## DESCRIPTION

To be able to use this asset, you need to have a basic understanding of HTML and the DOM. If these concepts are foreign to you, take your time to learn a little bit about them. Otherwise you won't know what I'm talking about.

Even though Unity has a built in way to take screenshots, it can only take screenshots of stuff happening in the game's canvas. This asset can take screenshots of everything happening in the DOM. The main use for this asset is when your game doesn't happen entirely in Unity's game canvas. An example of such a use case is when you set the camera's clear flag to *depth only* or *solid color* with *alpha 0* to show the webcam feed of another plugin. In such a case, the regular screenshot method won't work because the webcam feed is just an HTML video element and not a texture in Unity that you control.

The screenshot can be used in a Unity texture to render it in the game, sent to another server, downloaded, shared, etc. Whatever you want to do with it. The example scene shows how to render the screenshot to a Unity texture. Sharing and downloading can be achieved with my ShareNSaveWebGL asset, it's compatible with it out of the box.

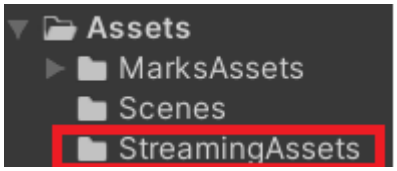
## HOW DOES IT WORK?

This plugin uses html2canvas. What html2canvas basically does is take a fake screenshot. It traverses the DOM and creates a canvas based on all the images, CSS properties, etc. The resulting canvas is then converted to an image, which is your screenshot. I say fake screenshot, because html2canvas is not actually reading the rendered pixels on the screen, but constructing an image based on the information of the DOM. The result is the same though.

## IMPORTING THE LIBRARY

There are 2 ways to import the library.

1. By importing the library at runtime automatically. It always fetches the latest version. This is the easiest method and the default one. To do this, see on the file *ScreenshotWebGL.jspre* under the plugins folder, the variable *loadDynamically*; set it to true(default). If you are using this method, delete the *html2canvas.min.jspre* file.
2. By having the library in the project itself. Set the variable *loadDynamically* to false, and move the *html2canvas.min.jspre* file in the Plugins folder to the StreamingAssets(case sensitive) folder under the Assets folder. If the folder doesn't exist, create one. Rename *html2canvas.min.jspre* to *html2canvas.min.js*(I named it with a .jspre because the asset store kept rejecting it, but it's supposed to be a .js file)




You can always find the latest version here https://cdnjs.com/libraries/html2canvas . At the time of writing it's the 1.3.2 version that is included. The file **must** be named *html2canvas.min.js*, even if you are not using the minified version.

You can define a callback to be fired when the plugin is loaded, and pass it to the *onLoaded* method. For example

```
ScreenshotWebGL.onLoaded(onLoadedCallback);
```

```
private void onLoadedCallback() {
      Debug.Log("library loaded now I can use it!")
            }
```

The callback will fire even if you called onLoaded after the library has already loaded. It will always fire once after detecting that the library is present.

**HOW CAN I USE IT?**

**Using the screenshot inside the Unity game**

To take a screenshot and use it inside Unity to render it to a texture, for example, use this method

```
public static void screenshot(Action<byte[], int, int> callback, string element = "body", string options = null, string type = "image/png", double encoderOptions = 0.92)
```

**Using the screenshot without needing it to be rendered or otherwise used in the game itself**

To take a screenshot and send it to a server, download, share, etc, a task that doesn't involve using the screenshot in the game, use the following method.

```
public static void screenshot(Action callback, string element = "body", string options = null, string type = "image/png", double encoderOptions =
```

```
0.92)
```

This variant takes a screenshot and lets it stored in the JS context, as a blob, never sending it to Unity. Specifically, it is stored at *Module.ScreenshotWebGL.screenShotBlob*. If you can use this method, it is preferable because it's faster for not having the overhead of sending the stream of bytes to Unity. You can then use this path on your plugin to do whatever you want with it. If you have my ShareNSaveWebGL asset, you can use it like this:

```
ShareNSaveWebGL.Share(shareCallback,
"Module.ScreenshotWebGL.screenShotBlob");
```

It's an instant process in my iPhone's Safari. On my Android Chrome, it still takes 2~4 seconds because of the process of traversing the dom and taking the actual screenshot(hopefully it's because my Android is old).

Let's talk about the parameters now

1. element: This is a string that is used by a querySelector to pick the DOM element that html2canvas will take a screenshot of. If you just want to take a screenshot of the game's canvas, set this option to "canvas". If you want to take a screenshot of the whole DOM, which is the main use case for this asset because you want to capture other elements besides the game's canvas, use "body"(default).
2. options: This is a string representing a JSON. You can check all the options here. Here's an example

```
ScreenshotWebGL.screenshot(screenshotCallback,"body",$"{{\"backgroundColor
\": \"#FF0000\" }}");
```

Takes a screenshot of the body and changes the background color to red. Please note that the properties **must** use double quotes. It won't work with single quotes.

3. type: The type of the image you want the screenshot to be.
4. encoderOptions: The encoding of the image. Not used for "image/png", which is the default.

There is also the callback parameter. For the first variant of the method, it's a function that takes a byte array, and 2 ints that represent the width and height of the screenshot, respectively. It's the callback that runs after the screenshot. Here's an example from the example scene

```
private void screenshotCallback(byte[] bytes, int width, int height) {
        texture2D = new Texture2D(width, height);
        texture2D.LoadImage(bytes);
        GetComponent<Renderer>().material.mainTexture = texture2D;
            transform.localScale = new Vector3((float)width/(float)height,
```

```
1.0f , 1.0f) * 5.0f;
    }
```

It takes the bytes and stores them on a texture.

For the second variant it's a callback with no arguments, because the data is never sent to Unity, and the callback is used solely to know when the screenshot is completed. Example:

```
private void screenshotCallback() {
      Debug.Log("screenshot completed");
            }
```

## IMPORTANT

**You must call `yield return new WaitForEndOfFrame();` before taking a screenshot, similar to how it's done using the official api.**

### Here are the remaining methods:

**getDOMProperty**

```
public static double getDOMProperty(string propertyPath)
```

This method gets the value of a DOM property. Here's an example:

```
ScreenshotWebGL.screenshot(screenshotCallback,"body",$"{{\"backgroundColor\":\"#FF0000\",\"scale\":{2.0*ScreenshotWebGL.getDOMProperty("window.devicePixelRatio")} }}");
```

It sets the background to red, and the scale to twice the size of devicePixelRatio to increase the sharpness of the image. The getDOMProperty method will only work with properties that start with *window* or *document*, and there is no limit to how long the path is. I am not using eval or Func to do this, so don't worry about security.

**clearScreenshotBlob**

```
public static void clearScreenshotBlob()
```

After using the variant of the screenshot method that stores a blob, you may want to clear it at some point. You can use the method above to do it.

**didLoad**

```
public static bool didLoad()
```

A method that you can call to check if the library has already loaded. It's an alternative to the onLoaded+callback combo. You'd call this method on the Update for example.

**getGamePositionX, getGamePosition, getGameWidth, getGameHeight**

```
public static double getGamePositionX()
public static double getGamePositionY()
public static double getGameWidth()
public static double getGameHeight()
```

These methods are used to retrieve the game's canvas x and y position(top left corner), the game's width and height, respectively. This is useful if you want to make cropped screenshots.

The example scene takes a screenshot 5 seconds after the game awakes. I added the delay to avoid taking a screenshot of the Unity Loading screen. Just so you know that those 5 seconds is not a delay from the screenshot. You can remove the delay if you want to.

**KNOWN ISSUES**

There are a lot of things you need to watch out for.

1. You must never disable the gameobject that is calling `yield return new WaitForEndOfFrame()` when taking a screenshot. If you do this, the yield will never return. It happened to me when I used a button to take a screenshot and then immediately used SetActive(false).
2. Don't call the screenshot method while there is already one running. Wait for it to finish, otherwise you will get an error.
3. Taking screenshots, especially when involving a video tag, can be really slow. Even if you use the screenshot method that doesn't send the data to Unity. Nothing I can do about this. On my iPhone's Safari it's instant but on my Android Chrome it takes anywhere from 2 to 4 seconds.
4. You can get an error called *RangeError: Maximum call stack size exceeded*. Check out this issue on github. If you are a victim of this bug, you will need to use method 2 of importing and manually patch the plugin. I couldn't find a way to monkey patch it, but if you find a way let me know. So here's what you are going to do:
    a. If you are using the minified version(which you should), search for the text "6e4"(it's short for 60000). Change it to a lower number. "5e4"(50000) works for me.
    b. If you are using the non-minified version, search for the text "`SLICE_STACK_SIZE`". Change the value from 60000 to something lower, like 50000.
5. You can end up taking screenshots that show white spaces on the top, bottom, left, or right. Remember that this plugin doesn't actually take a screenshot, it traverses the DOM, takes into account CSS properties, etc. Don't email me if you run into this issue. Just fiddle around with the properties until you get it right. What you can do to make this process faster, is edit the *webgl.framework.js*(on Unity 2020 and higher) file in the Build folder directly to hard code the options instead of changing the options on C# and having

to rebuild. The file will be minified, so you can use a service like this one to beautify it first and then edit. Search for *_ScreenshotOptimized_ScreenshotWebGL* if you are using the method that doesn't send data to Unity, or *_Screenshot_ScreenshotWebGL* if you are sending the screenshot to Unity. Then change
*html2canvas(document.querySelector(element), opts)*
to something like(example)
*html2canvas(document.querySelector(element), {width: 900})//completely discarding the options from C# and setting the width property to 900.*
Here's an example configuration that might fix the problem for you:

```
ScreenshotWebGL.screenshot(screenshotCallback,"body",$"{{\"width\":{
ScreenshotWebGL.getDOMProperty("document.documentElement.offsetWidth
")},\"height\":{ScreenshotWebGL.getDOMProperty("document.documentEle
ment.offsetHeight")} }}");
```