EXPIVI

Expivi - Viewer integration

Version 1.07

Created by	Siamak Mirzaie
Version	1.07
Updated at	15/08/2019

Table of contents

Introduction

This document describes the technical implementation of Expivi viewer into third party web pages or e-commerce solutions.

Expivi's Viewer API provides a communication layer through window.expivi.* and allows the creation of the 3D viewer on webpage, resolve product attributes and configure the product through API.

The document includes the following subjects on integration:

- Browser support
- Webpage inclusions and containers
- 3D Viewer
- Serialization
- Sample code

Browser support

Expivi's 3D Configurator requires WebGL support on client's browser and hardware (supported by all major browsers and desktop/mobile devices).

The minimum required browser version includes:

- Desktop:
  - Chrome 8+
  - Firefox 4+
  - Safari 8+
  - Opera 12.1+
  - Edge 12+
  - IE 11
- Mobile:
  - Safari 8+
  - Android browser 67+
  - Blackberry browser 10+
  - Opera mobile 12+
  - Chrome 71+
  - Firefox 64+
  - IE 11+

Webpage script inclusion and containers

In order to create an instance of the viewer, the webpage must have included required scripts and provide container to place the 3D viewer inside.

Scripts

Include the following javascript libraries inside the header tag (***<header></header>***) of HTML that is going to contain the configurator.

```
<script src="{api_asset_url}/{version}/viewer.js"></script>
```

{api_asset_url}
Replace {api_asset_url} with Expivi's provided asset directory.

{version}
Replace {version} with Expivi's viewer version:

Version	**Release date**
1.41	17/06/2019
1.42	29/06/2019
1.422	04/07/2019
1.423	09/07/2019
1.425	15/08/2019

Note
The library needs to be inserted into the page before the creation/initialization of expivi's viewer. This could be at any place inside your HTML page, within header or body. Including the library inside the header is meant only as a suggestion for best practice.

Containers

The configurator requires 1 container element inside the body tag (***<body></body>***) to render the 3D viewer inside.
The added element must contain an *id* attribute to be passed to viewer's constructor method.

Example container:

```
<div id="viewerContainer" style="width: 100%;height: 600px;"></div>
```

> ***Note***
> The viewer will fill its parenting container's size. The position and responsiveness of containers is left to webpage implementation.

3D Viewer

With the script and container included into the HTML page (see: Webpage script inclusion and containers), an instance of the viewer can be created to initialize and render the Expivi's elements into the provided container.

Creating an instance

Create a new instance of *ExpiviConfigurator* object after the window has loaded.

```
<script type="text/javascript">
  window.addEventListener('load', function () {
    window.expivi._create(aToken : string,
                          aCatalogueId : number,
                          aViewContainer : any,
                          aOptions? : {
                              cameraDropdownVisible : boolean,
                              defaultOptions? : {[key : string] : string},
                              defaultMaterials? : {[key : string] : string},
                              defaultApplied? : {[key : string] : string},
                              configuredProduct?: ConfiguredProductOptions[],
                              temporaryProductId? : string,
                              show360Indicator? : boolean,
                              showProgress? : boolean,
                              preset: string
                            }
                          );
  });
</script>
```

Note
Add the script tag at the end of body tag.

Constructor options

The viewer must be initialized with the following options:

```
(
 aToken : string,
 aCatalogueId : number,
 aViewContainer : any,
 aOptions? : {
    cameraDropdownVisible : boolean,
    defaultOptions? : {[key : string] : string},
    defaultMaterials? : {[key : string] : string},
    defaultApplied? : {[key : string] : string},
    configuredProduct?: ConfiguredProductOptions[],
    temporaryProductId? : string,
    show360Indicator? : boolean,
    showProgress? : boolean,
    preset: string
  }
);
```

aToken
API Token with read permission created from Expivi's admin panel.

aCatalogueId
The id of the product to be loaded.

aViewContainer
The element's selector to place the 3D Viewer inside (eg. '#viewerContainer' as defined in Containers).

aOptions [optional]
The options are optional and allow to customize the following settings.

cameraDropdownVisible [optional]
Show dropdown to select available cameras.
(false by default)

defaultOptions [optional]
A key-value object with *attribute id* as key and *attribute value* as value to load the product with.

defaultMaterials [optional]
A key-value object with *material group id* as key and *material reference id* as value to load the product with.

defaultApplied [optional]
A specialized key-value object to load product with applied materials (using material id instead of material references) with *material group id* as key and *material id* as value to load the product with.

configuredProduct [optional]
The serialized json object of a previous configuration created by *saveConfiguration()* method.

temporaryProductId [optional]
A custom id to assign to product. This id will be returned in *saveConfuration()* object as ***temporary_id*** along the serialized product.

show360Indicator *[optional]*
A boolean (true or false) to whether show the 360 animated icon when configurator loads. (true by default)

showProgress [optional]
Show the progress percentage in loading bar. (true by default)

preset [optional]
Load a preset product.

Note:
When using presets, it is important to dispatch the client_ready event to the viewer.

```
window.expivi._events.onReady.subscribe(() => {
        window.expivi._dispatch('client_ready').then(e => {
        });
});
```

Viewer API interaction

Expivi provides a communication layer through window.expivi.*.
The complete communication layer of Expivi's namespace consist as below:

```
{
   _events: {
      onReady: new ReplaySubject(1),
      onChange: new Subject()
   },
   _dispatch: Promise<any>,
   _create: _create,
   _destroy: _destroy,
   managers: {
      resolveMaterialGroups:  Promise<any>,
      resolveMaterialGroupManager:  Promise<any>,
      resolveOffsetGroups:  Promise<any>,
      resolveMaterials:  Promise<any>,
      resolveTextures:  Promise<any>,
      resolveMaterialManager:  Promise<any>,
   },
   getProductAttributes:  Promise<any>,
   setProductAttribute:  Promise<any>,
   setProductAttributeNamed: Promise<any>,
   getActiveMaterial:  Promise<any>,
   setActiveMaterial:  Promise<any>,
   getScreenshot:  Promise<any>,
   getPrice:  Promise<any>,
   setNodeProperty:  Promise<any>,
   changeCameraZoom:  Promise<any>,
   saveConfiguration:  Promise<any>,
   loadConfiguration:  Promise<any>,
   loadPreset:  Promise<any>,
   selectProductNode:  Promise<any>,
   autoRotateYAxis: Promise<any>,
   deselect:  Promise<any>,
   deleteNode:  Promise<any>
}
```

Event handling

After creation Expivi viewer will start to send events to client about its state and changes. There are two streams to subscribe to. The streams exists under ***window.expivi._events.****

onReady

This event is sent only once after Expivi viewer has initialized and is ready to interact with.

```
window.expivi._events.onReady.subscribe(function(){});
```

> ***Note***
> onReady event is of ReplaySubject type. Meaning once it has received the ready event, it will keep resending it on any new subscription, even when the subscription has taken place after the initial ready event is sent.

onChange

On every change in viewer, an event is sent to client, often these changes are requested by client itself. The client could respond to changes, such as change in attribute selection from rules, if needed.

```
window.expivi._events.onChange.subscribe(function(event){
  console.log('Event name:' + event.name + 'Event payload:', event.payload);
});
```

onChange events

The onChange event contains the following payload:

```
{
  name : string,
  payload? : any
}
```

name
The name of event received.

payload [nullable]
Event's payload. Based on every event the payload can differ or not exist.

Possible events

- 'frame': the viewer has updated the rendered view.
- 'zoom': the zoom of the camera has changed.
- 'buildready': sent once after the initial build of a product is ready.
- 'buildupdated': sent every time the product configuration has been updated.
- 'attribute': sent every time an attribute of the configuration has been changed.
    - Payload:

```
attribute_id: number

attribute_type: string

attribute_value: any //change value

attribute_value_id: number //id of value changes

attribute_value_name: string//string representation of change value

product_id: number //id of changed product (catalogue_id)

product_scene_id: number //id of product in scene

rebuild: boolean //whether the 3d will be updated after this change
```

- 'material': sent every time a material of a product has been changed
    - Payload:

```
group: string //material group name (unique)

id: number //id of material group

material_id: number //id of selected material

product_uuid: string //uuid of product on which material has changed

ref_id: number //id of selected material reference on material group
```

Managers

```
{
    resolveMaterialGroups:  Promise<any>,
    resolveOffsetGroups:  Promise<any>,
    resolveMaterials:  Promise<any>,
    resolveTextures:  Promise<any>
}
```

window.expivi.managers.* namespace provides access to some of the underlying managers within Expivi's viewer.

> ***Note***
> window.expivi.managers.* is meant to be used only as read-only. All interactions with the API are required to take place through window.expivi.* methods.

window.expivi.managers.resolveMaterialGroups returns an array of all available material groups with corresponding materials within each group that can be applied/selected on each group.

```
Array<{
id : number,//id of the group
hash : string,//unique hash
name : string,//human readable name of group
group_id : string,//unique group name
material_group_sku : string,//sku used for the material group
generate_code : boolean,//defines whether this group is used in bill of materials
catalogue_id : number,//id of the product this group belongs to
materials? : MaterialGroupRefModel[],//materials inside the group that can be selected
}>
```

MaterialGroupRefModel

```
{
 id : number;//reference id used to select material
 sku : string;//sku of the material
 name? : string;//name used to be presented to end-user
 price? : number;//price applied when this material is selected
 thumbnail? : string;//thumbnail of the material
 thumbnail_url? : string;//full path to the thumbnail
 default_selected : boolean;//whether this material is the default selected inside the group
 material_id : number;//the material id is being referenced
 material_name? : string;//material name being referenced
 visible? : boolean;//whether the material is visible in list
}
```

window.expivi.managers.resolveOffsetGroups returns an array of all offsetting nodes inside the product node.

```
Array<{
 id : number,//id of the group
 hash : string,//unique hash
 offset_id : string,//unique group name
 offset_vec_x : number,//offset in x axis
 offset_vec_y : number,//offset in y axis
 offset_vec_z : number,//offset in  axis
 matrix : number[],//offset's initial matrix
 rotation? : THREE.Quaternion,//applied rotation on the offset
 levels? : {[key : number]: THREE.Vector3},//stacked offsets in x,y,z axis
 visible? : boolean,//whether the node is visible
}>;
```

window.expivi.managers.resolveMaterials returns an array of all loaded materials inside the product.

```
Array<MaterialModel>//A complete list of material properties are out of scope of this documentation.
```

window.expivi.managers.resolveTextures returns an object of all loaded media models.

```
{[key : number]:
        {
         id : number;//id of the media file
         name : string;//name of the media file
         source_url : string;//complete path to the source of media file
         lod0? : string;//highest level of detail of the image media
         lod0_url? : string;//full path to lod0
         lod1? : string;//medium level of detail of the image media
         lod1_url? : string;//full path to lod1
         lod2? : string;//low level of detail of the image media
         lod2_url? : string;//full path to the lod2
         width : number;//width of the source image file
         height : number;//height of the source image file
         uv_scale_x : number;//scale used to down/upscale width to power of two size
         uv_scale_y : number;//scale used to down/upscale height to power of two size
         resource_folder_id : number;//the id of virtual folder containing the media
        }
}
```

API Methods

getProductAttributes

Get available attributes on a product. The client can apply/change these attributes then using setProductAttribute() method.

```
window.expivi.getProductAttributes().then(function(productAttributes){
  console.log('available attributes: ', productAttributes);
});
```

Attribute interface

```
{
  id : number; //attribute's id to be used to set Attribute
  name : string;//attributes name
  type : string; //type (question, matgroup, text_input, text_to_image, etc)
  slug? : string;//attributes slug
  hash? : string;//unique hash
  description : string;//description
  position : number;//position of attribute in rendered list
  thumbnail : string;//thumbnail file
  thumbnail_url : string;//complete url of thumbnail
  parent_id : number;//attribute's parenting attribute [nullable]
  meta : any;//meta represented as meta
  visibility : number;//whether the attribute is visible during initialization
  global : boolean;//is attribute local to product or global used by more products
  attribute_values? : AttributeValueInterface[];//available values, polymorphic based on type
  entity_properties? : EntityPropertyInterface[];//assigned entity properties on attribute
  selected_value_id? : number;//the currently selected attribute value id (term)
  selected_value_value? : any;//the currently selected value
  visible? : boolean;//visibility changed by rules system
}
```

setProductAttribute

Set/change an attribute's value on product.

```
window.expivi.setProductAttribute(aAttributeId : number, aValue : any);
```

aAttributeId
The id of the attribute to change its value.

aValue
Depending on the attribute, the provided value should correspond to the attribute's type.

Possible values based on attribute's type:

- 'question' : number (attribute values's id)
- 'matgroup' : number (material reference's id)
- 'text_input' : string //text to be set on input
- 'text_to_image' : string //text to be set
- 'image_upload' : string //url of uploaded image
- 'color_picker' : string //hex code
- 'color' : string //hex code

setProductAttributeNamed

Set/Change an attribute's value on product.

```
window.expivi.setProductAttribute(aAttributeName : string, aValue : any);
```

aAttributeName
The name of the attribute to change its value.

aValue
Depending on the attribute, the provided value should correspond to the attribute's type.
See: setProductAttribute

getActiveMaterial

Get the actively selected reference id of material inside a material group.

```
window.expivi.getActiveMaterial(aMaterialGroupId : number, aProductSceneId? : string) :
Promise<number>;
```

aMaterialGroupId
The ‘id’ of material group.

aProductSceneId (optional)
The unique product id containing the material group (resolved from *window.expivi.resolveProductNodes()*).

Returns
Promise containing the active reference id.

setActiveMaterial

Set/select the active material inside a material group using its material reference id.

```
window.expivi.setActiveMaterial(aMaterialGroupId : number, aReferenceId : number,
aProductSceneId? : string) : Promise<number>;
```

aMaterialGroupId
The ‘id’ of a material group.

aReferenceId
The reference id of the material inside a material group (see manager->resolveMaterialGroups).

aProductSceneId (optional)
The unique product id containing the material group (resolved from *window.expivi.resolveProductNodes()*).

Returns
Promise containing the selected reference id.

getScreenshot

Render the view to an image Base64 data and return it.

```
window.expivi.getScreenshot(aWidth : number, aHeight : number) : Promise<string>;
```

aWidth (optional)
The width of the screenshot.
Default 512.

aHeight (optional)
The height of the screenshot.
Default 512.

Returns
Promise containing string with Base64 Image data.

getPrice

Calculate and return the total price of a product.

```
window.expivi.getPrice(aProductSceneId? : string) : Promise<number>;
```

aProductSceneId (optional)
The unique product id to get its price (resolved from *window.expivi.resolveProductNodes()*).

Returns
Promise containing numeric value of the calculated price.

setNodeProperty

Set/change an internal property of a scene object.

```
window.expivi.setNodeProperty(aNodeName : string, aProperty : {name : string, value : any})
: Promise<number>;
```

aNodeName
The name of the scene object node to change its property.

aProperty
An object { name : string, value : any } containing the name of the property and the value of the property to be set.

Returns
Promise containing number of nodes affected.

changeCameraZoom

Change the camera zoom by delta.

```
window.expivi.changeCameraZoom(aZoomDelta : number) : Promise<number>
```

aZoomDelta
The delta, in negative or positive direction, to change the zoom with.

> ***Note***
> The zoom will be capped by minimum & maximum zoom defined in Expivi's editor.

Returns
Promise containing the applied delta.

saveConfiguration

Save/Serialize the configured product (see Serialization).

```
window.expivi.saveConfiguration(aWidth : number, aHeight : number) : Promise<{
bundle_uuid : string,
configured_products: []
}>;
```

aWidth (optional)
The width of the screenshot.
Default 512.

aHeight (optional)
The height of the screenshot.
Default 512.

loadConfiguration

Load a previously configured product (serialized with saveConfiguration).

```
window.expivi.loadConfiguration(aConfiguration : {
bundle_uuid : string,
configured_products: []
}) : Promise<boolean>;
```

Note
Reloading previous configurations can be done during initialization by providing the aConfiguredProduct attribute.

loadPreset

Load a preset configuration.

```
window.expivi.loadPreset(aProductSceneId : string | null, aPresetId : string) :
Promise<boolean>;
```

Note
You can apply preset only on an already created product of same catalogue id.

selectProductNode

Select active product node.

```
window.expivi.selectProductNode(aProductSceneId? : string) : Promise<ProductNode>;
```

aProductSceneId (optional)
The unique product id to select (resolved from *window.expivi.resolveProductNodes()*).
Default: main product node.

Note
When having more than one configurable product inside the viewer, active product node can be changed to set/get its attributes without providing product id. Selecting a product node is also useful when movement of objects inside the scene is also activated, selecting products will activate their movement options in that case.

autoRotateYAxis

Rotate the product automatically on the Y axis.

```
window.expivi.autoRotateYAxis(aProductSceneId? : string, aStopOnInteraction = true,
aSpeed = 1, aMaxDegree = 360) : Promise<ProductNode>;
```

aProductSceneId (optional)
The unique product id to select (resolved from *window.expivi.resolveProductNodes()*).
Default: main product node.

aStopOnInteraction (optional)

Stop the automatic rotation when user interacts with the viewer.
Default: true

aSpeed (optional)
The rotation speed of the animation.
Default: 1.0

aMaxDegree (optional)
The degree at which the animation will automatically stop.
Default: 360.0 | *infinite: -1*

deselect

Deselect any selected node inside the viewer.

```
window.expivi.deselect() : Promise<boolean>;
```

deleteNode

Delete a scene node from the viewer.

```
window.expivi.deleteNode(aSceneNodeId : string) : Promise<SceneNode>;
```

aSceneNodeId
Unique scene node id (in case of product node resolved from *window.expivi.resolveProductNodes()*).

Returns
Promise containing the deleted scene node (if any).

Serialization

The configuration created by a user can be serialized to a JSON object to be communicated to webshop cart; or to be saved to storage to reload a product, to a pre-configured state.

Saving configuration

Saving the configuration will create a thumbnail of 3D rendered product (which can be used by webshop cart), provide representable specifications (visible and selected attributes by user) and a snapshot of attribute's state for internal reloading purposes.

Method

```
window.expivi.saveConfiguration(width, height) : Promise<ConfiguredProductOptions[]>;
```

Parameters

- width: the width of thumbnail to be created.
- height: the height of thumbnail to be created.

Return

A promise which resolves an array of *ConfiguredProductOptions*

ConfiguredProductOptions

```
interface ConfiguredProductOptions {
  version : string; //interface version
  catalogue_id : string;//product's id
  thumbnail? : string;//thumbnail as base64 data
  configuration : {
    attributes : AttributeSerializedValue[]//configuration state for internal usage
  };
  attributes : AttributeSerializedValue[];//representable product specifications
  articles? : ProductArticleModel[];//bill of materials
}
```

AttributeSerializedValue

```
interface AttributeSerializedValue {
  attribute_id : number;//attribute's id in database
  attribute_name? : string;//attribute's name (product specification name)
  attribute_value_name? : string;//attribute value (product specification value)
}
```

ProductArticleModel

```
interface ProductArticleModel {
  models_id? : number;//component's id in database
  models_sku : string;//part's sku
  material_group_id? : number;//applied material's id in database
  material_group_sku? : string;//sku of applied material on part
  price_sku? : string;//price identifier to look into pricing matrix
  price_fields? : {[key : string]: number};//calculated total price for each predefined field
}
```

Example call

```
window.expivi.saveConfiguration(256, 256).then(function(aSaveResults){
  console.log("products thumbnail: ", aSaveResults[0].thumbnail);
  console.log("products specifications: ", aSaveResults[0].attributes);
  console.log("products articles: ", aSaveResults[0].articles);
});
```

Sample code

The following sample contains a responsive fullscreen example layout with configurator.

```
<html>
   <head>
     <title>Expivi - 3D Viewer sample code</title>

     <script src="vendor.lib.js"></script>
     <script src="index.js"></script>
  </head>
  <body style="width: 100%;height: 100%;margin:0;padding:0">

    <div id="viewerContainer" style="width: 100%;height: 600px;"></div>

    <script type="text/javascript">
      var apiToken = 'YOUR_TOKEN';
      window.addEventListener('load', function () {

          /**
          Create expivi viewer
          @param token string
          @param catalogue_id product's catalogue id
          @param container string dom selector to put viewer
          @param showDropdownCamera boolean
          **/
          window.expivi._create(apiToken,
                                421,
                                "#viewerContainer", {
                                    show360Indicator: true
                                });
```

```
        /**
        wait until expivi is ready to communicate with
        **/
        window.expivi._events.onReady.subscribe(function(){
                console.log("Expivi ready!");

                /**
                Tell expivi we are ready to receive events
                **/
                window.expivi._dispatch("client_ready").then(function(){
                        console.log("after client ready");
                        /**
                        Rotate the product automatically until user interacts with viewer
                        **/
                        window.expivi.autoRotateYAxis();

                        /**
                        Get all available attributes inside product
                        **/
                        window.expivi.getProductAttributes().then(function(attributeItems){
                                console.log("got attributes", attributeItems);
                        });
                });
        });

        /**
        subscribe to changes on viewer/configurator
        **/
        window.expivi._events.onChange.subscribe(function(aEvent){
                if(aEvent.name !== "frame"){
                        console.log("event change: ", aEvent);
                }
        });

    });
  </script>

 </body>
<html>
```