and trees with custom heights...

Inside each roof area you can customize different structures, select module models and get automated previews of your setup.

It also provides you with a perspective view and a sun simulator to determine where the shadows will be in your installation.

Ezzing 3DLayout is an embebible webapp. You can integrate it inside your own system and customize many elements inside, from module models to prefered default settings for each roof type.

In this document you will find a brief showcase of the different areas of the app, a technical explanation on how to integrate this webapp inside your platform, a full description of the API that will allow you to communicate with the 3DLayout, and finally a description on how to customize different parts of the app.



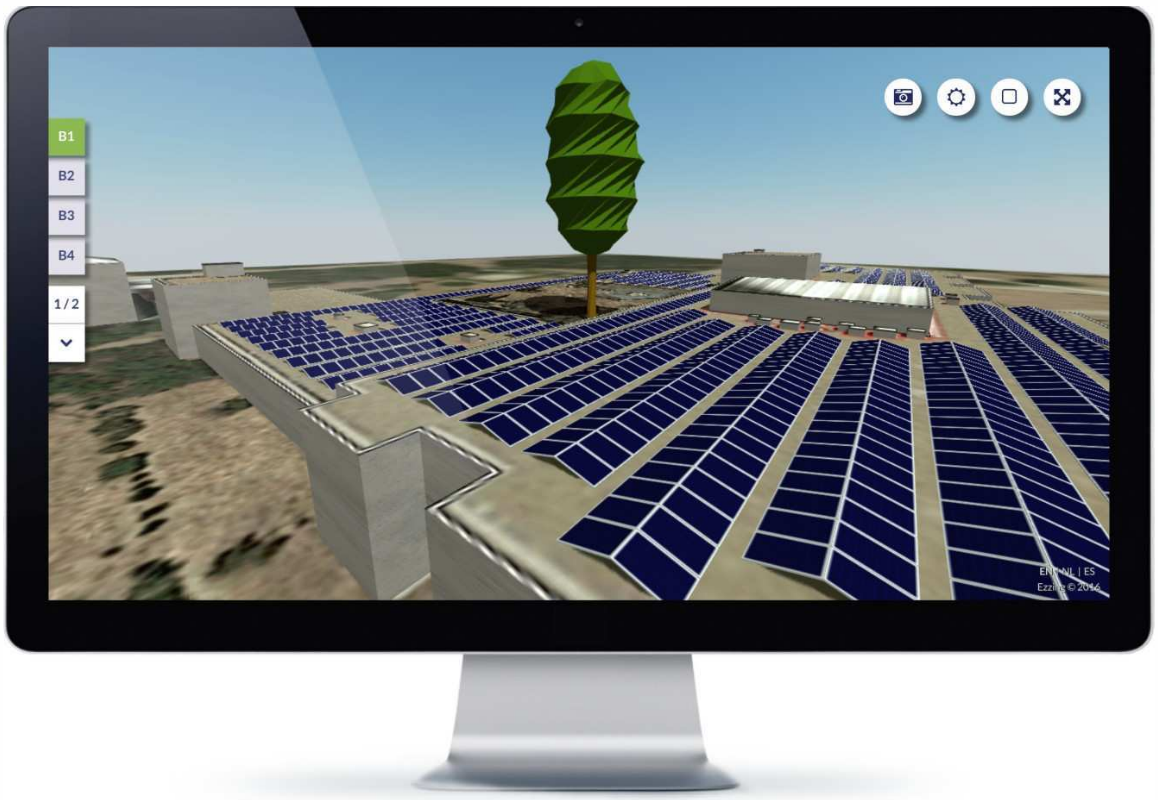
You can test the app by visiting this link:

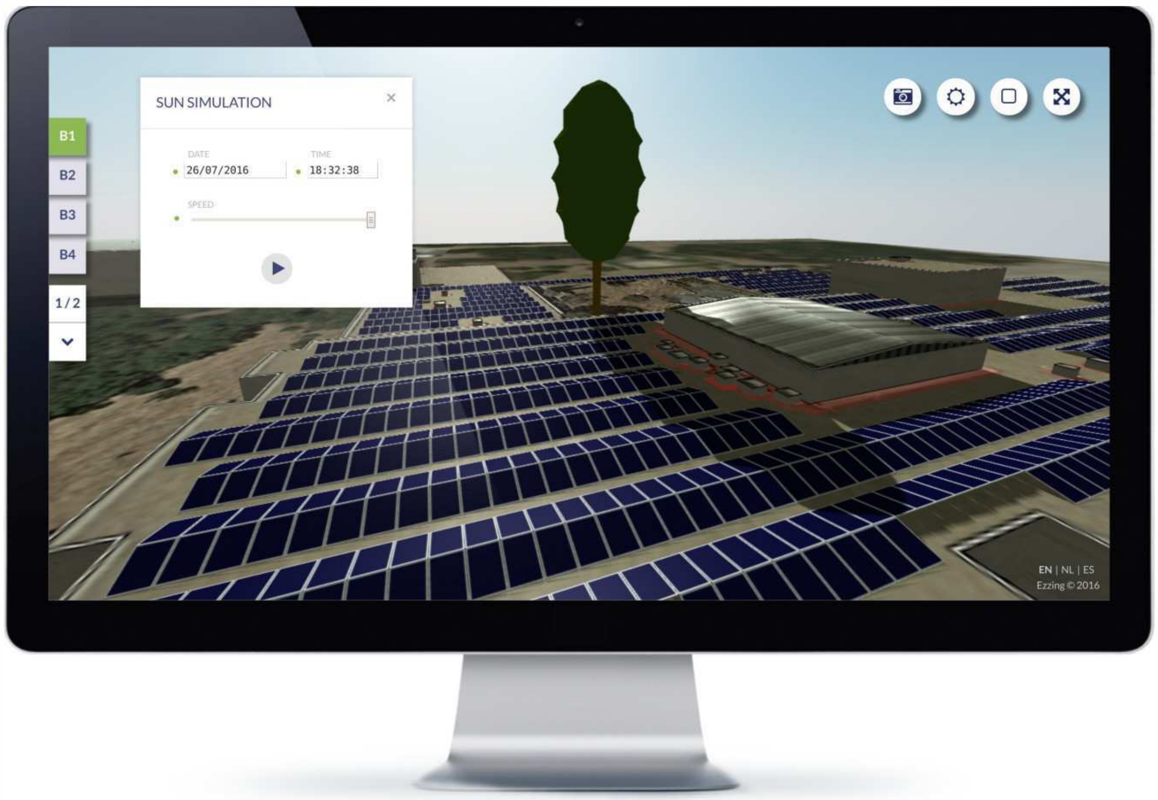
https://layout.ezzing.com/#/demo

Also you can follow a tutorial to learn the basics of the 3DLayout in this link:

https://layout.ezzing.com/#/tutorial

Screenshots



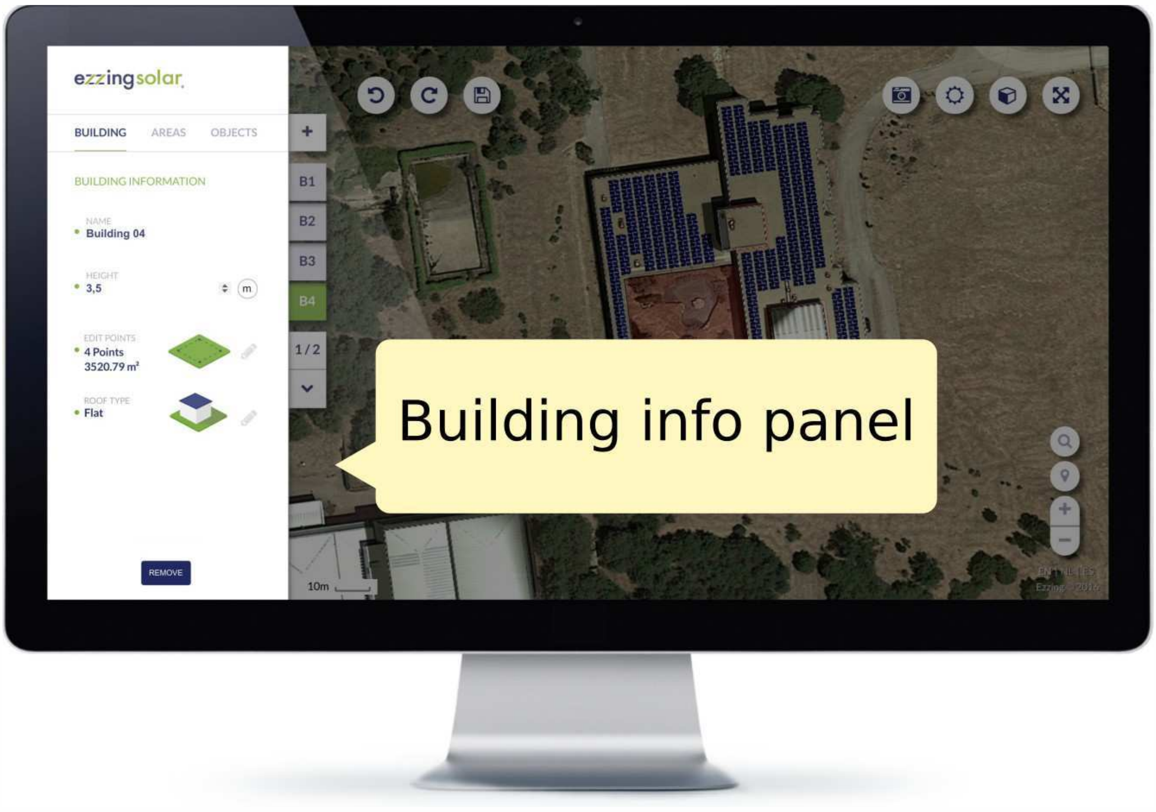


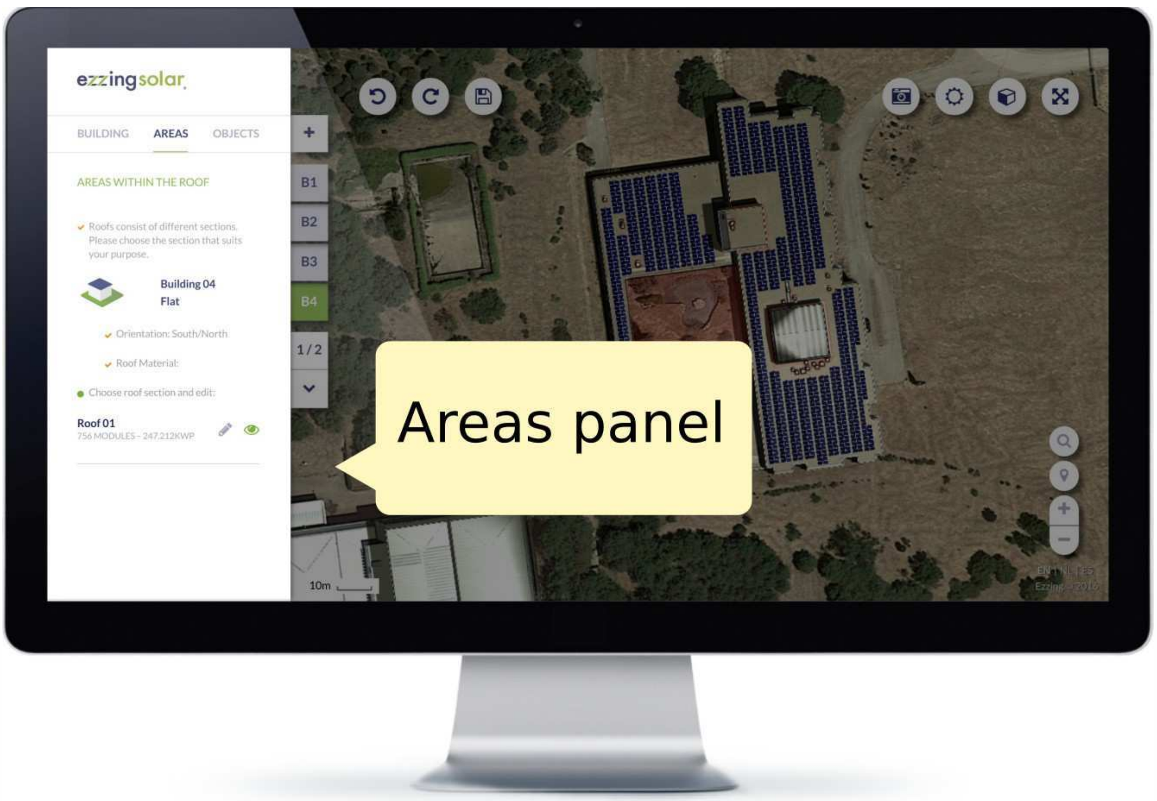
3DLayout Interface

The 3DLayout interface has two diferent parts. The **aside panel** and the **canvas area**.

Aside Panel

In the aside panel you can find functionalities related to the current active building and other objects in the scene.





AREAS WITHIN THE ROOF

Roofs consist of different sections. Please choose the section that suits your purpose.

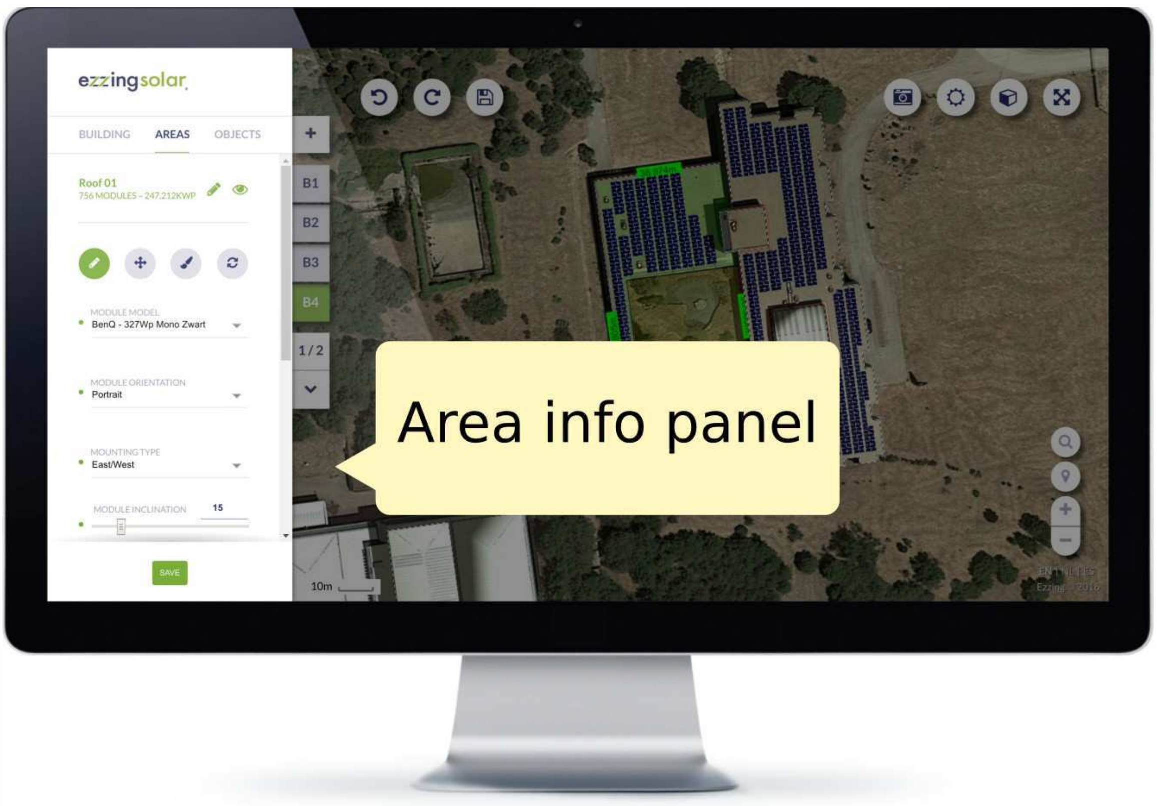
Building 04
Flat

Orientation: South/North

Roof Material:

Choose roof section and edit:

Roof 01
756 MODULES – 247.212KWP



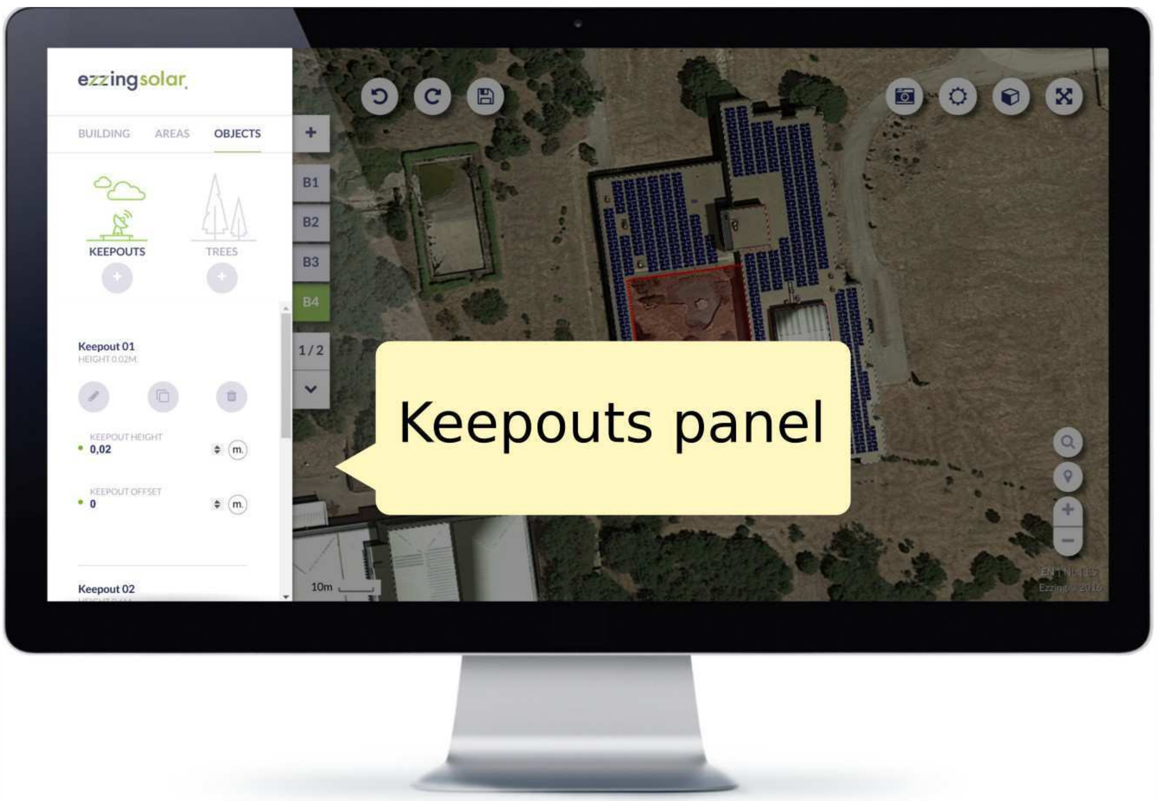
Roof 01
756 MODULES – 247.212KWP

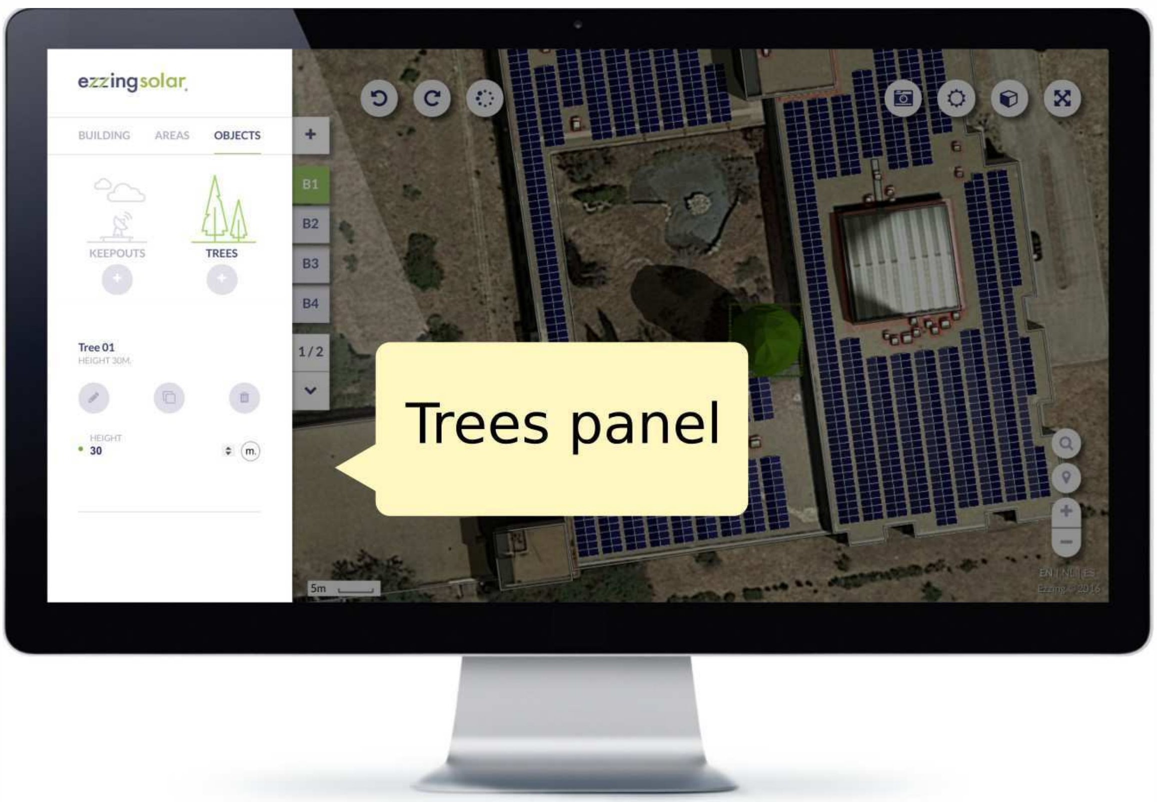
MODULE MODEL: BenQ - 327Wp Mono Zwart

MODULE ORIENTATION: Portrait

MOUNTING TYPE: East/West

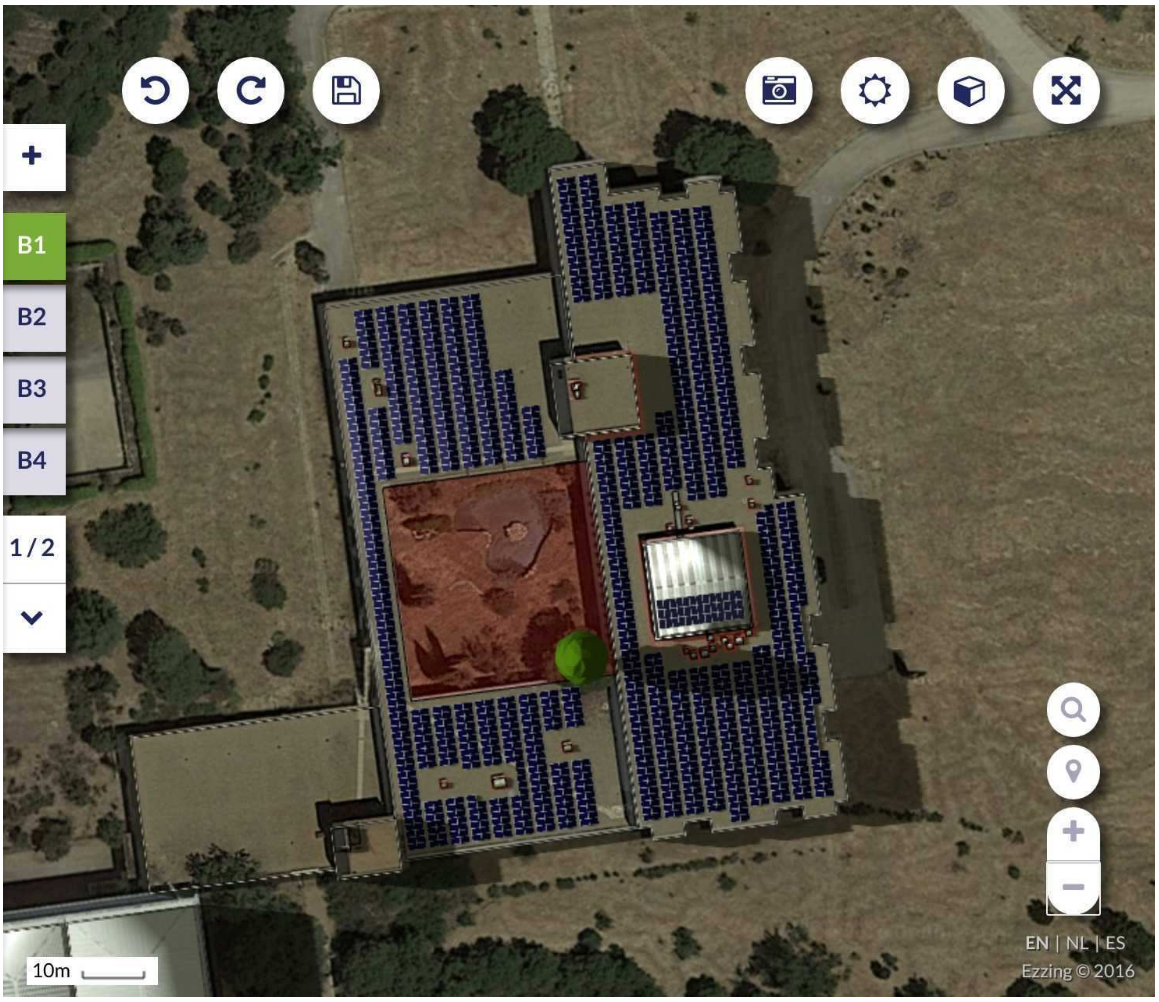
MODULE INCLINATION: 15


ezzingsolar
BUILDING
AREAS
OBJECTS
KEEPOUTS
TREES
B1
B2
B3
B4
1 / 2
Keepout 01
HEIGHT 0.02M
KEEPOUT HEIGHT
0,02
m.
KEEPOUT OFFSET
0
m.
Keepout 02
10m
Keepouts panel


ezzingsolar
BUILDING
AREAS
OBJECTS
KEEPOUTS
TREES
B1
B2
B3
B4
1 / 2
Tree 01
HEIGHT 30M
HEIGHT
30
m.
5m
Trees panel

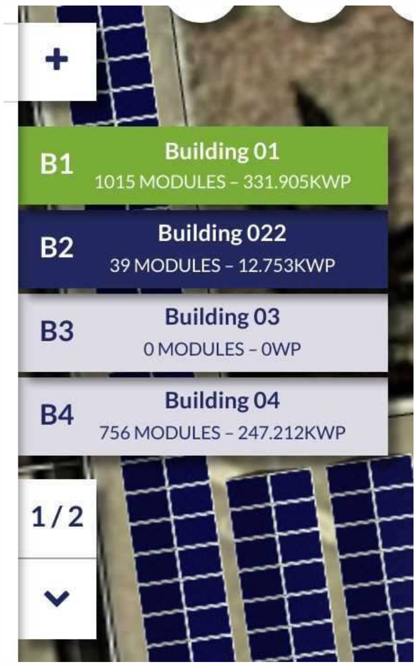
Canvas Area

In the canvas area you can see the satellite view and three diferent sets of elements, the **buildings index** on the left side, the **main options buttons** on the top, and the **control buttons** on the bottom-right corner.



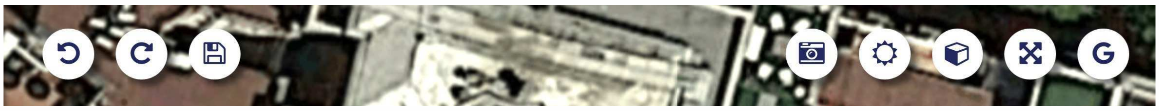
Buildings Index

In the buildings index you can see the active building and select another one to become active.



Main Options

These are the main options in the canvas area.



The three left-sided buttons are the **main options fixed buttons**:

- redo
- undo
- save

The right aligned buttons are the **main options custom buttons**. You can customize this set of buttons by hidding some of them or by adding new buttons.

The default custom buttons are:

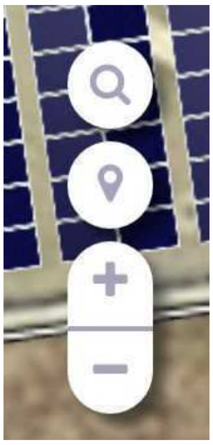
- snapshot
- sun simulation

- perspective view
- fullscreen
- satellite provider selector (only showed if available)

> Please, visit the section Custom Buttons to learn how to add your own functions.

Control Buttons

These are map related buttons. You can also customize the upper section of this set of buttons by hidding some of them or by adding new buttons



Fixed buttons in this area:

- zoom in
- zoom out

Default control custom buttons

- search address
- geolocation

> Please, visit the section Custom Buttons to learn how to add your own functions.

Integration In A Non-Angular Platform

To integrate the 3DLayout into your platform, you only need this five files

- angular_lib.js - include the angular library
- ezzing3DApp.js - Integration tools (contains the ezzing3D constructor and API functions)
- ezzing3D.js - contains the platform code.
- lib.js - some other useful third party libraries
- screen.css - contains the 3DLayout css styles

Copy this files into the desired path (i.e: public/ezzing3d/)

HTML Code

To embed the 3DLayout app into an html page, you only need to add some lines to the code.

DOM Header

Add this lines to the header element in your code:

```
<link rel="stylesheet" type="text/css" href="/path-to-ezzing3d-files/screen.css">
```

DOM Body

You need an element in the body of your html page, a div with id="layout" where the 3DLayout will fit inside.

> This div can not be bigger than the view size, and **no scroll** has to be applied to the html page.

You should not change the ezzing3d element size (width or height) but change the size of this container div.

Add this lines inside the body element in your code:

```
<div id="layout" style='height:100vh; width: 100vw'>
    <ezzing3d id='ezzing3d' layoutRules='{"logo": false}'></ezzing3d>
</div>
```

and these lines at the end of the body:

```
<script type="text/javascript" src="/path-to-ezzing3d-files/angular_lib.js"></script>
<script type="text/javascript" src="/path-to-ezzing3d-files/ezzing3DApp.js"></script>
<script type="text/javascript" src="/path-to-ezzing3d-files/ezzing3D.js"></script>
<script type="text/javascript" src="/path-to-ezzing3d-files/lib.js"></script>
```

Javascript Code

To instantiate the ezzing3D object and start the app you need to run this lines inside your javascript code:

```
var container = window.document.getElementById('ezzing3d');

var layout = new ezzing3D();

layout.bootstrap(container, {
    token: token,
    id: id,
    failURL: '#/fail',
    showcase: showcase
}, rules);
```

The ezzing3D contructor instance the object layout, in this object we have all the functionality to manage the 3DLayout

The bootstrap function init the 3DLayout in the defined container with the provided options.

The options for this function are:

- **container**: the DOM element with id='ezzing3d'
- **token**: the user token from the CRM

- **id**: the CRM project id. This value can also be used to activate special modes in the 3DLayout. Use id="demo" to run a 3DLayout demo with an empty project. Use id="tutorial" to run an interactive tutorial. Use id="landing" to view an special showcase demo project.
- **failURL**: an alternative URL to redirect the user when token or id from CRM fails.
- **showcase**: When this value is true the project is opened in a special showcase mode, without graphic interface and in a nice 3d view. This mode is read-only and the project can not be modified.
- **rules**: This is the custom rules to customize the 3DLayout, this attribute is optional, if it doesn exist the default values will be used.

> Visit the section Layout Rules Chapter to learn how to customize the 3DLayout.

Create A New Project

To create a new project use the function createProject()

An example on how to use this function.

```
var projectData = {
    "title": "Sevilla",
    "latitude": 37.39388,
    "longitude": -5.984179999999999,
    "address": "Calle María Auxiliadora, 13",
    "zip": "41003",
    "city": "Sevilla",
    "province": "Andalucía",
    "country": "Spain"
};

layout.createProject(projectData, function(data) {
    layout.bootstrap(container, {
        token: token,
        id: data.id,
        failURL: '#/fail',
        showcase: showcase,
    }, rules);
});
```

In this example we create a new project and open the 3Dlayout with this project.

The data to create a new project are:

- **title**: The title of the project
- **latitude**: The latitude were the project must be created
- **longitude**: The longitude were the project must be created
- **address**: The address of the project, this address can be different than the latitude and longitude position.
- **zip**: The zip code of the project
- **city**: The city of the project
- **province**: The province of the project
- **country**: The country of the project

3DLayout Communication System

Info Events Sent By 3DLayout

The 3DLayout trigger different events to report actions when they are accomplished or to inform on GUI changes.

An example on how to listen this events

```
var container = window.document.getElementById('ezzing3d');

container.addEventListener("buildingSelected", function(event, data){
    console.log(event.detail);
});
```

The full list of events emmited by the 3DLayout are:

- zoomChanged
- fullscreen
- tabChanged
- editArea
- editKeepout
- editTree
- areaChanged
- buildingChanged
- roofChanged
- editRoof
- editVertices
- buildingRemoved
- buildingSelected
- buildingCreated

ZoomChanged

This event is triggered when the zoom is changed in the canvas. It sends the zoom level value.

Fullscreen

This event is triggered when the user changes from normal view to fullscreen. It sends **true** when changing to fullscreen and **false** when disabling fullscreen mode

TabChanged

This event is triggerd each time the user changes the aside panel navigation tab. It sends a string with the current tab name, the values can be one of this: ["building", "areas", "keepouts", "trees"]

EditArea

This event is triggerd each time the user enters the edit section of an area. The event sends the **area.id**

EditKeepout

This event is triggerd each time the user enters the edit section of a keepout. The event sends the **keepout.id**

EditTree

This event is triggerd each time the user enters the edit section of a tree. The event sends the **tree.id**

AreaChanged

This event is triggerd each time an area attribute is changed. The event sends back an array with this info:

```
[area.id, attribute, value]
```

BuildingChanged

This event is triggerd each time a building is changed. The event sends back an array with this info:

```
[building.id, building attribute, value]
```

RoofChanged

This event is triggerd each time a roof attribute is changed. The event sends back an array with this info:

```
[building.id, roof attribute, value]
```

EditRoof

This event is triggerd each time the user enters the roof edit section of a building. The event sends the **building.id**

EditVertices

This event is triggerd each time the user enters the vertices edit section of a building. The event sends the **building.id**

BuildingRemoved

This event is triggerd each time a building is deleted. The event sends the **building.id** (after this operation this building no longer exist in the project)

BuildingSelected

This event is triggerd each time a new building becomes active. The event sends the **building.id**

BuildingCreated

This event is triggerd each time a new building is created. The event sends the **building.id**

Functions To Retrieve Info From 3DLayout

There are a set of functions to retrieve information from the 3DLayout.

For all these functions you can pass a callback as an argument to be executed when data is retrieved.

Generic Functions

Set of generic functions to retrieve project information from the layout. You just need to pass the **callback**, no other arguments are needed.

- getCurrentBuildingId
- getLayoutData
- getNumberOfModules
- getTotalPower
- getPower

GetCurrentBuildingId

```
layout.getCurrentBuildingId(callback);
```

This function returns the id value of the current active building.

GetLayoutData

```
layout.getLayoutData(callback);
```

This function returns a JSON with an array of buildings.

Each building in the array contains:

```
{
    id: the building id,
    name: the building name,
    areas: an array of areas in the building
}
```

each area in the areas array contains:

```
{
    id: the area id,
    name: the area name
}
```

GetNumberOfModules

```
layout.getNumberOfModules(callback);
```

This function returns a JSON with an array of buildings.

Each building in the array contains:

```
{
    id: the building id,
    name: the building name,
    number of modules: total of modules in the building
    areas: an array of areas in the building
}
```

Each area in the areas array contains:

```
{
    id: the area id,
    name: the area name,
    number of modules: total of modules in the area
}
```

GetTotalPower

```
layout.getTotalPower(callback);
```

Returns the total power for all the buildings in the project

GetPower

```
layout.getPower(callback);
```

Returns an array of all buildings in the project

Each building in the array contains:

```
{
    id: the building id,
    name: the building name,
    power: the total power for this building,
    areas: array of areas in this building
}
```

Each area in the areas array contains:

```
{
    id: the area id,
    name: the area name,
    power: total power in this area
}
```

Building Related Functions

Set of generic functions to retrieve building related information from the layout. In this set of functions you should pass an existing building id, and a callback.

- getBuildingInfo
- getRoofInfo
- getBuildingPosition

GetBuildingInfo

```
layout.getBuildingInfo(id, callback);
```

Returns building information for a given building.id

The data returned is:

```
{
    id: the building id,
    name: the building name,
    height: building height (in meters),
    regular: true if building angles are all equal to 90º, false otherwise.
    buildingArea: building area measure (in square meters),
    vertices: building vertices in lat/long coordinates,
    modules: total of modules in the building
    power: total power of the building,
}
```

GetRoofInfo

```
layout.getRoofInfo(id, callback);
```

Returns roof information for a given building.id]

The data returned is:

```
{
    height: roof height (in meters, not including building height),
    inclination: roof angle (in degrees),
    material: roof material (i.e: tiled/corugated),
    orientation: roof orientation (i.e: east/west or nort/south),
    type: roof type (i.e: flat, pent, gabled, etc...)
}
```

GetBuildingPosition

```
layout.getBuildingPosition(id, callback);
```

Returns building position info for a given building.id

The data returned is:

```
{
    center: the building center in lat/long coords
    vertices: an array of building vertices in lat/long coords
}
```

Area Related Functions

Set of generic functions to retrieve Area related information from the layout. In this set of functions you should pass an existing area id, and a callback.

- getAreaInfo
- getModuleInfoByArea
- getModulesSructureByArea
- getAreaOffset

GetAreaInfo

```
layout.getAreaInfo(id, callback);
```

returns area info for a given area.id]

The data returned is:

```
{
    id: the area id,
    name: the area name,
    offset: the area offset,
    placement: placement (i.e: portrait / landscape),
    structure: i.e: east-west / standard,
    inclination: modules inclination (in degrees),
    azimuth: modules azimuthal inclination (in degrees),
    areaMCoords: array containing area vertices coordinates in meters (with origin in the
building center),
    areaOffsetMCoords: array containing offseted area vertices coordinates in meters (with
origin in the building center) ,
    wallSizes: size in meters for each area wall,
    wallAzimuth: azimuthal angle for the external area wall,
    power: total power of the area.
}
```

GetModuleInfoByArea

```
layout.getModuleInfoByArea(id, callback);
```

returns module info for a given area.id]

The data returned is:

```
{
    id: the module id,
    name: the module model name,
    reference: extra model information,
    width: the width of the module (in meters),
    height: the height of the module (in meters),
    length: the lenght of the module (in meters),
    power: the power of the module
}
```

GetModulesSructureByArea

```
layout.getModulesSructureByArea(id, callback);
```

returns a JSON with an array of modules for a given area.id

The data for each module in the array is:

```
{
    x: x position of the module in meters (with origin in the building center),
    y: y position of the module in meters (with origin in the building center),
    col: column to which the module belongs,
    row: row to which the module belongs,,
    rX: rotation of the module in the X axis (inclination),
    rZ: rotation of the module in the Z axis (azimuth),
    color: the color of the module (only exist if color is not default),
}
```

GetAreaOffset

```
layout.getAreaOffset(id, offset, callback);
```

Returns an array of vertices containing the offseted area for a given area.id and offset

If the offset is a negative value, then the area is reduced by the offset value (in meters)

Layout Rules

The user can customize many options in the 3DLayout. By passing a 'rules' attribute to the 3DLayout instance, with a collection of objects, you can define the default values, add special behaviours to the 3DLayout and customize the interface.

Example of rules object:

```
var rules = {
    'display': true,
    'zoom': 21,
    'logo': false,
    'CustomPalette': ['#ff0000', '#00ff00']
}
```

Available rule objects expected by the 3DLayout:

- Special
  - display
  - zoom
  - logo
- Default Values
  - modules
  - DefaultRoofs
  - DefaultBuilding
  - CustomPalette
- Custom Buttons
  - MainoptionsCustomButtons
  - ControlCustomButtons
  - AsideCustomButtons

Special Behaviours

Display

When this options is set to true, the project starts in perspective mode.

```
{'display': true}
```

Zoom

You can set the starting zoom value. Zoom values use to range between 17 (far) to 24 (near).

```
{'zoom': 21}
```

Logo

You can choose to show (true) or hide (false) the ezzingsolar logo from the top part of the aside panel.

```
{'logo': true}
```

Default Values

User can define the default values for modules, buildings, roofs and even the color palete for modules.

Modules

User can define the solar modules available in the 3DLayout.

Sample values to define modules:

```
{
    "modules": [
        {
            "id": 4410,
            "name": "Canadian Solar",
            "reference": "250Wp Polykristallijn",
            "power": 250,
            "length": 1.638,
            "width": 0.982,
            "height": 0.04
        }, {
            "id": 41403,
            "name": "Canadian Solar",
            "reference": "260WP All Black",
            "power": 260,
            "length": 1.638,
            "width": 0.982,
            "height": 0.04
        }
    ]
}
```

The attributes for each module are:

```
{
    id: a reference number (should be unique for each model)
    name: brand and model of the module
    reference: the reference of the model
    power: power generated by the module (in watts peak)
    lenght: lenght of the module (in meters)
    width: width of the module (in meters)
    height: height of the module (in meters)
}
```

Note: the **id** value should be diferent for each module in the list.

DefaultRoofs

User can define the default values for each roof type.

The 3DLayout currently has five available roof types: **flat**, **pent**, **gabled**, **hipped** and **pyramid**.

Only roof types described in this objects are available to the user.

Sample values to define Default Roofs:

```
{
    "DefaultRoofs": {
```

```
    'gabled': {
        //roof related
        "roofMaterial": "tiled",
        "availableRoofMaterial": ["tiled", "corrugated"],
        "orientation": "EW",
        "availableorientation": ["EW", "SN"],
        "roofInclination": 30,
        //area related
        "moduleId": 47113,
        "structure": "Standard",
        "availableStructures": ["Standard", "EW"],
        "offset": 0.4,
        "inset": {
            "x": 0.02,
            "y": 0.02
        },
        "azimuth": "",
        "moduleInclination": 0,
        "moduleType": "portrait",
        "availableModuleType": ["portrait", "landscape"],
        "locked": [],
        "hidden": []
    },
    'hipped': {
        //roof related
        "roofMaterial": "tiled",
        "availableRoofMaterial": ["tiled", "corrugated"],
        "orientation": "EW",
        "availableorientation": ["EW", "SN"],
        "roofInclination": 30,
        //area related
        "moduleId": 47113,
        "structure": "Standard",
        "availableStructures": ["Standard", "EW"],
        "offset": 0.4,
        "inset": {
            "x": 0.02,
            "y": 0.02
        },
        "azimuth": "",
        "moduleInclination": 0,
        "moduleType": "portrait",
        "availableModuleType": ["portrait", "landscape"],
        "locked": [],
        "hidden": []
    },
    'pyramid': {
        //roof related
        "roofMaterial": "tiled",
        "availableRoofMaterial": ["tiled", "corrugated"],
        "orientation": "",
        "availableorientation": [],
        "roofInclination": 30,
        //area related
        "moduleId": 47113,
        "structure": "Standard",
        "availableStructures": ["Standard", "EW"],
        "offset": 0.4,
        "inset": {
            "x": 0.02,
            "y": 0.02
        },
        "azimuth": "",
        "moduleInclination": 0,
        "moduleType": "portrait",
        "availableModuleType": ["portrait", "landscape"],
        "locked": [],
        "hidden": []
    },
    'pent': {
        //roof related
        "roofMaterial": "tiled",
        "availableRoofMaterial": ["tiled", "corrugated"],
        "orientation": "S",
        "availableorientation": ["E", "W", "N", "S"],
        "roofInclination": 30,
        //area related
```

```
            "moduleId": 47113,
            "structure": "Standard",
            "availableStructures": ["Standard", "EW"],
            "offset": 0.4,
            "inset": {
                "x": 0.02,
                "y": 0.02
            },
            "azimuth": "",
            "moduleInclination": 0,
            "moduleType": "portrait",
            "availableModuleType": ["portrait", "landscape"],
            "locked": [],
            "hidden": []
        },
        'flat': {
            //roof related
            "roofMaterial": "Bitum",
            "availableRoofMaterial": ["Bitum", "Concrete", "EPDM", "PVC"],
            "orientation": "",
            "availableorientation": [],
            "roofInclination": 0,
            //area related
            "moduleId": 47113,
            "structure": "EW",
            "availableStructures": ["EW", "Standard"],
            "offset": 1,
            "inset": {
                "x": 0.02,
                "y": 0.01
            },
            "azimuth": "",
            "moduleInclination": 10,
            "availableModuleInclination": [10, 15, 20],
            "moduleType": "landscape",
            "availableModuleType": ["portrait", "landscape"],
            "locked": [],
            "hidden": []
        },
    }
}
```

Attributes explanation for default roof description

```
{
    //roof related
    "roofMaterial": should be one from the availableRoofMaterial list,
    "availableRoofMaterial": array of strings with the roof material names,
    "orientation": should be one from the availableorientation list, it can be an empty string for
flat types,
    "availableorientation": array of strings with the orientation names. ['EW', 'SN'] or ["E", "W",
"N", "S"]
    "roofInclination": angle for the roof (in degrees),
    //area related
    "moduleId": it is the default module model id, it is an optional attribute, if it doesn't exist the
first model in the module list will be used,
    "structure": should be one from the availableStructures list,
    "availableStructures": array of strings with the structure names ["EW", "Standard"],
    "offset": the distance to avoid from area borders (in meters),
    "inset": {
        "x": the distance between module borders in the x axis (in meters) ,
        "y":  the distance between module borders in the x axis (in meters)
    },
    "azimuth": angle between the module and the north pole, it can be set as an empty string
to let the 3DLayout to calculate the angle according to the external wall of the area,
    "moduleInclination": default module inclination. if availableModuleInclination is an array,
this value should match one of the options in the array,
    "availableModuleInclination": this value can be an array of integers to restrict the
inclination to this values. It can also be defined as an empty string to ignore the restriction,
    "moduleType": should be one from the availableModuleType list,
    "availableModuleType": array of strings with the available module types (i.e ["landscape",
"portrait"])
    "locked": array of strings with the area attributes that should be locked,
    "hidden": array of strings with the area attributes that should be hidden
}
```

DefaultBuilding

User can define the default building. This are the default values for each new created building.

Sample values to define default building:

```
{
    "DefaultBuilding": {
        data: {
            height: 10,
            roof: {
                type: 'gabled',
                inclination: 30,
                orientation: 'EW',
                material: 'tiled'
            },
            vertices: [],
            areas: [],
            keepouts: []
        }
    }
}
```

The attributes for the default building:

```
{
    data: {
        height: default height of the building (in meters),
        roof: {
            type: roof type (one of the available types in the defaultRoofs object),
            inclination: angle for the roof (in degrees),
            orientation: orientation of the roof ridge ('EW' for east-west or 'NS' for  north-south
orientation),
            material: material of the roof (one of the available materials in the defaultRoofs
object),
        },
        vertices: []
        areas: [],
        keepouts: []
    }
}
```

CustomPalette

User can define a custom set of hexadecimal colors. This colors are used to define module colors.

To define your own color palette use this squeme:

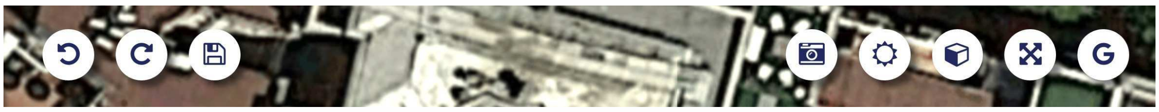
```
{
    "CustomPalette": [
        '#242345',
        '#000000',
        '#ff0000'
    ]
}
```

Custom Buttons

There are several areas in the 3DLayout where the user can add his own functions. This areas are the **main options buttons** and the **control buttons**. Each one are defined in a JSON description that can be added to the layoutRules object.

When the user adds a new button, this is defined as a event emitter.
We use the fontawesome icons collection, so you can use it to define new buttons.

MainoptionsCustomButtons



The buttons aligned to the right are custom buttons. User can define new custom buttons in this area.

Here you can hide some existing buttoms, change the order of them and create new ones with the ability to trigger an event.

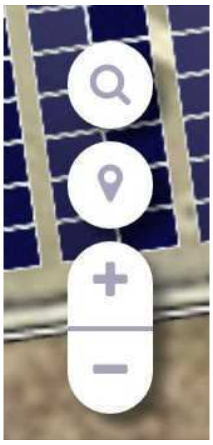
This is the default MainoptionsCustomButtons:

```
{
    "MainoptionsCustomButtons": [
        {
            click: "snapshot()",
            tooltip: "takeSnapshot",
            class: "fa fa-fw fa-camera-retro"
        }, {
            click: "widgets.simulation.toggle()",
            tooltip: "sunSimulation",
            class: "fa fa-fw fa-sun-o"
        }, {
            click: "setCamera('orthographic')",
            hide: "display.camera.name === 'orthographic'",
            tooltip: "switchTo2D",
            class: "fa fa-fw fa-square-o"
        }, {
            click: "setCamera('perspective')",
            hide: "display.camera.name === 'perspective'",
            tooltip: "switchTo3D",
            class: "fa fa-fw fa-cube"
        },{
            click: "setScreen('full')",
            hide: "fullScreen",
            tooltip: "switch to full screen",
            class: "fa fa-fw fa-arrows-alt"
        }, {
            click: "setScreen('normal')",
            hide: "!fullScreen",
            tooltip: "cancel full screen",
            class: "fa fa-fw fa-desktop"
        }, {
            click: "setProvider(0)",
            hide: "map.provider === 0 || map.provider === 1",
            tooltip: "cyclomedia",
            class: "fa fa-fw fa-map"
        }, {
            click: "setProvider(1)",
            hide: "map.provider === 1 || map.provider === 2",
            tooltip: "grid",
            class: "fa fa-fw fa-th"
        }, {
            click: "setProvider(2)",
            hide: "map.provider === 2 || map.provider === 0",
            tooltip: "google",
            class: "fa fa-fw fa-google"
        }
    ]
}
```

To customize this buttoms you can comment out the lines of the button you want to hide or extend the list by adding new objects with the same structure

```
{
    click: a string with the event name you want to trigger,
    hide: a condition to hide the button (this value is optional),
    tooltip: a string with the operator description, to show as a tooltip,
    class: a fontawesome class to define the icon
}
```

ControlCustomButtons



The buttons over the zoom in and zoom out can be customized the same way the mainoptions custom buttons.

This is the default ControlCustomButtons:

```
{
    "ControlCustomButtons": [
        {
            click: 'widgets.geocoding.toggle()',
            tooltip: 'searchAddress',
            class: 'fa fa-fw fa-search'
        },
        {
            click: '!geolocating && geolocate()',
            hide: 'geolocating',
            tooltip: 'geolocation',
            class: 'fa fa-fw fa-map-marker'
        },
        {
            click: '!geolocating && geolocate()',
            hide: '!geolocating',
            tooltip: 'geolocation',
            class: 'fa fa-fw fa-spinner ng-class:{"fa-spin": geolocating}'
        }
    ]
}
```

To customize this buttoms you can comment out the lines of the button you want to hide or extend the list by adding new objects with the same structure

```
{
    click: a string with the event name you want to trigger,
    hide: a condition to hide the button (this value is optional),
    tooltip: a string with the operator description, to show as a tooltip,
    class: a fontawesome class to define the icon
}
```

ezzingsolar®