

Cognex Mobile Barcode SDK for WEB

Requirements

Server-side A secure HTTPS connection on the server, which hosts the web application.

Client-side Latest versions of all major browsers: Chrome, Firefox, Safari, Edge, and Opera.

Grant permission for the site to access the camera if necessary.

Installation overview

The cmbWeb release comes in multiple variants:

1. plain/vanilla js - front_end directory of the [cmbWeb download](#)
2. webpack - webpack directory of the cmbWeb download
3. webpack inline - webpack_inline directory of the cmbWeb download
4. npm package - <https://www.npmjs.com/package/cmbSDK-cmbweb>

Main differences between the multiple variants are as follows:

- Plain / Vanilla JS can run directly in the browser. Its suitable for quickly trying things out.
- The webpack and webpack inline variants need to be built (i.e. packaged, bundled) first. Most (but not all) of the files cmbWeb comes with will be bundled, thus the resulting solution will be more compact.
In the case of webpack inline, the index.wasm file is embedded, thus you don't need to worry about loading it.
- The cmbSDK-cmbweb npm package can run directly in the browser, or it can be used as a nodejs module. All of the files cmbWeb comes with are embedded in a single JS file so you only need to use one file for cmbWeb and one more for configuring it.

Difference between using a solution with the index.wasm file or having it embedded:

- Plain / Vanilla JS and webpack have a setWasmPath method in the MWBConfig_wa.js file, where you can specify a different location for the index.wasm file, whereas webpack inline and the npm package embed it, thus they don't have the setWasmPath method.
- Having the index.wasm file embedded means you don't need to worry about loading it (e.g. having the correct 'application/wasm' MIME type on your server, compatibility in different environments, setting the correct path, etc.), but this comes at the cost of a larger size, about 33% more for the embedded format.

The one resource that does not come embedded in any variant is the licensing image; all variants expect to load the licensing image - this can be changed via the mwbScanner.setIcon method already present in the MWBConfig_wa.js file (see *Configuring the license image* bullet in the [Licensing the SDK](#) section).

You can go over the installation details below for each variant.

Installation with plain JS

Place the following files into your web application:

- assets (icons directory)
- barcode-scanner-preview-style.css
- cognex_icon.png
- index.html // Example
- index.js
- index.wasm
- main.js
- MWBConfig_wa.js
- MWBScanner_wa.js
- sdk_modular.js

The `index.html` page is not part of the cmbWEB SDK. It serves only as an example of scripts to include and usage of the exposed API methods.

Include the following scripts on the `index.html` page:

```
<script type="text/javascript" src="MWBScanner_wa.js"></script>
<script type="text/javascript" src="MWBConfig_wa.js"></script>
<script type="text/javascript" src="sdk_modular.js"></script>
<script type="text/javascript" src="main.js"></script>
<script async type="text/javascript" src="index.js"></script>
```

Installation with Webpack

1. Call webpack from the project directory. This builds the source files from `/src` into a `bundle.js` file and places it into the `/dist` directory.
2. The `/src/main.js` file serves as an entry point. It contains an example of how to use the cmbWEB SDK. The other files in the `/src` directory are part of the SDK.

The `/dist` directory contains all the necessary files:

The `index.html` page is not part of the cmbWEB SDK. It serves only as an example of scripts to include and usage of the exposed API methods.

- assets (icons directory)
- cognex_icon.png
- bundle.js // Built with webpack
- index.html // Example
- index.js
- index.wasm

3. Include the following scripts on the page:

```
<script src="bundle.js"></script>
<script src="index.js"></script>
```

Note: `index.js` (global scope) and `index.wasm` have to be present.

The structure of the final solution should be much like the one in the `/dist` directory.

- It's also important where the page, `index.html` in this example, that includes `index.js` and the webpack built file, `bundle.js` in this case, is located **in relation to** `index.wasm` and `cognex_icon.png`.

Make sure that the `index.html`, `index.wasm`, and `cognex_icon.png` files are in the same directory, as the `index.js` and `bundle.js` files expect to find them in the directory of the HTML page.

This is true even if *index.js* and *bundle.js* reside in a different directory - they still expect *index.wasm* and *cognex_icon.png* to be in the directory of the HTML page they are included in.

If you wish to change this you can do so by specifying another path through the `mwbScanner.setIcon` and `setWasmPath` methods respectively.

- The path of *index.wasm* can be set through the return statement of the **setWasmPath** method in *MWBConfig_wa.js*. In the webpack version this method (that is, the return of it) should be used as an argument:

```
MWB.setWasmPath(MWBCfg.setWasmPath());
```

in *main.js* or you could set a custom string here directly:

```
MWB.setWasmPath("example_path/index.wasm");
```

Installation with Webpack (inline wasm)

1. Call webpack from the project directory. This builds the source files from */src* into a *bundle.js* file and places it into the */dist* directory.
2. The *./src/main.js* file serves as an entry point. It contains an example of how to use the cmbWEB SDK. The other files in the */src* directory are part of the SDK.

The */dist* directory contains all the necessary files:

The *index.html* page is not part of the cmbWEB SDK. It serves only as an example of scripts to include and usage of the exposed API methods.

- assets (icons directory)
- cognex_icon.png
- bundle.js //built with webpack
- index.html //example

3. Include the following script on the page:

```
<script src="bundle.js"></script>
```

Note: Unlike the standard webpack version, this version inlines the wasm binary within the js source files, thus *index.js* and *index.wasm* are no longer present.

The structure of the final solution should be much like the one in the */dist* directory.

- It's also important where the page, *index.html* in this example, that includes the webpack built file, *bundle.js* in this case, is located **in relation to cognex_icon.png**.

Make sure that the *index.html* and *cognex_icon.png* files are in the same directory, as the *bundle.js* file expects to find *cognex_icon.png* in the directory of the HTML page.

This is true even if *bundle.js* resides in a different directory - it still expects *cognex_icon.png* to be in the directory of the HTML page it is included in.

If you wish to change this you can do so by specifying another path through the *mwbScanner.setIcon* method.

- As *index.wasm* is removed in this version, the **setWasmPath** method is also removed.

Installation with npm

1. Run *npm i cmbsdk-cmbweb*. This will copy the npm package content to *node_modules/cmbSDK-cmbweb/* and also add the package to your *package.json* if you already have one.
2. The *node_modules/cmbSDK-cmbweb/cmbweb.js* file contains the entire SDK and also serves as an entry point.

The *node_modules/cmbSDK-cmbweb/* directory contains all the necessary files:

The *index.html* page is not part of the cmbWEB SDK. It serves only as an example of scripts to include and usage of the exposed API methods.

- *cmbweb.js*
- *cognex_icon.png*
- *MWBConfig_wa.js*
- *index.html* //example

3. Include the following script on the page:

- For usage directly in browser

```
<script type="text/javascript" src="cmbweb.js"></script>
<script type="text/javascript" src="MWBConfig_wa.js"></script>

//copy the following files to your page's directory:
//cmbweb.js                                //cmbWeb SDK
//MWBConfig_wa.js                        //your custom configuration
//cognex_icon.png                        //your license image (may have a different name)
```

Note: Unlike the vanilla js and webpack versions, this version inlines the wasm binary, the css file and the assets within the js source file.

The structure of the final solution should be much like the one in the */node_modules/cmbSDK-cmbweb/* directory.

- For usage as a nodejs module

```
//in your entry point file:
var mwbScanner = require('cmbsdk-cmbweb');
window.mwbScanner = mwbScanner;

//in your page:
<script src="./bundle.js"></script> <!-- a bundled file for the browser that contains cmbsdk-cmbweb -->
<script type="text/javascript" src="MWBConfig_wa.js"></script>

//copy the following files to your page's directory:
//MWBConfig_wa.js           //your custom configuration
//cognex_icon.png          //your license image (may have a different name)
```

- It's also important where the page, *index.html* in this example, that includes the *cmbweb.js* / *bundle.js* file, is located **in relation to** *cognex_icon.png*.

Make sure that the *index.html* and *cognex_icon.png* files are in the same directory, as the *cmbweb.js* / *bundle.js* file expects to find *cognex_icon.png* in the directory of the HTML page.

This is true even if *cmbweb.js* / *bundle.js* resides in a different directory - it still expects *cognex_icon.png* to be in the directory of the HTML page it is included in.

If you wish to change this you can do so by specifying another path through the *mwbScanner.setIcon* method.

Integration with Blazor (no Blazor UI)

The *cmbsdk-cmbweb* npm package variant of the *cmbWEB* SDK is in fact the most suitable for integration with Blazor. Refer to the [previous section](#) for installation details.

In a Blazor WebAssembly App created from a project template in Visual Studio, you need to do the following to add and use *cmbWEB* in it:

1. Copy the *cmbweb.js*, *MWBConfig_wa.js*, *cognex_icon.png* (and *test_img.png* if you want) files into *wwwroot*.
2. Then in *index.html*:

```
<script src="_framework/blazor.webassembly.js"></script>

//after including blazor.webassembly.js, include the following cmbWEB files:
<script src="./cmbweb.js"></script>
<script type="text/javascript" src="MWBConfig_wa.js"></script>
```

With that, everything is set. Once the browser loads everything, you can start using the API, for example:

```
mwbScanner.startScanning(function(res){ console.log(res); }, 25,25,50,50);
```

on a button click or even from the console, which will start a cameraPreview and start scanning. You can check out the API [here](#).

Note that, this is using cmbWeb side-by-side with Blazor; the UI from cmbWeb is HTML and it does DOM manipulation which is fine as long as you don't intend Blazor to be aware of it, as [stated](#):

Only mutate the Document Object Model (DOM) with JavaScript (JS) when the object doesn't interact with Blazor. Blazor maintains representations of the DOM and interacts directly with DOM objects.

Configuration

The scanner is configured through the *MWBConfig_wa.js* file, which contains the following:

- Subscribing to *scannerModuleLoaded* event, after which the scanner is ready to be used and its methods can be invoked. The scanner load and start up process could take longer on slower devices and slower networks.

```
document.addEventListener("scannerModuleLoaded", function(e) {
    console.log(e.detail); //Prints "Scanner is ready."
    //can use mwbScanner.* methods now
});
```

Note: Attempting to use the API, for example making the scan call before the scanner is ready, meaning that the *scannerModuleLoaded* event has not fired yet, would result in an error.

- Getting available cameras and setting the desired camera. Usually for situations where the device has multiple rear cameras, and the one without auto-focus is used by default.

There is a commented-out example for doing this in the MWBConfig_wa.js file:

```
mwbScanner.getCameras().then(function(foundCameras){
    let cameraCount = foundCameras.length;
    console.log("Cameras found: " + cameraCount);

    //list found cameras with label / name and id
    for (let i = 0; i < cameraCount; i++)
        console.log("Camera " + i + " name: " + foundCameras[i].label + " with ID: " + foundCameras[i].id);

    //use desired camera.id for the param
    let desiredCameraIndex = 0;
    let desiredCameraId = foundCameras[desiredCameraIndex].id;
    mwbScanner.setCamera(desiredCameraId); //overrides the effect of the MWBuseFrontCamera setting

    //can use mwbScanner.startScanning methods now
});
```

Note: When using the **getCameras** method, wait for the operation to complete before using the scanner.

When it comes to choosing which of the cameras to use for situations where the device has multiple back cameras, and the one without auto-focus is used by default, currently there is no indicator which one has auto-focus so you will have to try which one works best. However, there seems to be a common denominator for the back camera with AF - it will typically have a label like "camera2 0, facing back" so the back camera with a "0" in the label is usually the best one.

- Enabling the camera switcher on the cameraPreview. Usually for situations where the device has multiple cameras, the camera switcher UI lists all found cameras, and allows to switch to a different camera. Doing so stops the current scanning and starts a new scanning session with the chosen camera.

The camera switcher is enabled in the MWBConfig_wa.js file with the following setting:

```
{"method" : "MWBenableCameraSwitcher", "value" : [true]}
```

Options for the camera switcher can be set with the following setting:

```
{"method" : "MWBsetCameraSwitcherOptions", "value" : [
    //mw_c.CAMERA_SWITCHER_INIT_ON_START |
    //mw_c.CAMERA_SWITCHER_USE_ON_START |
    //mw_c.CAMERA_SWITCHER_USE_BEST_CAMERA |
    0x0 //for binary-OR syntax purposes
]} //Uncomment wanted options

Default value is 0x0 (no option enabled)
@param[in] cameraSwitcherOptions ORed bit mask of camera switcher options
@n CAMERA_SWITCHER_INIT_ON_START - Init camera switcher before starting a scan
@n CAMERA_SWITCHER_USE_ON_START - Use the first camera from all detected cameras (implies CAMERA_SWITCHER_I
@n CAMERA_SWITCHER_USE_BEST_CAMERA - Try to use the main back camera from multiple back cameras
```

Detailed explanation of the configuration options:

CAMERA_SWITCHER_INIT_ON_START

The camera switcher initialization happens after a startScanning call, i.e. on creating a cameraPreview. The operation for listing all available cameras is async, thus it can be awaited which ensures the camera list is obtained first before proceeding to starting the camera and scanning, otherwise, it would be executed as asynchronous, where the camera and scanning will be started most likely before the camera list is obtained. This option will await the initialization, thus it will take longer until the camera and scanning is started.

CAMERA_SWITCHER_USE_ON_START

By default the MWBuseFrontCamera setting has priority at picking which camera (back or front) to use on the cameraPreview, and will continue to be the case until a specific camera is picked and switched to from the camera switcher UI. This option will use the first camera from the camera list on the start of the cameraPreview, thus the MWBuseFrontCamera setting would no longer have effect. Using CAMERA_SWITCHER_USE_ON_START also implies CAMERA_SWITCHER_INIT_ON_START.

CAMERA_SWITCHER_USE_BEST_CAMERA

[Experimental] Devices with multiple back cameras may provide the multiple back cameras separately rather than as one system, and the wide-angle camera which has no autofocus could be used by default. While there is no official indicator which camera has autofocus, based on observed common denominator in such cases, this option will try to order the multiple found cameras such that the main camera which has autofocus will be listed first and used as default.

- The following is an example for setting an example callback for handling the return from **mwbScanner.startScanning**, when a barcode is found, the scan is canceled or an error has occurred:

```
mwbScanner.setCallback(
    function (result) {
        if (result.type != "Multicode") {
            if (result.type == "Cancel") console.log("No Barcode.");
            else alert(result.type + '\n' + result.code);
        }
        //detailed example for handling result.type == "Multicode"
        //is found in MWBConfig_wa.js
    }
);
```

The result object that is returned to the callback function of **mwbScanner.startScanning** or **mwbScanner.scanImage** has the following format:

```
//if using multicode:
result.code      - empty string
result.type      - string "Multicode"
result.count     - the number of detected barcodes
result.codes     - json array of detected barcodes in the result format below

//if using single code (or single code result.codes array elements in case of multicode):
result.code      - string representation of barcode result
result.parsedCode - string json representation of parsed barcode result (if any)
result.type      - type of barcode detected or 'Cancel' if scanning is canceled
result.bytes     - bytes array of raw barcode result
result.isGS1     - (boolean) barcode is GS1 compliant
result.locationPoints - contains rectangle points p1,p2,p3,p4 with the corresponding x,y
result.imageWidth  - Width of the scanned image
result.imageHeight - Height of the scanned image
result.modulesCountX - Number of modules horizontally
result.modulesCountY - Number of modules vertically
result.moduleSizeX  - Width of a module
result.moduleSizeY  - Height of a module
result.ppm         - Pixels per module

//if an error has occurred related to camera use after calling startScanning
result.code      - string representation of a user-friendly error description
result.type      - string "Error"
result.errorDetails - an object containing error properties (name, message) and various explanations (mozilla,
```

Possible result types other than symbology type for a found barcode:

FUNCTION	RESULT.TYPE	DESCRIPTION
----------	-------------	-------------

FUNCTION	RESULT.TYPE	DESCRIPTION
<code>mwbScanner.startScanning</code>	Error	Indicates that an error has occurred, more information included in result.code and result.errorDetails .
<code>mwbScanner.startScanning</code>	Cancel	Indicates that the previously started scan has been canceled by means of <code>mwbScanner.closeScanner</code> or a timeout.
<code>mwbScanner.startScanning</code>	Multicode	Indicates that there are multiple barcode detections in a single image or frame.
<code>mwbScanner.scanImage</code> and <code>mwbScanner.scanFrame</code>	NoResult	Indicates that no barcode has been found or detected.
<code>mwbScanner.scanImage</code> and <code>mwbScanner.scanFrame</code>	Multicode	Indicates that there are multiple barcode detections in a single image or frame.

- Configuring settings with an array of *key:value* pairs for methods and values for their arguments which are invoked when the scanner is loaded and/or started.

```
var mw_c = mwbScanner.getConstants(),
    settings = [
        {"method" : "MWBsetLevel", "value" : [2]}, //1-5
        {"method" : "MWBsetDecoderTimeout", "value" : [30]} //10-60
    ];
```

Some of the settings listed in the *MWBConfig_wa.js* file are commented-out and can be uncommented to enable them, and their values can be changed.

See all configuration methods and parameters listed under the [Configuration Reference](#) section.

Usage | Exposed API Methods

Once all the modules have loaded and the scanner is ready, meaning that the *scannerModuleLoaded* event has fired, the exposed methods from the *mwbScanner* object can be invoked:

- `mwbScanner.startScanning`** method:

fullscreen scanner:

```
mwbScanner.startScanning()
mwbScanner.startScanning(function callback)
```

partial view scanner:

```
mwbScanner.startScanning(x, y, width, height)
mwbScanner.startScanning(function callback, x, y, width, height)
```

x, y, width, height - rectangle of the view in percentages relative to the screen size

callback - result callback

using a div element as a container:

```
<div id="cmbweb-preview-container"
style="border:1px solid; position:fixed; top:25%; left:25%; width:50%; height:30%; background-color:gray;"></div>
```

If a div element with "cmbweb-preview-container" id is used in the html page, the sdk will detect this and operate in 'container mode'.

The cameraPreview will fill the container, and calling startScanning will start the preview. Using the x,y,w,h args for startScanning has no effect in container mode. The resizePartialScanner method also has no effect in container mode.

A "cameraReady" event is triggered when the camera is fully initialized after calling the startScanning method:

```
document.addEventListener("cameraReady", function(e) {
    console.log(e.detail); //Prints "Camera is ready."
});
```

Note: do not add the event listener for this to scannerConfig(), i.e. do so outside of it.

• mwbScanner.scanImage method:

```
mwbScanner.scanImage(URI)
mwbScanner.scanImage(URI, function callback)
```

URI - the path to the image

callback - result callback

• Examples and remaining **exposed API methods**

```
Scan fullscreen - mwbScanner.startScanning()
Scan in view - mwbScanner.startScanning(5,5,90,50)
Scan image - mwbScanner.scanImage('test_img.png')
Pause/Resume - mwbScanner.togglePauseResume()
Resume - mwbScanner.resumeScanning()
Close - mwbScanner.closeScanner()
Flash - mwbScanner.toggleFlash()
Zoom - mwbScanner.toggleZoom()

Resize partial view - mwbScanner.resizePartialScanner(25,25,50,50)
```

- **mwScanner.scanFrame** method

```
mwScanner.scanFrame(ImageData | dataURL)
mwScanner.scanFrame(ImageData | dataURL, function callback)
```

ImageData | dataURL - either an object of type ImageData OR a dataURL image string

callback - result callback

*Note: Using **ImageData** is convenient and faster, because the data is already in the desired format. The only requirement is that the data remains unchanged during the scan, until it is copied to memory.*

*Note: Using **dataURL** is slower and not recommended for live scanning scenarios. The actual data is represented in base64 in an appropriate encoding/compressing format, for example, png or jpeg. It requires extra decoding steps to get the raw pixel format, which takes significant time.*

Note: `MW_methods.helpers.reset_Decoder()`; and `scannerConfig()`; are called on each start of this method. However, they can be unnecessary, if scanning rects, which determine the viewfinder area the decoder scans for each symbology with values 0-100%, are not modified at any other point than the initial config.

Licensing the SDK

Licensing is done by using an image that contains the licensing information. You can see such an image file as the license key itself.

- **Configuring the license image (path)name:**

The cmbWEB SDK sample comes with a license image file *cognex_icon.png*, which does not contain a license, it only serves as an example. The actual license image file can have a different name and/or path, for example, *carrier_icon.png*.

Use **mwScanner.setIcon()** to specify the license image name and path in the *MWBConfig_wa.js* file by including the path and image name as an argument.

```
mwScanner.setIcon("carrier_icon.png");
```

The image file *carrier_icon.png* is expected to be in the same directory as the *index.html* file and the *.js* file that calls the **setIcon** method.

Using a path works in relation to the *index.html* file in this example, or the root of the server in case of /path

The setIcon method can be overloaded with two arguments.

```
mwbScanner.setIcon(iconURI, allowCrossOrigin);
```

The setIcon method now also accepts a 2nd argument (boolean) which if set to true allows cross origin, otherwise if set to false or not used at all it keeps the default (no cross origin).

```
mwbScanner.setIcon("carrier_icon.png", true);
```

This would enable the option of using a license image that is hosted on another server i.e. originates from another domain than the one the web app is hosted on, such as a CDN for example.

Note: The server hosting the licensing image also needs to be configured to allow CORS for it.

- License status (if invalid, expired or invalid domain) will be provided in e.detail of the scannerModuleLoaded event, see *Subscribing to scannerModuleLoaded event* bullet in the [Configuration](#) section.

```
//Possible values:  
"Scanner is ready." //(license is valid)  
"Invalid domain."  
"Invalid license."  
"Expired license."
```

Regardless of string value the scanner is ready once the scannerModuleEvent has fired (results will be masked if not licensed).

• Obtaining and configuring a license

Request a license on <https://cmbdn.cognex.com/lpr>.

After clicking **request license** and selecting CMBWEB/Wasm License, the site navigates to the WASM License Purchase Request page where a license can be created by generating a small image from the provided information, such as:

- the type of license evaluation / commercial / non-production / production
- the licensing period – usually preset by type
- domain(s) / host(s) the web sdk will run on. Instead of a regular domain, a wildcard host or an IP address can be used in specific use cases.

Note: A custom logo can be uploaded to be used instead of the default Cognex logo.

The generated image serves as a carrier for the licensing information.

Note: Unless otherwise specified, browsers cache these resources, and upon an eventual license upgrade you will need to ensure you're getting the new image (and not the cached one). The simplest way is to use a different name, possibly with a version number.

Sample app

Each variant comes with a sampleApp directory, which makes use of the cmbWeb SDK.

Files specific to the sample app are:

- index.html
- sample_app_UI.js
- sample-app-style.css
- icon-sprite.svg
- MWBConfig_wa.js

There are slight differences in what *index.html* includes and in how *MWBConfig_wa.js* carries out the configuration, specific to each variant, but other than that, the content of the sample app is identical for all.

Note that if you are using webpack or webpack inline, you'll need to replace the *src/MWBConfig_wa.js* file with *sampleApp/MWBConfig_wa.js* and re-build so that the sample app's config file is used in the built *bundle.js* file.

Also note that the *MWBConfig_wa.js* file uses the optional chaining operator (*?.*) which has recently gained support in all browsers, see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining#browser_compatibility_

When it comes to what most will be of interest for utilizing cmbWeb, you can check out the following files:

- index.html - Has static UI buttons and basic API usage
- sample_app_UI.js - Dynamically fills the page with UI elements for configuring the scanner
- MWBConfig_wa.js - Makes use of stored configuration values set by UI elements (see sample_app_UI.js)

For taking a closer look at the *sample_app_UI.js* file, you can skip the *GUI_helper* object and check out the event listener method for the "scannerModuleLoaded" event, as well as the *add_gui_controls* method.

Configuration Reference

The following methods are meant to be used in a settings array in *MWBConfig_wa.js*:

```
@name "MWBSetActiveCodes" Sets active or inactive status of decoder types
@param[in] activeCodes ORed bit flags (MWB_CODE_MASK_...) of decoder types to be activated.
```

```
@n MWB_CODE_MASK_NONE
@n MWB_CODE_MASK_QR
@n MWB_CODE_MASK_DM
@n MWB_CODE_MASK_RSS
@n MWB_CODE_MASK_39
@n MWB_CODE_MASK_EANUPC
@n MWB_CODE_MASK_128
@n MWB_CODE_MASK_PDF
@n MWB_CODE_MASK_AZTEC
@n MWB_CODE_MASK_25
@n MWB_CODE_MASK_93
@n MWB_CODE_MASK_CODABAR
@n MWB_CODE_MASK_DOTCODE
@n MWB_CODE_MASK_11
@n MWB_CODE_MASK_MSI
@n MWB_CODE_MASK_MAXICODE
@n MWB_CODE_MASK_POSTAL
@n MWB_CODE_MASK_TELEPEN
@n MWB_CODE_MASK_ALL
```

CodeMask constants are available for all codeMask variables

```
@name "MWBSetActiveSubcodes" Set active subcodes for given code group flag. Subcodes under some decoder type are all acti
```

```
@param[in] codeMask Single decoder type/group (MWB_CODE_MASK_...)
@param[in] subMask ORed bit flags of requested decoder subtypes (MWB_SUBC_MASK_)
```

```
@name "MWBsetFlags" Sets active or inactive status of decoder types
@param[in] codeMask Single decoder type (MWB_CODE_MASK_...)
@param[in] flags ORed bit mask of selected decoder type options (MWB_FLAG_...)
```

```

@name "MWBsetMinLength" configures minimum result length for decoder type specified in codeMask.
@param[in]   codeMask   Single decoder type (MWB_CODE_MASK_...)
@param[in]   minLength  Minimum result length for selected decoder type

@name "MWBsetDirection"
@param[in]   direction  ORed bit mask of direction modes given with MWB_SCANDIRECTION_... bit-masks
@n         MWB_SCANDIRECTION_HORIZONTAL - horizontal lines
@n         MWB_SCANDIRECTION_VERTICAL   - vertical lines
@n         MWB_SCANDIRECTION_OMNI       - omnidirectional lines
@n         MWB_SCANDIRECTION_AUTODETECT - enables BarcodeScanners autodetection of barcode direction
@n         MWB_SCANDIRECTION_CUSTOM     - custom barcode direction

@name "MWBsetScanningRect"
Sets the scanning rectangle
Parameters are interpreted as percentage of image dimensions, i.e. ranges are 0 - 100 for all parameters.
@param[in]   codeMask   Single decoder type selector (MWB_CODE_MASK_...)
@param[in]   left        X coordinate of left edge (percentage)
@param[in]   top         Y coordinate of top edge (percentage)
@param[in]   width       Rectangle width (x axis) (percentage)
@param[in]   height      Rectangle height (y axis) (percentage)

@name "MWBsetLevel"
Effort level of the scanner values can be
@param[in]   level       1,2,3,4 and 5
example : [{"method" : "MWBsetLevel", "value" : [3]}]

@name "MWBsetOverlayMode"
@param[in]   OverlayMode
@n OverlayModeNone      No overlay is displayed
@n OverlayModeMW        Use MW Dynamic Viewfinder with blinking line
@n OverlayModeImage     Show image on top of camera preview
example : [{"method" : "MWBsetOverlayMode", "value" : [mw_c.OverlayModeImage]}]

@name "MWBresizePartialScanner"
Resizes partial scanner dimensions. If usePartialScanner is true the scanner will open in a window with these dimensions
@param[in]   left        X coordinate of left edge (percentage)
@param[in]   top         Y coordinate of top edge (percentage)
@param[in]   width       Rectangle width (x axis) (percentage)
@param[in]   height      Rectangle height (y axis) (percentage)
example : [{"method" : "MWBresizePartialScanner", "value" : [0,0,50,50]}]

@name "MWBusePartialScanner"
Boolean value that opens a partial scanner if set true
@param[in]   bool         true/false
example : [{"method" : "MWBusePartialScanner", "value" : [true]}]

@name "MWBsetActiveParser"
Set active parser types
@param[in]   ActiveParser  ORed values
@n         MWP_PARSER_MASK_NONE
@n         MWP_PARSER_MASK_AUTO
@n         MWP_PARSER_MASK_GS1
@n         MWP_PARSER_MASK_IUID
@n         MWP_PARSER_MASK_ISBT
@n         MWP_PARSER_MASK_AAMVA
@n         MWP_PARSER_MASK_HIBC
@n         MWP_PARSER_MASK_SCM
example : [{"method" : "MWBsetActiveParser", "value" : [mw_c.MWP_PARSER_MASK_GS1 | mw_c.MWP_PARSER_MASK_IUID]}]

//additional settings:

@name "MWBsetBlinkingLineVisible"
Set blinking line visible
Default value is true
@param[in]   visible
example : [{"method" : "MWBsetBlinkingLineVisible", "value" : [true]}]

@name "MWBsetPauseMode"
What happens when the scanner is paused
Default value is PM_PAUSE
@param[in]   pauseMode
@n         PM_NONE          - Nothing happens
@n         PM_PAUSE         - Blinking lines are replaced with a pause view
@n         PM_STOP_BLINKING - Blinking lines stop blinking
example : [{"method" : "MWBsetPauseMode", "value" : [mw_c.PM_STOP_BLINKING]}]

```

```

@name "MwBenableHiRes"
Enable or disable high resolution scanning. It is recommended to enable it when target barcodes are of high density or small
Accepts boolean (false - 480p, true - 720p) or constants (mw_c.CamRes_SD - 480p, mw_c.CamRes_HD - 720p, mw_c.CamRes_FHD - 1080p)
Default value is true (enabled) / CamRes_HD (720p)
@param[in] enableHiRes
example : [{"method" : "MwBenableHiRes", "value" : [true]}]

@name "MwBenableFlash"
Enable or disable flash toggle button on scanning screen. If device does not support flash mode button will be hidden regardless of param.
Default value is true (enabled)
@param[in] enableFlash
example : [{"method" : "MwBenableFlash", "value" : [true]}]

@name "MwBturnFlashOn"
Set default state of flash (torch) when scanner activity is started
Default value is false (disabled)
@param[in] flashOn
example : [{"method" : "MwBturnFlashOn", "value" : [false]}]

@name "MwBenableZoom"
Enable or disable zoom button on scanning screen. If device does not support zoom, button will be hidden regardless of param.
Default value is true (enabled)
@param[in] enableZoom
example : [{"method" : "MwBenableZoom", "value" : [true]}]

@name "MwBsetZoomLevel"
Set desired initial zoom level. Zoom is supported only by chrome. Initial zoom level can be 0 - no zoom, 1 - 50% or 2 - max
@param[in] zoomLevel 0,1 and 2
example : [{"method" : "MwBsetZoomLevel", "value" : [1]}]

@name "MwBenableClose"
Enable or disable close button on cameraPreview (partial view). If cameraPreview is in full screen mode, button will be present regardless of param.
Default value is false (disabled)
@param[in] enableClose
example : [{"method" : "MwBenableClose", "value" : [true]}]

@name "MwBcloseScannerOnDecode"
Enable/disable continuous scanning. If 'shouldClose' is 'false', result callback will be performed and scanner will be paused
@param[in] shouldClose
example : [{"method" : "MwBcloseScannerOnDecode", "value" : [true]}]

@name "MwBsetParam"
Set custom decoder param id / value pair for decoder type specified in a codeMask.
@param[in] codeMask Single decoder type (MwB_CODE_MASK_...)
@param[in] paramId ID of param
@param[in] paramValue Integer value of param
example : [{"method" : "MwBsetParam", "value" : [codeMask, paramId, paramValue]}]

@name "MwBduplicateCodeDelay"
Ignore result if scanned the same code in continuous scanning mode
@param[in] delay Time interval between 2 scan results with the same result.code in milliseconds
example : [{"method" : "MwBduplicateCodeDelay", "value" : [1000]}]

@name "MwBuseFrontCamera"
Use front facing camera
@param[in] useFrontCamera Whether or not to use front facing camera
Default value is false
example : [{"method" : "MwBuseFrontCamera", "value" : [false]}]

@name "MwBsetDecoderTimeout"
The amount of time in seconds the camera preview in either partial view or full screen mode will be on before closeScanner
@param[in] timeout 10-60, 0 to disable
example : [{"method" : "MwBsetDecoderTimeout", "value" : [30]}]

@name "MwBsetDpsLimit"
Decodes per second, i.e. number of frames that are sent for decoding in a given second. This setting only works with our camera
@param[in] dpsLimit 1-30
example : [{"method" : "MwBsetDpsLimit", "value" : [2]}]

```

Most of the additional settings configure only the implementation of the camera preview. In the case of using a custom preview or camera feed with the **scanFrame** method, it is up to your implementation to

limit the number of frames that are decoded in a given second, meaning the number of calls to **scanFrame**. This affects performance, especially on iOS.

Performance and Browser Support

As described in previous sections, WebAssembly makes CPU intensive execution at near-native speeds available on the client-side. This means that the performance of the device in question can be limited by its hardware capabilities. In general, the decoding speed is faster on desktop computers and slower on mobile devices. Browser support for various features tends to favor desktop versions as well.

Currently, the best browser support is provided by Google Chrome. Other browsers do well in general, with some known differences at this time:

- Firefox offers camera choice regardless of whether the front/back camera was specified. Firefox also has no flash/zoom API support.
- Safari requires an additional user click to start the camera/video for the first time and does not keep the choice for future page access. Safari also has no flash/zoom API support.
 - All execution on Safari on iOS is on the main thread. This happens because the decoder is the part that uses the CPU the most, it can compete with the camera preview, and the user would end up with a lot of lag.

You can improve Safari iOS performance in the following ways:

- Use of **dpsLimit** value decodes per second, meaning the number of frames that are sent for decoding in a given second, is recommended, such as 1 or 2 at the most. In realistic use cases, there is no practical need for a higher decoding rate, even on devices that are capable of such performance.
- Limit symbologies to only the ones needed. With single symbology decoding, for example just QR code decoding, there might be no lag and stutter noticeable, even though it is on the same thread.
- Using a lower effort level for the decoder, and/or a lower camera resolution can also help reduce lag and stutter produced by CPU overuse.

Example configuration for improving performance on iOS Safari:

```
{ "method" : "MWBsetDpsLimit", "value" : [1]},  
{ "method" : "MWBsetActiveCodes", "value" : [ mw_c.MWB_CODE_MASK_QR ]},  
{ "method" : "MWBsetLevel", "value" : [1]},  
{ "method" : "MWBenableHiRes", "value" : [false]}
```

Note: Some codes like PDF require a higher effort level and can have detail which cannot be captured with lower camera resolution. Consider these capture limitations, when choosing between 480p, 720p (default) or 1080p for demanding barcodes.

The APIs that different browsers provide may differ in other ways, however, we aim to mitigate differences as much as possible to provide a consistent experience across different platforms.

Web vs native

When it comes to using a web or a native SDK, both can scan with similar performance but a native solution (android or iOS SDK) is slightly better because natively there is a slightly better camera focus, and that affects the scan the most.

For the most part, this slight difference in-camera focus won't be an issue, but, depending on your use case, such as cases where your code has a high density, it might take more time to scan compared to a native solution.

In such demanding cases, you might do fine with either 720p or 1080p resolution, but your camera must have auto-focus otherwise you'll have a hard time getting a clear frame and a successful scan. If not, then maybe a better barcode sample with a larger size might offset the lack of camera quality.

In general, while higher-end mobile devices are better, there is no limit to how low-end you can go with a particular mobile device, again, as long as it has a good enough camera with autofocus (of course you should test for your use case to see how suitable it is).