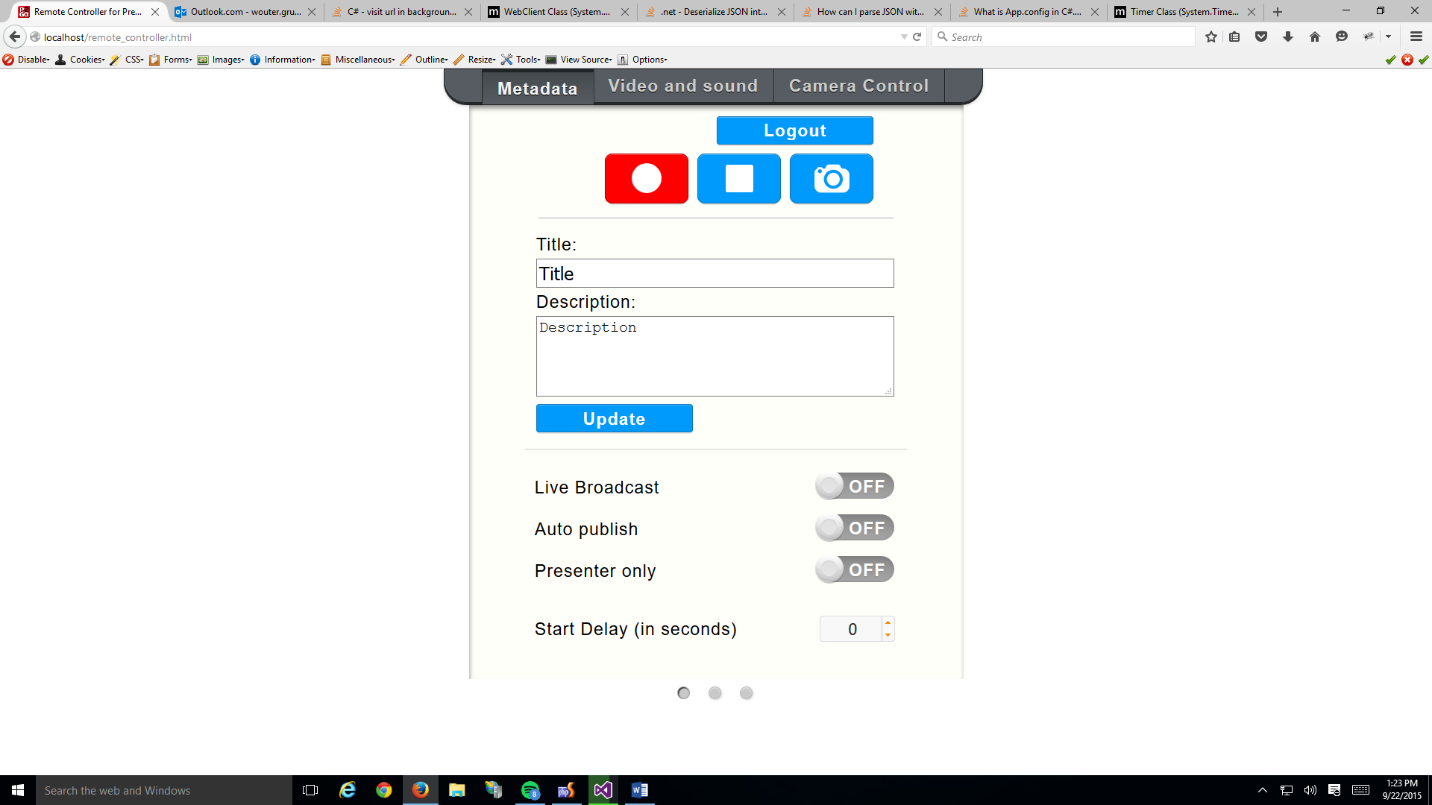
Presentations 2Go web application overview



# Introduction:

This website is made with the presentations2go encoder API. To get more information on the API go to <http://upgrade.presentations2go.eu/p2g.api/Encoder.RemoteControl/APIv3/>

# Scripts:

I made the scripts and functions within them as easily understandable by others as possible. In here I will explain why I did certain things to explain even better. Before every function I have briefly explained what the function does and what the parameters are, so you could understand the function before looking at the code itself.

You will read a lot about “successfunction”. This variable is a function that will be executed when the Ajax call has finished successfully. This way, Ajax calls with the datatype “jsonp” can be executed in order. Datatype “jsonp” is a cross browser datatype that does not support “async: false” inside the Ajax request.

Before every request to the encoder, I get a new sessionId. This way, if the software has some trouble keeping the sessionId (which it certainly sometime does), the user does not have to log back in again. When the user does something, it also happens in the software if the software is no currently being used. If it does, there is a 2 minute timer that checks if the software has different data than the user. If it is different, the user will get a prompt where he can see what is different, and he can chose which he wants to keep.

### File: “controlfunc.js” explanation

The functions which are used to connect and process data to and from the encoder are in file: “**controlfunc.js**”.

* The first function: “getNewSessionId(**successfunction**)” gets a new ‘sessionId’ from the encoder. You will need a sessionId to do just about anything with the encoder. When you move the mouse on the computer with the encoder or when you login on a different computer to this encoder, the sessionId should expire. But I have found that sometimes when you press the record button, your sessionId will also expire. This is why you will see that before every command send to the encoder, it will first get a new sessionId, so it is 100% sure whatever it sends, will happen in the encoder.
* This is why there is a **successfunction** parameter. This is a function that will be executed after it successfully received a new sessionId. Before it executes the successfunction it will put the new sessionId inside a *cookie* called “sessionId”.
* The second function “ajaxToEncoderCall(**Url**, **Type**, **successfunction**)” makes the actual call to the encoder to process your data.
* **Url**: This is the url the call will go to. This url is made in a different function, so I will be talking about how to make this url a little bit further on.
* **Type**: This is what it should do after the call was successful. Example: “GetMetaData” will execute **successfunction** and gives the value of the metadata that it received with the function.
* **successfunction**: This is a function that will be called when the call to the encoder was successful.
* The third function “processCall(**action**, **actiondetail**, **Type**, **successfunction**)” will get a new sessionId by calling function “getNewSessionId()” and gives a function on that gets data from the cookies, and makes a Url with it.

**A Url is built with the following parts**

1. http://
2. The **Ip** of the computer the encoder is connected to.
3. ?action= ‘an action. You can find all the actions in the API documentation’
4. & ‘actiondetail’, actiondetail can be a number of things. When you want to set some metadata, you can have the type of metadata = value. Ex: Title=this is a title. For more information on this, read the API documentation.
5. &user= ‘username of the encoder’
6. &password= ‘password set on the encoder’
7. &sessionId= ‘The sessionId you received earlier’

* **action:** is the action for the url. To know what actions can be, see the API documentation.
* **actiondetail:** the actiondetail in the url. Ex: Title=this is a title (inserts a title), or actionDetail=Title (gets the title).
* **Type:** will be transferred to “ajaxToEncoderCall()” function.
* **successfunction:** will be transferred to “ajaxToEncoderCall()” function.
* The fourth function “setMetaData(**type**, **data**, **successfunction**)” will set metadata in the encoder.
* **type:** the metadata that you want to set. Ex: ‘Title’ or ‘Description’.
* **data:** the data you want to set the **type** to.
* **successfunction:** will be transferred to “processCall()” function.
* The fifth function “getMetaData(**type**, **successfunction**)”
* **type:** the metadata that you want to get. Ex: ‘Title’ or ‘Description’.
* **successfunction:** will be transferred to “processCall()” function. Inside the successfunction you can set a parameter that will be the data gotten from the encoder.
* The sixth and seventh functions are the same as the fourth and fifth function but that for the configuration instead of metadata.
* The eight function “getRecStatus(**successfunction**)”
* **successfunction:** will be transferred to “processCall()” function. Inside the successfunction you can set a parameter that will be the data gotten from the encoder.

### File: “login-data.js” explanation

In the file: “**login-data.js**” all the function are defined which are used to check the login data.

* Function “checkLogin()” checks if the user has already logged in by checking its username. This is useful when you run the website location on a separate server. When the website is run inside the encoder. This step will be neglected since we already know all this data.
* Function “checkValidLogin([**givenip**], **givenpass**)” checks if the login data is valid. It does this by trying to get a sessionId. If it fails, the data is most likely wrong. When it was successful, the cookies get set to the appropriate data and you will get redirected to the remote\_controller page.
* **givenip:** is the IP that can be given to the function. In the encoder version, this parameter is not there, since there it gets the IP from “location.host”
* **givenpass:** is the password that can be given to the function.

### File: “remote-control.js” explanation

In the file: “**remote-control.js**” all the function are defined which are used in the remote\_controller page.

* Function “logout()” will log the user out.
* Function “getMetaDataAndFillHtml(**attribute**, **type**)” will get some metadata and put it inside an html attribute.
* **attribute:** an attribute inside the html that the value inside will be the metadata received from the encoder.
* **type:** the type of data you want to get from the encoder. Ex: Title.
* Function “getConfigurationAndUse(**attribute**, **type**)” will get a configuration from the encoder and changes the attribute. Right now this only works for checkboxes!
* **attribute:** an attribute inside the html that the will change to the configuration received from the encoder.
* **type:** the type of data you want to get from the encoder. Ex: live.
* Function “btnPress(**buttontype**)” this will press a button inside the encoder.
* **buttontype:** the button you want to press inside the encoder. See the API documentation to know what the buttons are called.
* Function “initializeimagevent(**imgid**, **delayms**)” will initialize an interval that updates pictures in the browser.
* **imgid:** is the id / class of the img tag that is updated.
* delayms: is the time that is between the updates of the images.
* Function “initializesingleimagevent(**imgid**)” will initialize an image once. This is used when the user passes past the middle screen, the img is updates once so the image is not out of date or empty.
* **imgid:** is the id / class of the img tag that is updated.
* Function “updatecurrentlylive(**successfunction**)” updates a variable that says of the user is currently livestreaming.
* **successfunction:** a function that will be executed after it got the current recording status.

### File: “indoc-events.js” explanation

In the file: “**indoc-events.js**” all the events inside the html pages are defined. These events are made with JQuery! The first 2 functions are the separate html documents to keep it a little more organized.

The other functions are made to have as less code to be rewritten.

### File: “Cookies/handlecookies.js” explanation

In this file all the function to handle cookies are defined.

* Function “setCookie(**cname**, **cvalue**, **exdate**)” is a function that can set and or update a cookie.
* **cname:** is the name of the cookie. To get the value of a cookie you will need the name.
* **cvalue:** is the value of the cookie.
* **exdate:** the expire date of the cookie. Nothing for end of the session and “infinite” for forever.
* Function “getCookie(**cookiename**)” is a function that can get the value of a cookie.
* **cookiename:** the name of the cookie you want to get the value of.
* Function “checkUserLoggedIn()” is a function that looks at the username cookie. If this is not empty, it returns true, else it returns false. This is useful if the web application is running on a separate server. When the website is running on the encoder itself. This is unnecessary since the encoder already knows the username and only needs a password.
* Function “checkCookiesEnables()” checks if the cookies are enabled on the user’s computer. If not, the user will be redirected to nocookiesenabled.html page.
* Function “understandcookies()” will set the cookie “understandcookie” to true forever and will have the cookieinfo bar in the top of the page disappear.

### File layout explanation

I made the file layout as easy as possible. When you go to the main directory. You should automatically go to the **index.html** page where you would be able to log in. If you are on a mobile device that the website supports, before loading anything else, you will get redirected to the mobile folder. Inside there is the same **index.html** file but changed so it looks better on a mobile device.

When logged into the encoder, you will be redirected to **remote\_controller.html** inside the current folder. So when you got redirected to the mobile site, your **remote\_controller.html** file will be inside the mobile folder. This will also be optimized to be used on a mobile device.

When using the site on a tablet, or anything that has a wider resolution than **1024px**, your style will be changed so the site will still give you the best experience possible.

All JavaScript files are inside the folder “**Scripts**” (except a couple). This is done because when using the site on a mobile device, you don’t need different script.

Everything you need for the cookies, are in the folder “**Cookies**”. There are 2 html files that are optimized for both desktop and mobile devices/tablets. When the user wants more information on the cookies in the web page, he will be redirected to “**Cookieinformation.html**”. This will explain how the site uses cookies and briefly explains what cookies are.  
When the cookies are disabled, the user gets redirected to “**nocookiesenabled.html**”. This will explain how to turn cookies on, will be able to redirect to the “Cookieinformation.html” file for more information and briefly explains what the site uses cookies for.   
With that, there is a style file. This is for all the styles inside these 2 html files.

And there is a handlecookies.js file. In this file we define all the functions that are used to set and get cookies. Detect if there are no cookies enables, and so on.

All the images for the website are stored in the “**img**” folder. To optimize the website, you will want to use as little images as possible.

The style files for the desktop version of the website, ae stored inside the “**Style**” folder.

Inside the mobile folder, some changed styles for the website itself and for the sweetalerts, are stored. With that, there is a JavaScript file to be able to use scrolling on mobile with your fingers.

# Used libraries:

### JQuery

JQuery is very nice to use with JavaScript, this library makes working with JavaScript a lot easier and faster. As their motto describes, “write less, do more”

### JQuery ui

Because JQuery itself is really good to use but does it’s more focused on making things in less time, and being able to use JavaScript easier. JQuery on its own does not make these nice ui elements. Like buttons next to an input where you can click on and the number goes up, or a nice slider to control the sound. That’s why we use JQuery ui.

### JQuery mobile

Unfortunately the normal JQuery does not have great support for mobile devices. Some things JQuery just does not have, like slide recognition. JQuery ui can see that I am sliding left or right. We can then use this to slide to the next screen on mobile. That’s why we used JQuery mobile as well.

### SweetAlert

Ooh sweetalert. The name says it all. The normal alerts in JavaScript just don’t look good enough for a casual user of the web page. Sweetalert makes the equivalent to normal JavaScript alerts, but they look a whole lot better. With that, in the code they are very easy to use. You can put actual html inside the description, this way I customized the alert so it’s very easy to use for the user.

### Extensions

I also used some more extensions, like JQuery ui touchpunch. This adds more usability to the JQuery ui library on mobile. The normal JQuery ui library does not have support for fingers tapping the screen, or sliding on the screen. Touchpunch translates the finger gestures on the screen to a normal mouse event. This way, normal JQuery ui can see the fingers of the user as an input.