Toshiba Embedded Signage

Documentation

Version	1.5.0
Status	Final
Owner	Toshiba Europe GmbH
Privacy	Confidential – Distribution under NDA only
File name	ToshibaEmbeddedSignage-Documentation-1.5.0
Last page	49

Version history

Version	Date	Description/Changes	Writers
1.0	30/03/2016	First version	Giovanni Conz
1.1	02/06/2016	Updated Download Manager and Storage Manager	Giovanni Conz
1.2	18/08/2016	Updated Download Manager and Storage Manager, as well as other minor APIs; updated details of basic operations with monitor	Giovanni Conz
1.3	13/01/2017	Updated Storage Manager (e.g., added unzip API) and Custom Download Manager as well as other minor APIs. Added Screenshot API	Giovanni Conz
1.3.1	14/01/2017	Minor updates on custom APIs	Giovanni Conz
1.4.0	18/04/2017	Development guidelines	Giovanni Conz
1.4.1	28/04/2017	App Cache	Giovanni Conz
1.4.2	19/05/2017	Added RS232 document reference	Giovanni Conz
1.4.3	20/05/2017	Fixed StorageManager getPlayableFile	Giovanni Conz
1.4.4	13/06/2017	Garbage Collector API and memory statistics	Giovanni Conz
1.4.5	03/07/2017	Fixed copy/paste issue for Storage Manager APIs	Giovanni Conz
1.4.6	01/09/2017	Added Videowall sync and offset guidelines	Giovanni Conz
1.4.7	12/09/2017	Added video transition guidelines	Giovanni Conz
1.4.8	24/01/2018	Added more guidelines: default NTP, 24h RST recommendation, SETSTARTURL instructions, Service Workers	Giovanni Conz
1.4.9	21/03/2018	Video and Audio supported codec	Giovanni Conz
1.5.0	10/07/2018	FileManager API	Giovanni Conz

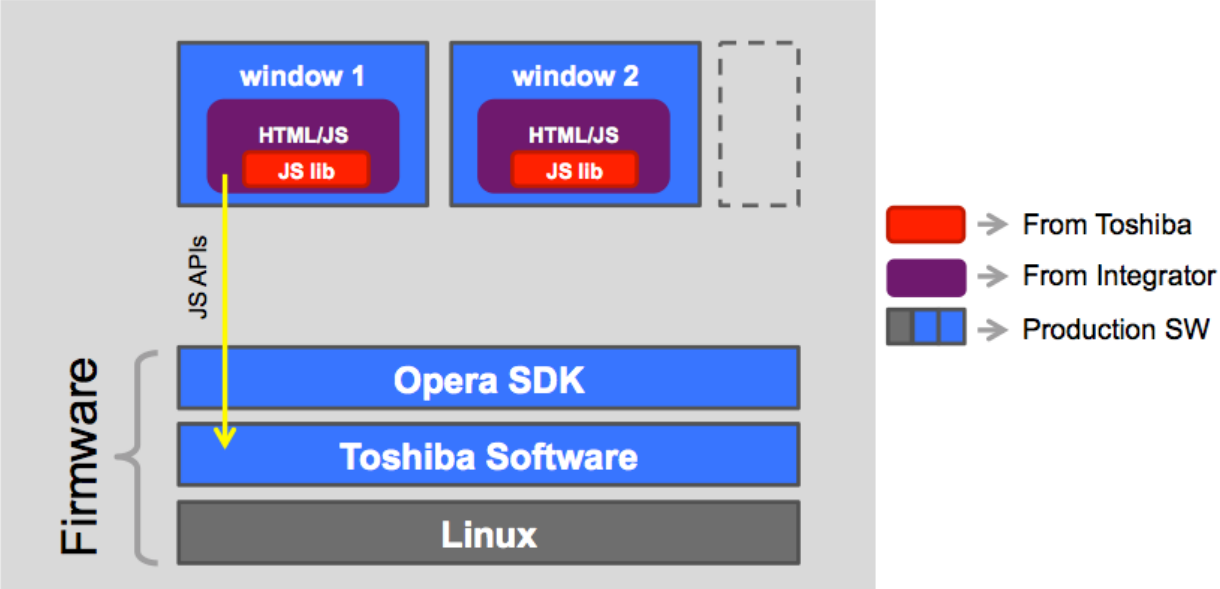
Table of contents

1. Introduction

This document presents Toshiba Europe GmbH (TEG)'s documentation of Embedded Signage, a platform that allows any third party to develop a browser based software for Digital Signage solutions.

Toshiba provides to system integrators and software providers a zip "tools" package including demos and utilities built in HTML/CSS/JS so that they could port or build a software running on the board embedded in the B2B monitors.

The overall architecture looks like the following diagram:



In order to overcome technological limits of in-browser developments with HTML5, there is the need to enable certain APIs to access local resources. This is achieved by some custom APIs: there are some special JS objects available in the browser of the monitor, that when called will trigger some native functions on the monitor. A big part of those APIs specifications were based on the OIPF specification v2.3 24/01/2014, Volume 5 – Declarative Application Environment and the HBBTV 1.5 APIs while others – not covered by open standards, are defined by Toshiba.

For OIPF specifications, please refer to:
http://www.oipf.tv/docs/OIPF-T1-R2_Specification-Volume-5-Declarative-Application-Environment-v2_3-2014-01-24.pdf

For HBBTV specifications, please refer to:
https://www.hbbtv.org/wp-content/uploads/2015/07/HbbTV-specification-1-5_Aug2012.pdf

In this document the details of the APIs are described; in order to simplify the management, Toshiba will also provide some sample JS libraries, with full documentation, that will perform common operations that are usually done using the APIs listed in this document. Therefore, for some of them (e.g., Application Manager) there is no need to use the low level APIs – it's possible to use a higher level management library, or start from it and customize it to fit the specific needs without developing such a service module from scratch.

2. Development tools

Toshiba provides a set of additional documentation, examples, demos, tutorials to ease the development of an application compliant and optimized for the Embedded Signage platform. Once the provided zip file is unpacked, head to the “docs” folder and open the “1.README.md” and the “2.HOWTO.md” files with a text editor; you will find an overview on the package content, as well as detailed instructions on how to install and run the tools for any platform.

Once started, you will be able to browse through the additional documentation, the demos, utility modules, etc. through a browser.

All the examples, utility modules and demos are provided with full plain source code and specific comments.

2.1. Basic monitor operations for development

The Opera SDK embedded is V4.6, which has Chromium M44 in its core.

To run a test application, it’s needed a sample development monitor. The development firmware onboard the monitor will allow easy debug and deploy operations, while production firmware – while having the same performances and APIs, won’t have open channels for debug.

Once the monitor is setup, turned on and connected to the local network, it can be reached through two special channels from a PC in the same local network: a remote console for the monitor and remote browser debug.
The remote console can be reached with telnet at port 1986:

```
telnet <monitorip> 1986
```

This console is useful to send to the monitor specific, low level commands; the most important one for embedded development is *OPENURL*, which will open a new browser window with a specific URL. In the console opened through telnet, type:

```
OPENURL http://www.bbc.com/
```

and hit return; the monitor will render the HTML page downloaded at that URL. If an SSL page is opened, the monitor might show an SSL error in case the date-time is not set correctly (in case it’s not, you can simply set the right time with the API **`setCurrentDateTime`** called from the browser debug console).

The remote browser debug works through the standard Chromium developer tools; developers can use all the debug tools normally used for local web development remotely on the device. To start it, it’s enough to place the monitor IP address with the port 4725 with a Chrome/Chromium M44 browser, putting in the address bar an URL like

```
http://<monitorip>:4725
```

This will display a basic HTML page listing all the windows currently opened. In case there is none, just run the console command OPENURL to open one and refresh the page. Clicking on the link of the page that will be displayed, the Chrome/Chromium developer tools will be attached to the remote monitor, with the ability to use all the standard tools available in the normal console/developer tool.

2.2. Browser requirements

In order to use properly the browser developer tools, it’s recommended to use a Chromium version 44 on the PC performing the remote debug, because it’s the same version of the one onboard the monitor. With latest Chrome releases, the debug tools changed a lot and are not retro-compatible any more from version 50 – therefore, a JS error would be shown in the console trying to connect with the latest Chrome version.

Furthermore, Chromium 44 is mandatory for emulating the app's behavior on a local PC. Unfortunately, the current available versions of Chromium 44 for Windows and Linux downloadable from Internet don't have codecs enabled, therefore video playout is not working locally. About this issue, Toshiba can provide a customized version of Chromium 44 that allows to have a development and emulation environment totally suitable with the monitor features.

3. Standard APIs specifications

Referring to the OIPF/HbbTV APIs, this is the list of the objects that might be of interest developing an Embedded Signage solution:

Object Name	Status	Notes
ObjectFactory Class	Available	
Video/broadcast Object	Available	
A/V Control Object	Available	
Configuration Class	Partially Implemented	Some attributes are not implemented
Disc Info Class	Available	

All the details of the implemented objects can be found in the OIPF and HBBTV standard APIs reference mentioned in the introduction.

4. Custom APIs specifications

The list of custom plugins available is as follows. The detailed behavior if the APIs can be checked looking at the "**API check**" demo in the tools package.

Object Name	Status	Description
ApplicationManager	Available	Manage browser windows
Network Manager	Available	Manage network interfaces
Date Time Settings	Available	Manage DateTime
Input Manager	Available	Set/Get current AV Input, Set the video position to the specified coordinates etc.
StandbyController	Available	Enter/Quit/Query ActiveStandby
Storage Manager	Available	get Removable Device information, List/Format available storages
General Settings	Available	Set/Get VolumeLevel, Query active standby, etc.
MouseDevice	Available	Checks whether mouse attached, activates and deactivates mouse
DatabaseManager	Available	Manage data on lookup table of corresponding key.
CustomDownloadManager	Available	Downloads file from the given URL address.

RemoteKeyGenerator	Available	Generates a remote key press event from browser.
Rs232LanCommand	Available	Sends RS232/LAN command to the platform.
MacId	Available	Returns mac address to identify device
Screenshot	Available	Takes monitor's screenshot and save it to a path

They are described in the next sections by functional areas.

4.1. Application Manager

Application Manager object allows to manage new/existing windows from within the HTML/JS environment. Once in a window started by default, the `ApplicationManager` JS object is available to manage windows.

This is shown and described in detail in the demo called "**Main App in Application Manager demo**" in the tools package.

The IDLs used for the Application are the following:

```
///
  ///
  interface KeySet {
      ///
      /// Sets the value of the keyset which this DAE application requests to receive. It also
reserves the back key to the application when included in other_keys.
      /// @param value Value of the keyset.
     /// Supported values are: RED: 0x1, GREEN: 0x2, YELLOW: 0x4, BLUE: 0x8,
     /// NAVIGATION (VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_ENTER): 0x10,
     /// VCR (VK_PLAY, VK_PAUSE, VK_STOP, VK_NEXT, VK_PREV, VK_FAST_FWD, VK_REWIND, VK_PLAY_PAUSE):
0x20,
     /// SCROLL (VK_PAGE_UP, VK_PAGE_DOWN): 0x40, INFO: 0x80, NUMERIC: 0x100,
     /// ALPHABETIC: 0x200, OTHER (other_keys parameter): 0x400
      /// @param other_keys If the OTHER bit in the value property is set then this indicates those
key events
which are available to the browser which are not included in one of the constants defined in this
class,
      /// If the OTHER bit in the value property is not set then this property is meaningless.
     /// The string is expected to be a comma separated list.
     /// The string values are: VK_UNDEFINED : 0, VK_CANCEL : 3, VK_BACK_SPACE : 8, VK_TAB : 9,
VK_CLEAR : 12, VK_ENTER : 13, VK_SHIFT : 16, VK_CONTROL : 17, VK_ALT : 18, VK_PAUSE : 19,
VK_CAPS_LOCK : 20, VK_KANA : 21, VK_FINAL : 24, VK_KANJI : 25, VK_ESCAPE : 27, VK_CONVERT : 28,
VK_NONCONVERT : 29, VK_ACCEPT : 30, VK_MODECHANGE : 31, VK_SPACE : 32, VK_PAGE_UP : 33, VK_PAGE_DOWN
: 34, VK_END : 35, VK_HOME : 36, VK_LEFT : 37, VK_UP : 38, VK_RIGHT : 39, VK_DOWN : 40, VK_COMMA :
44, VK_PERIOD : 46, VK_SLASH : 47, VK_0 : 48, VK_1 : 49, VK_2 : 50, VK_3 : 51, VK_4 : 52, VK_5 : 53,
VK_6 : 54, VK_7 : 55, VK_8 : 56,  VK_9 : 57, VK_SEMICOLON : 59, VK_EQUALS : 61, VK_A : 65, VK_B :
66, VK_C : 67, VK_D : 68, VK_E : 69, VK_F : 70, VK_G : 71, VK_H : 72, VK_I : 73, VK_J : 74, VK_K :
75, VK_L : 76, VK_M : 77, VK_N : 78, VK_O : 79, VK_P : 80, VK_Q : 81, VK_R : 82, VK_S : 83, VK_T :
84, VK_U : 85, VK_V : 86, VK_W : 87, VK_X : 88, VK_Y : 89, VK_Z : 90, VK_OPEN_BRACKET : 91,
VK_BACK_SLASH : 92, VK_CLOSE_BRACKET : 93, VK_NUMPAD0 : 96, VK_NUMPAD1 : 97, VK_NUMPAD2 : 98,
VK_NUMPAD3 : 99, VK_NUMPAD4 : 100, VK_NUMPAD5 : 101, VK_NUMPAD6 : 102, VK_NUMPAD7 : 103, VK_NUMPAD8
: 104, VK_NUMPAD9 : 105, VK_MULTIPLY : 106, VK_ADD : 107, VK_SEPARATER : 108, VK_SUBTRACT : 109,
VK_DECIMAL : 110, VK_DIVIDE : 111, VK_F1 : 112, VK_F2 : 113, VK_F3 : 114, VK_F4 : 115, VK_F5 : 116,
VK_F6 : 117, VK_F7 : 118, VK_F8 : 119, VK_F9 : 120, VK_F10 : 121, VK_F11 : 122, VK_F12 : 123,
VK_DELETE : 127, VK_NUM_LOCK : 144, VK_SCROLL_LOCK : 145, VK_PRINTSCREEN : 154, VK_INSERT : 155,
VK_HELP : 156, VK_META : 157, VK_BACK_QUOTE : 192, VK_QUOTE : 222, VK_RED : 403, VK_GREEN : 404,
VK_YELLOW : 502, VK_BLUE : 406, VK_GREY : 407, VK_BROWN : 408, VK_POWER : 409, VK_DIMMER : 410,
VK_WINK : 411, VK_REWIND : 412, VK_STOP : 413, VK_EJECT_TOGGLE : 414, VK_PLAY : 415, VK_RECORD :
416, VK_FAST_FWD : 473, VK_PLAY_SPEED_UP : 418, VK_PLAY_SPEED_DOWN : 419, VK_PLAY_SPEED_RESET : 420,
VK_RECORD_SPEED_NEXT : 421, VK_GO_TO_START : 422, VK_GO_TO_END : 423, VK_TRACK_PREV : 424,
VK_TRACK_NEXT : 425, VK_RANDOM_TOGGLE : 426, VK_CHANNEL_UP : 427, VK_CHANNEL_DOWN : 428,
VK_STORE_FAVORITE_0 : 429, VK_STORE_FAVORITE_1 : 430, VK_STORE_FAVORITE_2 : 431, VK_STORE_FAVORITE_3
: 432, VK_RECALL_FAVORITE_0 : 433, VK_RECALL_FAVORITE_1 : 434, VK_RECALL_FAVORITE_2 : 435,
VK_RECALL_FAVORITE_3 : 436, VK_CLEAR_FAVORITE_0 : 437, VK_CLEAR_FAVORITE_1 : 438,
VK_CLEAR_FAVORITE_2 : 439, VK_CLEAR_FAVORITE_3 : 440, VK_SCAN_CHANNELS_TOGGLE : 441, VK_PINP_TOGGLE
: 442, VK_SPLIT_SCREEN_TOGGLE : 443, VK_DISPLAY_SWAP : 444, VK_SCREEN_MODE_NEXT : 445,
VK_VIDEO_MODE_NEXT : 446, VK_VOLUME_UP : 447, VK_VOLUME_DOWN : 448, VK_MUTE : 449,
```

```
VK_SURROUND_MODE_NEXT : 450, VK_BALANCE_RIGHT : 451, VK_BALANCE_LEFT : 452, VK_FADER_FRONT : 453,
VK_FADER_REAR : 454, VK_BASS_BOOST_UP : 455, VK_BASS_BOOST_DOWN : 456, VK_INFO : 457, VK_GUIDE :
458, VK_TELETEXT : 459, VK_SUBTITLE : 460, VK_BACK : 461, VK_MENU : 462, VK_PLAY_PAUSE : 463
      ///
      void setValue([in] Integer value, [in] String other_keys);

      ///
      /// The value of the keyset.
      ///
      readonly attribute Integer value;

      ///
      /// The other keys.
      ///
      readonly attribute Integer[] other_keys;

    }

    ///
    ///
    interface Window {
        ///
        /// Load a webpage or application from the specified URL.
        /// @param uri Url to load.
        ///
        void loadUrl([in] String uri);

        ///
        /// Post message to the listener which is a Window instance.
        /// @param msg Message that is sent to target window.
        /// @param target_window Target window
        ///
        void postMessage([in] String msg, [in] Window target_window);

        ///
        /// Make the window active.
        ///
        void activate();

        ///
        /// Make the window invisible.
        ///
        void hide();

        ///
        /// Make the window visible.
        ///
        void show();

        ///
        /// Make the window deactivated.
        ///
        void deactivate();

        ///
        /// Move the window to the front of the z-order.
        ///
        void bringToFront();

        ///
        /// Move the window to the back of the z-order.
        ///
        void sendToBack();

        ///
        /// Places a window below another window in the window stack.
        /// @param reference_window The handle of the reference window.
        ///
        void moveBelow([in] Window reference_window);

        ///
        /// Places a window above another window in the window stack.
        /// @param reference_window The handle of the reference window.
        ///
        void moveAbove([in] Window reference_window);
```

```
        ///
        /// Event raised when message has been received
        ///
        interface message : EventListener {
            String msg; /// Message that has been received.
        }

        ///
        /// Adds DOM2 event listener for specified event.
        ///
        /// Possible values are:
        /// event_name: "message", listener: function that has arguments similar to the message
interface properties.
        ///
        /// @param event_name Name of the event.
        /// @param listener Javascript function that has arguments like the properties listed in the
listener's interface.
        ///
        void addEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes single DOM2 event listener for specified event
        ///
        /// Possible values are:
        /// event_name: "message", listener: function that has arguments similar to the message
interface properties.
        ///
        /// @param event_name Name of the event
        /// @param listener Named javascript function that has arguments like the properties listed
in the listener's interface.
        ///
        void removeEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes all DOM2 event listeners for specified event
        ///
        /// @param event_name Name of the event
        /// @see removeEventListener([in] String event_name, [in] EventListener listener);
        ///
        void removeEventListener([in] String event_name);

        ///
        /// The active status
        ///
        readonly attribute Boolean active;

        ///
        /// The visibility of the window
        ///
        readonly attribute Boolean visible;

        ///
        /// The event dispatch priority of the window.
        ///
        attribute Integer event_dispatch_priority;

        ///
        /// Name of the window
        ///
        readonly attribute String name;

        ///
        /// Current url loaded in the window
        ///
        readonly attribute String url;

        ///
        /// The object representing the user input events sent to the DAE application.
        ///
        readonly attribute KeySet keySet;

    }

    ///
    /// This is a class that implements global ApplicationManager object.
    ///
```

```
    interface ApplicationManager {
        ///
        /// Creates a full-screen generic window.
        /// @param name Name of the window
        /// @param event_dispatch_priority The event dispatch priority of the window.
        /// @return Newly created window.
        ///
        [out] Window createWindow([in] String name, [in] Integer event_dispatch_priority);

        ///
        /// Gets full-screen CSP window. Note that destroyWindow must be called at JS level once it
is closed to keep track of JSPP window list properly.
        /// @return CSP window.
        ///
        [out] Window getCSPWindow();

        ///
        /// Returns the active window.
        /// @return The current active window.
        ///
        [out] Window getActiveWindow();

        ///
        /// Gets the window of the specified name
        /// @param name Name of the window.
        /// @return window of the specified name; null if it does not exist.
        ///
        [out] Window getWindowByName([in] String name);

        ///
        /// Returns window list
        /// @return The current window list
        ///
        [out] Window[] getWindows();

        ///
        /// Destroys a specified window and releases any reserved resources.
        /// @param window Window object.
        ///
        void destroyWindow([in] Window window);


       ///
       /// Event raised when a change in application state of any window occurs. Currently only
terminated event is raised.
       ///
       interface appStateChangeEvent : EventListener {
              Integer    state;    ///    Current    application    state    [possible    value:
ApplicationManager.TERMINATED]
              String name; /// Name of the window
       }

       ///
       /// Adds DOM2 event listener for specified event.
       ///
       /// Possible values are:
       /// event_name: "appStateChangeEvent", listener: function that has arguments similar to the
appStateChangeEvent interface properties.
       ///
       /// @param event_name Name of the event.
       /// @param listener Javascript function that has arguments like the properties listed in the
listener's interface.
       ///
       void addEventListener([in] String event_name, [in] EventListener listener);

       ///
       /// Removes single DOM2 event listener for specified event
       ///
       /// Possible values are:
       /// event_name: "appStateChangeEvent", listener: function that has arguments similar to the
appStateChangeEvent interface properties.
       ///
       /// @param event_name Name of the event
       /// @param listener named javascript function that has arguments like the properties listed
in the listener's interface.
       ///
```

```
        void removeEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes all DOM2 event listeners for specified event
        ///
        /// @param event_name Name of the event
        /// @see removeEventListener([in] String event_name, [in] EventListener listener);
        ///
        void removeEventListener([in] String event_name);

        ///
        /// Application terminated
        ///
        const Integer TERMINATED = 1;

    }
```

4.2. Custom Download Manager

Typically, a monitor showing digital signage should always display contents; it's therefore a core feature to be able to download locally certain resources to be displayed, so that they could be played out even if the network is down. While most of this can be achieved through HTML5 app cache or local storage, they have several limitations (file size, media type, etc.). It is important than to be able to save media on the local filesystem and be able to playback them in the BOM of a page.

The solution identified is to use some specific APIs, based on the OIPF Download Manager but a bit simplified, grouped under an Object called CustomDownloadManager, that allows to download locally resources as files. Those can then be used through a lightweight local webserver mapping the folder where they are downloaded through "http" protocol, and with XHR properly configured.

This is shown and described in detail in the demo called "**Download manager**" in the tools package.

The APIs and events of such object are:

```
///
/// This is a class which implements global CustomDownloadManager object. Responsible for starting a
download and keeping a record of it until it is finished either with success or error. Every
download should have a unique destination. User is responsible for providing a unique destination
and the safety of the destination file.
///
interface CustomDownloadManager {
        ///
        /// Queries if the given storage has available free space in given size in bytes. Note:
Destination should be the root of the storage. Root of storage is received from the StorageManager
return of getListOfAvailableStorages()->RemovableDevice.path. Using registerDownloadURL does not
reserve the final size of the download file so while a download is in progress Storage size
decreases, meaning that this function will have different results if there are downloads in
progress. This function is most useful before starting any download. User should know the summation
of the sizes of all files to be downloaded and check if this will fit. Example: A 5MB video file and
a 2MB image file will be downloaded. This function should be called before starting the downloads
with the parameter 7 000 000
        /// @param storage_device_path Storage device root path.
        /// @param sizeInBytes Free space to query.
        ///  @return one of the following: DOWNLOAD_POSSIBLE, DOWNLOAD_INSUFFICIENT_STORAGE,
DOWNLOAD_STORAGE_NOT_AVAILABLE.
        ///
        [out] enumerations checkDownloadPossible([in] String storage_device_path, [in] Integer
sizeInBytes);

        ///
        /// Starts download from the given url address. Non-blocking, can start multiple downloads
of the same URL but destinations should be unique. Each download will raise a downloadStateChange
event with a DOWNLOAD_IN_PROGRESS status. The destination path should consist of both the directory
and the filename.
        /// @param url Download url of the remote file.
        /// @param destination Download destination of the file.
        /// @return Returns true if download request is added successfully. If false is returned, a
download to the same destination is in progress already(User will not receive an event that
```

```
indicates that the download is already in progress. File will not be created for a false return so
you don't need to clean it up.). Note: does not indicate result of the download
        ///
        [out] Boolean registerDownloadURL([in] String url, [in] String destination);

        ///
        /// Requests for a list of ongoing downloads. Each download will raise a downloadStateChange
event with a DOWNLOAD_IN_PROGRESS status.
        /// @return False on internal error, otherwise true.
        ///
        [out] Boolean getDownloads();

        ///
        /// Synchronous check if the download is in progress. Note: StorageManager actions on a
download destination should be performed if and only if this function returns false.
        /// @param destination Destination that matches a download.
        /// @return True if download is in progress, otherwise false.
        ///
        [out] Boolean isInProgress([in] String destination);

        ///
        /// Removes a download in progress. Notes: User should use StorageManager to clean up the
destination after removal. Does not remove the file of a download that is completed.
        /// @param destination Download destination of the file.
        /// @return False on internal error otherwise true.
        ///
        [out] Boolean remove([in] String destination);

        ///
        /// Event raised when the downloading process is complete
        ///
        interface downloadStateChange : EventListener {
            String url; /// Download url
            enumerations Status; /// Download type(Success, in Progress, Fail).
            String destination; /// Physical location of download to be used with Media Player
            String errorMessage; /// Download status error message. Returns HTTP Status Code if
status is DOWNLOAD_REMOTE_ERROR
        }

        ///
        /// Adds DOM2 event listener for specified event.
        ///
        /// Possible values are:
        /// event_name: "downloadStateChange", listener: function that has arguments similar to the
downloadStateChange interface properties.
        ///
        /// @param event_name Name of the event.
        /// @param listener Javascript function that has arguments like the properties listed in the
listener's interface.
        ///
        void addEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes single DOM2 event listener for specified event
        ///
        /// Possible values are:
        /// event_name: "downloadStateChange", listener: function that has arguments similar to the
downloadStateChange interface properties.
        ///
        /// @param event_name Name of the event
        /// @param listener Named javascript function that has arguments like the properties listed
in the listener's interface.
        ///
        void removeEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes all DOM2 event listeners for specified event
        ///
        /// @param event_name Name of the event
        /// @see removeEventListener([in] String event_name, [in] EventListener listener);
        ///
        void removeEventListener([in] String event_name);
        ///
        /// Insufficent space for download
        ///
        const Integer DOWNLOAD_INSUFFICIENT_STORAGE = 1;
```

```
        ///
        /// Download failed
        ///
        const Integer DOWNLOAD_FAILED = 8;

        ///
        /// Couldn't create the file. e.g path was invalid.
        ///
        const Integer DOWNLOAD_FILE_CREATION_ERROR = 5;

        ///
        /// There was a problem with the local network interface.
        ///
        const Integer DOWNLOAD_LOCAL_ERROR = 7;

        ///
        /// Storage is unreachable
        ///
        const Integer DOWNLOAD_STORAGE_NOT_AVAILABLE = 2;

        ///
        /// The server could not provide the content. Check downloadStateChange.errorMessage. e.g.
the file may not exist on the server.
        ///
        const Integer DOWNLOAD_REMOTE_ERROR = 6;

        ///
        /// remove function was called and successfully executed
        ///
        const Integer DOWNLOAD_CANCELED_BY_USER = 3;

        ///
        /// Couldn't write to destination path. e.g. file was deleted.
        ///
        const Integer DOWNLOAD_FILE_WRITE_ERROR = 4;

        ///
        /// Download is in progress
        ///
        const Integer DOWNLOAD_IN_PROGRESS = 2;

        ///
        /// Have enough space for download
        ///
        const Integer DOWNLOAD_POSSIBLE = 0;

        ///
        /// Download is successful
        ///
        const Integer DOWNLOAD_COMPLETED = 1;

}
```

4.3. Source Input management

There is the need to manage the different inputs of the monitor and have APIs to programmatically control the current input selected.

The following APIs are available:

```
interface InputManager {
      ///
      /// This function returns list of available AVInputs.
      /// @return List of available AVInputs
      ///
      [out] AVInput[] getAVInputs();
```

```
	///
	/// This function returns current AV Input.
	/// @return Current AVInput
	///
	[out] AVInput getCurrentAVInput();

	///
	/// OBSOLETE. This function will be removed from API and should not be used.
	/// @return Current AVInput
	///
	[out] AVInput getAVInput();

	///
	/// Selects the specified AVInput.
	/// @param input AVInput object.
	///
	void selectSource([in] AVInput input);

	///
	/// Set the video position to the specified coordinates.
	/// @param x X coordinate of the new video position.
	/// @param y Y coordinate of the new video position.
	/// @param width Width of the new video position.
	/// @param height Height of the new video position.
	///
	void setVideoWindow([in] Integer x, [in] Integer y, [in] Integer width, [in] Integer height);

	///
	/// Stop currently selected AVInput.
	///
	void stopCurrentSource();

	///
	/// Event raised when a change in the AVInput has been occurred
	///
	Interface currentAVInputChangeEvent: EventListener {
	 enumerations status; /// current status
	}

	///
	/// Event raised when a change in the external source list has been occurred
	///
	interface externalSourceListUpdated: EventListener {
	}

	///
	/// Adds DOM2 event listener for specified event.
	///
	/// Possible values are:
	/// event_name: "currentAVInputChangeEvent", listener: function that has arguments similar to
the currentAVInputChangeEvent interface properties.
	///
	/// @param event_name Name of the event.
	/// @param listener javascript function that has arguments like the properties listed in the
listener's interface.
	///
	void addEventListener([in] String event_name, [in] EventListener listener);


	///
	/// Removes single DOM2 event listener for specified event
	///
	/// Possible values are:
	/// event_name: "currentAVInputChangeEvent", listener: function that has arguments similar to
the currentAVInputChangeEvent interface properties.
	///
	/// @param event_name Name of the event
	/// @param listener named javascript function that has arguments like the properties listed in
the listener's interface.
	///
	void removeEventListener([in] String event_name, [in] EventListener listener);

	///
	/// Removes all DOM2 event listeners for specified event
	///
	/// @param event_name Name of the event
```

```
        /// @see removeEventListener([in] String event_name, [in] EventListener listener);
        ///
        void removeEventListener([in] String event_name);
        ///
        /// External Interface Status None
        ///
        const Integer None = 0;

        ///
        /// External Interface Status Connected
        ///
        const Integer Connected = 1;

        ///
        /// External Interface Status Disconnected
        ///
        const Integer Disconnected = 2;

        ///
        /// External Interface Status AttributesChanged
        ///
        const Integer AttributesChanged = 3;

        ///
        /// External Interface Status VideoReady
        ///
        const Integer VideoReady = 4;

        ///
        /// External Interface Status VideoLost
        ///
        const Integer VideoLost = 5;

        ///
        /// External Interface Status AutoAdjustDone
        ///
        const Integer AutoAdjustDone = 6;

        ///
        /// External Interface Status SourceChanged
        ///
        const Integer SourceChanged = 7;
}


interface AVInput {
            ///
            /// type of the externalAVInput
            ///
            readonly attribute String type;

            ///
            /// number of the externalAVInput
            ///
            readonly attribute Integer number;

            ///
            /// name of the externalAVInput
            ///
            readonly attribute String name;

            ///
            /// status of the externalAVInput
            ///
            readonly attribute Boolean enabled;
}
```

4.4. Settings management

There are several objects to manage the different settings of the monitor (e.g., volume, language, active standby, etc.).

```
///
/// This is a class which implements global Power Settings.
///
interface PowerSettings {
	///
	/// @brief Get current activeStandby status.
	/// @return true if activeStandby is enable or not.
	///
	/// Note that this function will be deprecated. Use isActiveStandbyEnabled() instead.
	///
	[out] Boolean getActiveStandby();

	///
	/// @brief Set active standby status.
	/// @param active enable activeStandby or not.
	/// @return  true if function call is successful, false otherwise
	///
	/// Note that this function will be deprecated. Use setActiveStandbyEnabled() instead.
	///
	[out] Boolean setActiveStandby([in] Boolean active);

	///
	/// @brief Get whether active standby feature is enabled. Active standby means, no real (low
power) standby,
	/// always keep in pseudo standby
	/// @return true if activeStandby is enabled or not.
	///
	[out] Boolean isActiveStandbyEnabled();

	///
	/// @brief Set Active Standby Enabled.
	/// @param active enable activeStandby or not.
	/// @return  true if function call is successful, false otherwise
	///
	[out] Boolean setActiveStandbyEnabled([in] Boolean active);

	///
	/// @brief set auto off mode.
	/// @param mode sets AutoTvOffMode.
	/// @return  true if function call is successful, false otherwise
	///
	[out] Boolean setAutoTvOffMode([in] enumerations mode);

	///
	/// @brief get auto off mode.
	/// @return  current AutoTvOffMode
	///
	[out] Integer getAutoTvOffMode();

	///
	/// AutoTvOffInvalid mode
	///
	const Integer AutoTvOffInvalid = -1;

	///
	/// AutoTvOffDisabled mode
	///
	const Integer AutoTvOffDisabled = 0;

	///
	/// AutoTvOff1Hour mode
	///
	const Integer AutoTvOff1Hour = 1;

	///
	/// AutoTvOff2Hour mode
	///
	const Integer AutoTvOff2Hour = 2;

	///
	/// AutoTvOff3Hour mode
	///
	const Integer AutoTvOff3Hour = 3;
```

```
	///
	/// AutoTvOff4Hour mode
	///
	const Integer AutoTvOff4Hour = 4;

	///
	/// AutoTvOff5Hour mode
	///
	const Integer AutoTvOff5Hour = 5;

	///
	/// AutoTvOff6Hour mode
	///
	const Integer AutoTvOff6Hour = 6;

	///
	/// AutoTvOff7Hour mode
	///
	const Integer AutoTvOff7Hour = 7;

	///
	/// AutoTvOff8Hour mode
	///
	const Integer AutoTvOff8Hour = 8;
}

///
/// This is a class which implements global Sound Settings.
///
interface SoundSettings {
	///
	/// @brief Get current volume level.Note that if headphone is connected, headphone's volume
will be returned.
	/// @return Current volume level.
	///
	[out] Integer getVolumeLevel();

	///
	/// @brief Set volume level.
	/// @param volume_level New volume level. Note that if headphone is connected, headphone's
volume will be changed.
	/// @return  true if function call is successful, false otherwise
	///
	[out] Boolean setVolumeLevel([in] Integer volume_level);


	///
	/// @brief Gets audio output device.
	/// @Current audio output device.
	///
	[out] Integer getCurrentAudioOutputDevice();

	///
	/// Default TV Speaker
	///
	const Integer TVSpeaker = 0;

	///
	/// Head Phone Connected
	///
	const Integer HeadPhone = 1;

	///
	/// Digital Line Out Connected
	///
	const Integer DigitalLine = 2;
}
```

4.5. Power management

There are specific APIs to control panel and device power.
This is shown and described in detail in the demo called “**Video Active Standby**” in the tools package.

```
interface StandbyController {
	///
	/// Gets if device in active standby mode or not.
	/// @return true if device is in standby mode, false otherwise.
	///
	[out] Boolean isDeviceInActiveStandby();

	///
	/// Enters active standby.
	///  Note that, if active standby feature is not enabled (either from OSD menu or
///PowerSettings::setActiveStandbyEnabled()),
	/// Monitor will power down when this function is called.
	/// @return if it returns false, enter standby can not be performed.
	///
	[out] Boolean enterActiveStandby();

	///
	/// Quits active standby operations if any active standby operations is continue.
	/// @return true, if monitor is in active standby state and quit command is executed, returns
false if monitor is not in active standby state.
	///
	[out] Boolean quitActiveStandby();
}
```

4.6. Settings Storage Manager

There are some simple APIs to be able to easily read and write simple data that could be accessed from any HTML window running – regardless of the URL domain:

```
///
/// This is a class which implements global JsppDatabaseManager object.
///
interface DatabaseManager {
	///
	/// Get data from lookup table of corresponding key.
	/// @param key String that is queried in lookup table.
	/// @return String data if key is found in lookup table; empty string, otherwise.
	///
	[out] String read([in] String key);

	///
	/// Insert key-value pair into lookup table. If key already exists in lookup table, then its
value is updated. That means key is master and unique. Exceeding size of parameters throws runtime
error.
	/// @param key String data - max size is 255 char.
	/// @param value String data - max size is 255 char.
	/// @return true if data is written successfully; false, otherwise.
	///
	[out] Boolean write([in] String key, [in] String value);

	///
	/// Clear lookup table.
	/// @return true if lookup table exists and is cleared successfully, false otherwise.
	///
	[out] Boolean clear();

	///
	/// Event raised when value of any key has been changed
	///
	interface keyValueChanged : EventListener {
		String key; /// Key of which data has changed.
		String new_value; /// New string data of the key
	}

	///
	/// Adds DOM2 event listener for specified event.
	///
	/// Possible values are:
	///  event_name: "keyValueChanged", listener: function that has arguments similar to the
keyValueChanged interface properties.
	///
	/// @param event_name Name of the event.
```

```
        /// @param listener Javascript function that has arguments like the properties listed in the
listener's interface.
        ///
        void addEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes single DOM2 event listener for specified event
        ///
        /// Possible values are:
        /// event_name: "keyValueChanged", listener: function that has arguments similar to the
keyValueChanged interface properties.
        ///
        /// @param event_name Name of the event
        /// @param listener Named javascript function that has arguments like the properties listed
in the listener's interface.
        ///
        void removeEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes all DOM2 event listeners for specified event
        ///
        /// @param event_name Name of the event
        /// @see removeEventListener([in] String event_name, [in] EventListener listener);
        ///
        void removeEventListener([in] String event_name);
}
```

4.7. Date / Time

Often there is the need to have access to local date/time set in the monitor. For this there are the following IDLs available:

```
///
/// This is a class which implements global DateTimeSettings object.
///
interface DateTimeSettings {
        ///
          /// This function sets current date time. Invalid type of parameter throws runtime error.
          /// @param current_date_time Ctime of the current time. Should be in unix timestamp format.
          /// @return Return true if operation is completed successfully, false otherwise
          ///
          [out] Boolean setCurrentDateTime([in] Integer current_date_time);

          ///
          /// This function checks whether broadcast time is known or not.
          /// @return Return true if Broadcast time is known
          ///
          [out] Boolean isBroadcastTimeKnown();

          ///
          /// This function gets Local time.
          /// @return current time value that contains timezone information.
          /// Returned value is in unix timestamp format.(Based on seconds since standard epoch of
1/1/1970)
          ///
          [out] Integer getLocalCTime();

          ///
          /// This function gets total timezone offset count.
          /// @return Return count of timezone offsets supported
          ///
          [out] Integer getTimeZoneCount();

          ///
          /// This function gets timezone offset value
          /// @return Return timezone offset value as seconds.
          ///
          [out] Integer getTimeZoneOffset();

          ///
          /// This function gets UTC time.
```

```
         /// @return Current time value that does not contain timezone information.
         /// Returned value is in unix timestamp format.(Based on seconds since standard epoch of
1/1/1970)
         ///
         [out] Integer getUTCCTime();
}
```

4.8. Network

There is the need to have APIs to manage network. The following IDLs are available for this purpose:

```
///
/// This is a class which implements global NetworkManager object.
///
interface NetworkManager {
        ///
        /// This function checks internet connection.
        /// @return If the connection is established, function will return True. If the connection
is not established, function will return false
        ///
        [out] Boolean checkInternetConnection();

        ///
        ///  This function enables wifi and starts scan. When the scan is completed
scanCompletedEvent will be fired.
        /// Then Available Wi-Fi networks can be reached by calling getWifiScanResults function.
        /// @return Returns true if wireless network scan is started successfully. Otherwise
function returns false.
        ///
        [out] Boolean startWifiNetworkScan();

        ///
        /// This function is used for connecting to wired network.
        /// @return If there is a wired connection returns true, otherwise returns false.
        ///
        [out] Boolean joinWiredNetwork();

        ///
        /// This function returns connected wifi network
        /// @return Return wifi network
        ///
        [out] WifiNetwork getConnectedWifiNetwork();

        ///
        /// This function returns wifi scan results.
        /// @return Return wifi network array
        ///
        [out] WifiNetwork[] getWifiScanResults();



        ///
        /// This function connects wifi network whose id is given.
        /// @param id Id of the wifi network.
        /// @param password Password.
        /// @return If the connection is established, function will return True. If the connection
is not established, function will return false
        ///
        [out] Boolean joinWifiNetwork([in] Integer id, [in] String password);

        ///
        /// This function returns network interface type according to given mac id.
        /// @param mac_address Mac address network interface.
        ///  @return If type is wired ,returns WiredNetwork.If type is wireless return
WirelessNetwork.
        ///
        [out] enumerations getNetworkInterfaceType([in] String mac_address);

        ///
        /// Event raised when wifi networks scan is completed
```

```
        ///
        interface scanCompletedEvent : EventListener {
        }

        ///
        /// Event raised when internet connection is established or internet connection is lost
        ///
        interface internetConnectionEvent : EventListener {
                enumerations connection_type; /// Connection type(Online,Offline).
        }

        ///
        /// Adds DOM2 event listener for specified event.
        ///
        /// Possible values are:
        /// event_name: "internetConnectionEvent", listener: function that has arguments similar to
the internetConnectionEvent interface properties.
        /// event_name: "scanCompletedEvent", listener: function that has arguments similar to the
scanCompletedEvent interface properties.
        ///
        /// @param event_name Name of the event.
        /// @param listener Javascript function that has arguments like the properties listed in the
listener's interface.
        ///
        void addEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes single DOM2 event listener for specified event
        ///
        /// Possible values are:
        /// event_name: "internetConnectionEvent", listener: function that has arguments similar to
the internetConnectionEvent interface properties.
        /// event_name: "scanCompletedEvent", listener: function that has arguments similar to the
scanCompletedEvent interface properties.
        ///
        /// @param event_name Name of the event
        /// @param listener Named javascript function that has arguments like the properties listed
in the listener's interface.
        ///
        void removeEventListener([in] String event_name, [in] EventListener listener);

        ///
        /// Removes all DOM2 event listeners for specified event
        ///
        /// @param event_name Name of the event
        /// @see removeEventListener([in] String event_name, [in] EventListener listener);
        ///
        void removeEventListener([in] String event_name);

        ///
        /// No internet connection
        ///
        const Integer Offline = 0;

        ///
        /// Internet connection
        ///
        const Integer Online = 1;

        ///
        /// Connection type wired
        ///
        const Integer WiredNetwork = 0;

        ///
        /// Connection type wireless
        ///
        const Integer WirelessNetwork = 1;

}

///
///
interface WifiNetwork {
        ///
        /// Strength information of the wifi network
```

```
        ///
        readonly attribute Integer strength;

        ///
        /// Name information of the wifi network
        ///
        readonly attribute String name;

        ///
        /// Access type information of the wifi network
        ///
        readonly attribute Boolean isLocked;

        ///
        /// Wifi network mac address
        ///
        readonly attribute String macAddress;

        ///
        /// Wifi network id
        ///
        readonly attribute Integer id;

        ///
        /// Channel information of the wifi network
        ///
        readonly attribute Integer channel;

}


/// function that returns mac address to identify device

interface MacId {
        ///
        /// return mac address of device
        /// @return returns mac id of device
        ///
        [out] String getMacId();
}
```

4.9. Storage Manager

In order to manage the full lifecycle of the files downloaded, the Storage Manager APIs are defined:

```
/// This is a class which implements global StorageManager object.
///
interface StorageManager {
    ///
    /// This function returns avaliable storages.
    /// @return List of removable devices
    ///
    [out] RemovableDevice[] getListOfAvailableStorages();




    ///
    /// This function copies a given file given to a new path given. Any folder can be copied by
using this function only if the filePath ends with '/' character.
    /// @param filePath Absolute path of the file that will be copied
    /// @param newPath New absolute path that the file will be copied to
    /// @return resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4,
WriteError:5, MemoryAllocFail:6, BadParameter:7, DiskFull:8, DiskMissing:9, DirNotEmpty:10,
DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations copy([in] String filePath, [in] String newPath);

    ///
```

```
    /// This function sets mediaFileReadProgressEvent. The list of playable files will be sent through mediaFileReadProgressEvent event and, when progress is finished, mediaFileReadProgressCompletedEvent is raised.
    /// @param path Absolute path of removable device
    /// @param filetype Type of file which is wanted to be listed (Music,Picture,Video,all)
    /// Available file types are: MediaTypePicture:0, MediaTypeAudio:1, MediaTypeVideo:2, MediaTypeRecord:3, MediaTypeAll:4
    ///
    void getListOfPlayableFiles([in] String path, [in] enumerations filetype);

    ///
    /// This function formats USB removable device given.
    /// @param path absolute path of removable device that will be formatted
    /// @return resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4, WriteError:5, MemoryAllocFail:6, BadParameter:7, DiskFull:8, DiskMissing:9, DirNotEmpty:10, DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations format([in] String path);

    ///
    /// This function returns in a StorageInfo object the available space of removable device in kB and the errorCode corresponding to the file system
    /// error code.
    /// Available error codes are: NoError:0, BadParameter:7, DiskMissing:9, OperationFail:14
    /// @param path Absolute path of removable device that will be checked for empty space
    /// @return resulting StorageInfo object containing file system error code and available space in kB.
    ///
    [out] StorageInfo checkEmptySpace([in] String path);


    ///
    /// This function returns the StorageInfo object.
    /// Return value corresponds to the total size of removable device in kB and errorCode corresponds to the file system error code.
    /// Available error codes are: NoError:0, BadParameter:7, DiskMissing:9, OperationFail:14
    /// @param path Absolute path of removable device that will be checked for empty space
    /// @return resulting StorageInfo object containing file system error code and total size in kB.
    ///
    [out] StorageInfo checkTotalSize([in] String path);



    ///
    /// This function checks whether format of given device is required or not.
    /// @param path Absolute path of removable device that will be checked
    /// @return true if format is required; false, otherwise.
    ///
    [out] Boolean isFormatRequired([in] String path);


    ///
    /// This function checks whether the given device is internal or external device.
    /// @param path Absolute path of removable device that will be checked
    /// @return true if the device is internal; false, otherwise.
    ///
    [out] Boolean isInternalUSB ([in] String path);

    ///
    /// This function sets mediaFileReadProgressEvent. File list will be sent through mediaFileReadProgressEvent event and, when progress is finished, mediaFileReadProgressCompletedEvent is raised.
    /// @param path Absolute path of the directory under that files will be listed.
    /// @return resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4, WriteError:5, MemoryAllocFail:6, BadParameter:7, DiskFull:8, DiskMissing:9, DirNotEmpty:10, DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations listFiles([in] String path);


    /// This function returns MD5 of a file stored on the storage. It will throw 'Invalid file' exception if file does not exist, path is not valid, or it is not accessible. It will throw 'File read error' exception if file cannot be opened.
    /// @param filePath      absolute path of the file whose MD5 will be get
```

```
    /// @return MD5 of the file on success; empty string, otherwise.

[out] String getFileMd5 ([in] String filePath)


    ///
    /// This function renames the file or folder given to new name given
    /// @param filePath Absolute path of the file that will be renamed
    /// @param newName New name with absolute path of the file
    /// @return resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4,
WriteError:5,   MemoryAllocFail:6,   BadParameter:7,   DiskFull:8,   DiskMissing:9,   DirNotEmpty:10,
DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations rename([in] String filePath, [in] String newName);

    ///
    /// This function creates new folder
    /// @param path Absolute path of the file that will be created
    /// @return Resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4,
WriteError:5, MemoryAllocFail:6, BadParameter:7, DiskFull:8, DiskMissing:9, DirNotEmpty:10,
DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations makeDir([in] String path);

    ///
    /// This function deletes the file given. '/' character should be written at the end of the path
in order to specify the folder.
    /// @param filePath Absolute path of the file that will be deleted
    /// @return resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4,
WriteError:5, MemoryAllocFail:6, BadParameter:7, DiskFull:8, DiskMissing:9, DirNotEmpty:10,
DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations delete([in] String filePath);


    ///
    /// This function decompresses the zip file given.
    /// @param file Absolute path of zip file that will be extracted
    /// @param path Absolute path where zip file will be extracted into
    /// @return resulting file system error code
    /// Available error codes are: NoError:0, FileNotFound:1, OpenError:2, EndOfData:3, ReadError:4,
WriteError:5, MemoryAllocFail:6, BadParameter:7, DiskFull:8, DiskMissing:9, DirNotEmpty:10,
DirPathNotFound:11, DirAlreadyExists:12, PermissionDenied:13, OperationFail:14
    ///
    [out] enumerations unzip([in] String file, [in] String path);

    ///
    /// Event raised when getting playable files in the storage is completed
    ///
    interface mediaFileReadProgressCompletedEvent : EventListener {
    }

    ///
    /// Event raised when a removable device is connected
    ///
    interface removableDeviceConnected : EventListener {
            RemovableDevice device; /// Removable device
    }

    ///
    /// Event is raised when a removable device disconnected
    ///
    interface removableDeviceDisconnected : EventListener {
            RemovableDevice device; /// Removable device
    }

    ///
    /// Event is raised when getting 10 playable files in the storage is completed
    ///
    interface mediaFileReadProgressEvent : EventListener {
            MediaFile[] files; /// List of playable files in the storage
    }
```

```
    ///
    /// Adds DOM2 event listener for specified event.
    ///
    /// Possible values are:
    /// event_name: "removableDeviceConnected", listener: function that has arguments similar to the
removableDeviceConnected interface properties.
    /// event_name: "removableDeviceDisconnected", listener: function that has arguments similar to
the removableDeviceDisconnected interface properties.
    /// event_name: "mediaFileReadProgressEvent", listener: function that has arguments similar to
the mediaFileReadProgressEvent interface properties.
    ///
    /// @param event_name Name of the event.
    /// @param listener Javascript function that has arguments like the properties listed in the
listener's interface.
    ///
    void addEventListener([in] String event_name, [in] EventListener listener);


    ///
    /// Removes single DOM2 event listener for specified event
    ///
    /// Possible values are:
    /// event_name: "removableDeviceConnected", listener: function that has arguments similar to the
removableDeviceConnected interface properties.
    /// event_name: "removableDeviceDisconnected", listener: function that has arguments similar to
the removableDeviceDisconnected interface properties.
    /// event_name: "mediaFileReadProgressEvent", listener: function that has arguments similar to
the mediaFileReadProgressEvent interface properties.
    ///
    /// @param event_name Name of the event
    /// @param listener Named javascript function that has arguments like the properties listed in
the listener's interface.
    ///
    void removeEventListener([in] String event_name, [in] EventListener listener);

    ///
    /// Removes all DOM2 event listeners for specified event
    ///
    /// @param event_name Name of the event
    /// @see removeEventListener([in] String event_name, [in] EventListener listener);
    ///
    void removeEventListener([in] String event_name);

    ///
    /// Media type picture
    ///
    const Integer MediaTypePicture = 0;

    ///
    /// Media type audio
    ///
    const Integer MediaTypeAudio = 1;

    ///
    /// Media type video
    ///
    const Integer MediaTypeVideo = 2;

    ///
    /// Media type record
    ///
    const Integer MediaTypeRecord = 3;

    ///
    /// Media type all
    ///
    const Integer MediaTypeAll = 4;

    ///
    /// Operation completed successfully.
    ///
    const Integer NoError = 0;

    ///
    /// An invalid parameter was passed to the function.
    ///
```

```
const Integer FileNotFound = 1;

///
/// Error attempting to open or create a file.
///
const Integer OpenError = 2;

///
/// End of data was reached.  This is end of file
///
const Integer EndOfData = 3;

///
/// Error occurred reading data from the disk.
///
const Integer ReadError = 4;

///
/// Error occurred writing data to the disk.
///
const Integer WriteError = 5;

///
/// Unable to allocate memory within the implementation of a function.
///
const Integer MemoryAllocFail = 6;

///
/// An invalid parameter was passed to the function.
///
const Integer BadParameter = 7;

///
/// The relevant disk is full.
///
const Integer DiskFull = 8;

///
/// The drive's removable disk is not present -- for future support.
///
const Integer DiskMissing = 9;

///
/// Attempt was made to delete a directory that is not empty.
///
const Integer DirNotEmpty = 10;

///
/// Attempt was made to create a directory with a wrong path
///
const Integer DirPathNotFound = 11;

///
/// Attempt was made to create a directory that already exists.
///
const Integer DirAlreadyExists = 12;

///
/// Not allowed to execute an operation
///
const Integer PermissionDenied = 13;

///
/// An unspecified error occurred.
///
const Integer OperationFail = 14;

///
/// Removable device file system format fat
///
const Integer Fat = 0;

///
/// Removable device file system format ntfs
///
const Integer Ntfs = 1;
```

```
    ///
    /// Removable device file system format ext3
    ///
    const Integer Ext3 = 2;

    ///
    /// Removable device file system format ext2
    ///
    const Integer Ext2 = 3;

    ///
    /// Removable device type
    ///
    const Integer DeviceTypeUsb = 0;

    ///
    /// Removable device type
    ///
    const Integer DeviceTypeDlna = 1;

}

///
/// This object is used to keep Storage information
///
interface StorageInfo {

    ///
    /// Error Code for file system operations
    ///
    readonly attribute enumerations errorCode;

    ///
    /// Available free space on Removable Device
    ///
    readonly attribute Integer returnValue;
}

///
/// This object is used to keep Removable Device information
///
interface RemovableDevice {
    ///
    /// Removable Device File System Type
    ///
    readonly attribute enumerations fileSystemFormat;

    ///
    /// Removable Device Number
    ///
    readonly attribute Integer deviceNumber;

    ///
    /// Removable Device Type
    ///
    readonly attribute enumerations type;

    ///
    /// Removable Device Absolute Path
    ///
    readonly attribute String path;
}

///
///
///
interface MediaFile {

    ///
    /// Media file name
    ///
    readonly attribute String name;

    ///
    /// Media file creation time
```

```
    ///
    readonly attribute Integer creationTime;

    ///
    /// Media file last modify time
    ///
    readonly attribute Integer lastModifyTime;

    ///
    /// Media file operating system path
    ///
    readonly attribute String osPath;

    ///
    /// Media file path
    ///
    readonly attribute String path;

    ///
    /// Media file short name
    ///
    readonly attribute String shortName;

    ///
    /// Media file type
    ///
    readonly attribute enumerations type;


    ///
    /// Media file file size
    ///
    readonly attribute Integer fileSize;

}
```

4.10. Remote control key simulation

It's possible to simulate a remote control key press:

```
///
/// This is a class which implements global RemoteKeyGenerator object.
///
interface RemoteKeyGenerator {
      /// This function generates a remote key press event from browser
      /// @param key Key code that is wanted to be processed by platform.
      /// Currently available key codes (from standard key codes):
      /// VK_ENTER, VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT,
      /// VK_BACK, VK_BACKSPACE, VK_DELETE, VK_TAB, VK_ESCAPE,
      /// VK_RED, VK_GREEN, VK_BLUE, VK_YELLOW,
      /// VK_0, VK_1, VK_2, VK_3, VK_4, VK_5, VK_6, VK_7, VK_8, VK_9,
      /// VK_REWIND, VK_PLAY, VK_FAST_FWD, VK_PAUSE, VK_STOP, VK_PLAY_PAUSE,
      /// VK_TRACK_PREV, VK_TRACK_NEXT, VK_CHANNEL_UP, VK_CHANNEL_DOWN, VK_INFO, VK_SUBTITLE,
      /// VK_RECORD, VK_TELETEXT, VK_GUIDE, VK_MENU,
      /// VK_MUTE, VK_VOLUME_UP, VK_VOLUME_DOWN, VK_POWER, VK_HOME,
      ///
      /// If any other code is given, it will be converted into unknown key.
      /// @return true if works well, false otherwise.
      ///
      [out] Boolean generateKeyPress([in] Integer key);
}
```

4.11. RS232 commands

It's possible to trigger any supported RS232 command through a specific API:

For the list of supported commands please refer to the document ToshibaEmbeddedSignage-RS232CommandsSpecification-1.2.0.doc

```
///
/// This is a class which implements global RS232LANCommand object.
///
interface Rs232LanCommand {
      /// Sends RS232/LAN command to the platform.
      /// @param command
      ///
      void execute([in] String command);

      ///   Event raised when execution finishes. This event returns the result of the command executed.
      ///
      interface executionResultEvent: EventListener {
        String result; /// Resulting string
      }

      /// Adds DOM2 event listener for specified event.
      ///
      /// Possible values are:
      /// event_name: "executionResultEvent", listener: function that has arguments similar to the executionResultEvent interface properties.
      ///
      /// @param event_name Name of the event.
      /// @param listener Javascript function that has arguments like the properties listed in the listener's interface.
      ///
      void addEventListener([in] String event_name, [in] EventListener listener);

      /// Removes single DOM2 event listener for specified event
      ///
      /// Possible values are:
      /// event_name: "executionResultEvent", listener: function that has arguments similar to the executionResultEvent interface properties.
      ///
      /// @param event_name Name of the event
      /// @param listener Named javascript function that has arguments like the properties listed in the listener's interface.
      ///
      void removeEventListener([in] String event_name, [in] EventListener listener);

      /// Removes all DOM2 event listeners for specified event
      ///
      /// @param event_name Name of the event
      /// @see removeEventListener([in] String event_name, [in] EventListener listener);
      ///
      void removeEventListener([in] String event_name);
}
```

4.12. Screenshot

There is an utility API to take a monitor screenshot. A screenshot can be taken in different formats and is saved into a specific path.

This is shown and described in detail in the demo called “**Screenshot upload**” in the tools package.

```
///
/// Takes the screenshot, saves it to the path where the last external USB is mounted.
/// The available modes are 'SVG', 'HTML', 'TELETEXT', 'SUBTITLE', and 'COMPOSITE' as a string parameter. Returns 'undefined' otherwise.
/// @param screenshot_mode Mode of the screenshot.
/// @return Path where the screenshot is saved.
///
[out] String takeScreenshot([in] String screenshot_mode);
```

4.13. Garbage Collector

JavaScript by default uses garbage collection to reclaim the memory occupied by strings, objects, arrays, and functions that are no longer in use. The garbage collector is able to determine when it is safe to reclaim memory.
For various reasons, the monitor makes available a special API able to trigger a manual run of the garbage collector, giving the options to the developer to invoke garbage collection when it could be needful during the execution of the application.
The API can be invoked using a simple

```
window.gc()
```

There is no a specific timing when gc() must be called, the only advice is to not abuse of it, because calling gc() more than necessary can generate a side effect, like an overhead of operations to perform.
The application should be clever enough to understand when the used memory is over a certain limit. Memory usage statistics can be easily achieved taking advantage of the memory information exposed by the browser like

window.performance.memory.jsHeapSizeLimit
window.performance.memory.totalJSHeapSize
window.performance.memory.usedJSHeapSize

For more information about these statistics please refer to paragraph 5.10
This API is available only from firmware version 7.48

4.14. File Manager

FileManager plugin allows to read/write files from/to USB flash drive. This plugin is very helpful to write logs on the USB key.
This API is available only from firmware version 7.75

```
const Integer FILE_WRITE_MODE_POSITION = 2
const Integer BAD_PARAMETER = 7
const Integer FILE_OPEN_ERROR = 2
const Integer NO_ERROR = 0
const Integer MEMORY_ALLOC_FAIL = 6
const Integer FILE_WRITE_ERROR = 5
const Integer OPERATION_FAIL = 10
const Integer END_OF_DATA = 3
const Integer FILE_WRITE_MODE_APPEND = 1
const Integer PERMISSION_DENIED = 9
const Integer FILE_WRITE_MODE_TRUNCATE = 0
const Integer DISK_FULL = 8
const Integer FILE_READ_ERROR = 4
const Integer FILE_NOT_FOUND = 1

/// Write data to file.
/// @param data Data to be written.
/// @param path Path of file. If path is not a removable device path, FileManager.PERMISSION_DENIED
/// error is returned.
/// @param position Starting position for writing. Only valid if mode is set to
FileManager.FILE_WRITE_MODE_POSITION.
/// @param mode Writing mode.
/// @param offset Offset of data to be written.
/// @param chunk_length Number of bytes to write. If chunk_length is greater than data byte length,
it is set to data byte length. If offset+chunk_length is greater than data byte length, it is set to
data_byte_length - offset.
/// Returns Result of write operation.

enumerations FileManager::writeFile ([in] String data,
                                     [in] String path,
                                     [in] Integer position,
                                     [in] enumerations mode,
```

```
                                    [in] Integer offset,
                                    [in] Integer chunk_length
)

//Example
var errorCode = FileManager.writeFile(textData, filePath, 0,
FileManager.FILE_WRITE_MODE_TRUNCATE, 0, chunkLength);



/// Read data from file.
/// @param path Path of file. Throws exception if path is not a removable device path.
/// @param position Starting position to read.
/// @param chunk_length Number of bytes to read.
/// Returns Data read from file, or empty string in case of error.

String FileManager::readFile ([in] String path,
                              [in] Integer position,
                              [in] Integer chunk_length
)

/// Example
var readContent = FileManager.readFile(fileToRead, position, chunkLength);
```

5. Development guidelines

5.1. Datetime management

Date and time management is a fundamental topic to face up to when starting the development of a new application. Basically, the problem raises because the monitor has no support for RTC (Real Time Clock). This means that, until the monitor is powered, the datetime is updated and works correctly. When power is off, the datetime is reset to a default date, like 01-01-2010 00:00:00. Thus, every time the application is launched the current time must be set again, otherwise several issues may occur, like

- new Date() doesn't return the correct datetime, it returns a date in the past
- content scheduling is not possible
- HTTPS requests fail, because the validity dates of the certificate are in the future

Developers must treat the absence of a correct datetime on the monitor as the first problem to be addressed during development. This issue must have high priority, because it can compromise the behavior of the whole application.

There are 2 different solutions:

- Use NTP server: in the RS232 APIs, there is a command "UNTP" that force the update of datetime using the NTP server set in the firmware configurations. IMPORTANT: is highly recommended to pre-configure the monitor with a different NTP than the default provided out-of-the-box. Default NTP is not 100% reliable, therefore **pool.ntp.org** should be used instead. This can be done using UNTP command or through the OSD menu as well.
- Set the datetime manually. This can be performed getting the current time as UNIX timestamp from your own server, and then set this timestamp as current time through the method setCurrentDateTime of the DateTimeSettings custom plugin. Example:

```
var t = 1491487989
DateTimeSettings.setCurrentDateTime(t);
```

1491487989 is the current time UNIX timestamp, retrieved from your own server.
Obviously, this server must not be queried in HTTPS, because the SSL certificate would result invalid due to the invalid date.

Both solutions are valid, but the fundamental and mandatory rule is to set correct datetime before to start any other component of the application.

5.2. Media management

Media management is a tricky topic of every application that run in an embedded context. Because of hardware resources limitation, often is easy to face with crashes and annoying bugs, not user-friendly black screen spot switching between video or image animations not smoothly.
There are some rules to take into account handling media. They are not rules valid for a traditional web application displayed on a PC browser, where resources usage (RAM and CPU) is not an issue; they are just mandatory best practice to optimize the code and save memory consumption as much as possible, thinking of the context where the application have to run.

5.2.1. SRC attribute of <img> tag

In case an <img> tag should be removed from the DOM, specification recommend to redefine the SRC attribute of the image to “” (blank) BEFORE the node is detached. This operation triggers the monitor to flush memory quickly.

5.2.2. SRC attribute of <video> tag

When the application uses a single video tag, and many video have to be played in this tag, specifications recommend to set the SRC attribute to “” (blank) at the end of each video, just before replacing it with a new source URL. After the SRC attribute is set to blank, it can be set up with a new video URL. This instruct the monitor to flush quickly the buffer created for the previous video.

Later on, when the new SRC value is set, avoid to call play() function immediately. First of all, a listener must be attached at the <video> tag in order to listen for the events “ended” and “canplay”. When “ended” event is raised, these steps should be followed:

- SRC video attribute should be set to “”
- new video URL is set
- load() method is called
- when “canplay” event is raised, a call to play() can finally be performed.

This trick assures that the next video is always ready when play() is called.

5.2.3. Listener

Listeners are one of the first causes of memory leak, because often they are not removed.
The recommendation is to always remove a previously attached listener when detaching a node from the DOM. At the same time, particularly for image nodes, avoid to attach new listener if the node is never detached and only the SRC attribute of the image changes. Listener should be added once, until image node is not detached. Before detaching the node, the listener should be removed.
Generally speaking, this recommendation must be applied to every kind of node, not only images.

5.3. Available Storages

Here below are listed the 4 options available in the browser to store data.

5.3.1. Local Storage

The Local Storage is a really simple key/value persistence system. Should be considered as a temporary storage made available by the browser, where applications can persist temporary data. The term “temporary data” indicates that such data will be available until FTI (First Time Installation) or Firmware Update are performed. These 2 actions completely erase the local storage, therefore developers must avoid to rely on it to store that kind of information needed even when monitor is reset.

Usage of the Local Storage is recommended only for such data like template configuration, playlist file, volatile entries and any other type of data needed when the application is already set up.
Avoid to store mandatory information whose absence prevents application’s initialization, like license code, activation code, registration code, etc. As will be explained in paragraph 5.3.4, these entries must be stored using the DatabaseManager custom plugin.
Also, avoid to store binary files in the Locale Storage, because its size is very limited.

Here below are listed some useful information about Local Storage limitation on the monitor:

- max size is ~10 MB
- size of single item in local storage is 5242877 characters

Finally, these two alert:

- it isn’t transactional, so it should be used taking care of possible race conditions that could corrupt application’s data.
- the API is synchronous, so it can block the DOM

5.3.2. WebSQL

WebSQL is basically sqlite embedded into the browser.
Its limitations aren't so much related to size, because it can go at least 1 GB without a problem, but inherent in flat file databases (high levels of concurrency) and missing features (stored procs and other higher-end database features).
However, it’s a fast and pretty feature-rich SQL implementation, in fact SQL constructs like select/insert/update/delete are available as well joins, inner selects, etc.
In an embedded signage context, it is often used for store logs, but take care of these cons:

- it blocks the DOM quite a bit, at least for heavy operations
- since 2010 it has been deprecated in favor of IndexedDB.

5.3.3. IndexedDB

IndexedDB is basically a simple flat-file database with hierarchical key/value persistence and basic indexing. Under the hood, it takes advantage of LevelDB, a Google produced database, and is considered the successor to both LocalStorage and WebSQL.

IndexedDB exposes an asynchronous API that supposedly avoids blocking the DOM, but this is not completely true. In fact, IndexedDB works swimmingly well in a web worker, where it runs at roughly the same speed but without blocking the DOM. So that means applications can always offload expensive IndexedDB operations to a worker thread, where there’s no chance of blocking the UI thread. It’s also worth acknowledging that IndexedDB is the only storage option inside of a web worker (or a service worker, for that matter).

5.3.4. DatabaseManager

DatabaseManager is a custom plugin available on the monitor. It exposes two methods, read() and write(), by which you can store key/value entries.

It's a simple persistence system like Locale Storage but unlike Local Storage it has the capability to store data in a persistent storage so that, even after a Firmware Upgrade or a First Time Installation (FTI), they could be retrieved successfully. Usually, after these operations all values in the other storages are completely erased. DatabaseManager custom plugin stores data differently and for this reason it must be highly preferred compared to Local Storage.

Usage example:

DatabaseManager.write("myKey","myValue");
DatabaseManager.read("myKey"); // this returns "myValue"

Some example of information suitable to be stored in DatabaseManager are: license code, registration code and, generally speaking, persistent data required by the application to work again after a monitor reset. Avoid to store such critical data in the local storage.

5.4. jQuery

jQuery is a powerful JavaScript framework making easy to build simple UIs with small effort. Unfortunately, it's not a memory saving library and a lot of programmers don't understand that due to its power a massive amounts of CPU cycles are used.
In an embedded context, resources like CPU and memory are limited and every single line of code must be optimized to guarantee acceptable performances and avoid monitor crashes. Therefore, the recommendation is to avoid the usage of jQuery, please use only native JavaScript when possible.
Also, other third party libraries similar to jQuery should be avoided.

5.5. Garbage collector

JavaScript uses garbage collection to reclaim the memory occupied by strings, objects, arrays, and functions that are no longer in use. This frees the developer from having to explicitly deallocate memory himself and is an important part of what makes JavaScript programming easier than, say, C programming.
A key feature of garbage collection is that the garbage collector must be able to determine when it is safe to reclaim memory. Obviously, it must never reclaim values that are still in use and should collect only values that are no longer reachable; that is, values that cannot be referred to through any of the variables, object properties, or array elements in the program.
For the most part, it's pretty easy to write code that makes life easy for the garbage collector. The only thing you really need to do is not keep references to things not used.

For example, here's some typical code:

```
function test()
{
   var myString = 'a string';
}
test();
```

After the function test has executed, the variable myString will be marked as available to be freed/deleted. This is because the function creates a scope for the variables declared within it. Upon entering the function the browser will create space for the myString variable, and once it ends, everything created in that scope will be marked for collection. At some point later (it's up to the browser to determine when that is) the GC will run and the myString object will be "truly" removed and memory freed up.

Of course, if there are any remaining references, the GC won't collect the variable.

For example:

```
var another = null;

function test()
{
    var str = 'A string I am';
    another = str;
}

test();
```

In this case, the "*another"* variable, which is in global scope, is a reference to the string created inside the function. Since the scope is global, the garbage collector will continue to think it's active.

Also watch out for any cases where you leave off the "*var*" keyword. JavaScript interprets this as declaring a globally scoped variable, for example:

```
// var b = null; // commented out now

function test()
{
    var str = 'A string I am';
    b = str; // oops, no var keyword means this is a global
}
test();
```

Without the "*var"* keyword you could be assuming the *b* variable will be freed because the function has gone out of scope, which is not the case.

How to really make sure the stuff created gets deleted, i.e. cleaned up by the garbage collector?
Firstly, JavaScript has a built-in delete keyword. Unfortunately, it cannot be used to mark an object as removable. delete is there to make possible to specify that a property is removable from its object. Whilst that serves as an indirect way of clearing a reference to an object or variable, it's not explicitly a way of deleting a variable. delete is useful when you want to explicitly make the property of an object undefined, rather than just null. For example:

```
var s = { data: 'test' };
delete s.data;
```

s.data will now be undefined (and also marked for removal by the garbage collector).

delete falls down though if you try to do the same with a variable:

```
var m = 'test';
delete m; // silently returns false (not allowed)
m === 'test'; // true - oops, still a value
```

The delete call will run just fine, but it will silently fail to delete the variable (since it's not a property) and therefore the reference won't be cleared and the memory won't be reclaimed by the garbage collector. The delete call will actually return false, to indicate the variable couldn't be deleted; how nice of it.

The right way to clear a variable is pretty simple: **set every reference it to null**, then let the garbage collector do its job.

```
var m = 'test';
m = null;
m === 'test'; // false
```

This applies to contained properties and objects as well, so if you do:

```
var s = { data: 'test' };
s.data = null;  // not required
s = null;   // this will automatically clear s.data as well
```

Then the data variable is going to be automatically cleared up as well – since no reference remains to it because its outer-scoped parent object was cleared.

Avoid creating lot of objects! Of course, the easiest way to minimize garbage collection is to not create objects at all. Where possible, try to create the object on startup, and simply recycle the same object for as long as possible.
You can actually re-cycle an existing object (providing it has no prototype chain) by deleting all of its properties, restoring it to an empty object like {} and adding properties again. For clean the object you can use the delete keyword:

```
// remove all own properties on obj,
effectively reverting it to a new object
function swipe(obj)
{
        for (var p in obj)
        {
                if (obj.hasOwnProperty(p))
                        delete obj[p];
        }
};
```

Don't abuse (usually referred to as "polluting") the global namespace, i.e. do not create multiple global variables. A more resourceful approach, which does not pollute the global namespace, would be to wrap variable in the module pattern and only use one global variable while exposing multiple variables.

```
//Calculate is the only exposed global variable
var Calculate = function () {
 //all defintions in this closure are local, and will not be exposed to the global namespace
 var Coordinates = [];//array for coordinates
 var Coordinate = function (xcoord, ycoord) {//definition for type Coordinate
   this.x = xcoord;//assign values similar to a constructor
   this.y = ycoord;
  };

          return {
            //these methods will be exposed through the Calculate object
            AddCoordinate: function (x, y) {
            Coordinates.push(new Coordinate(x, y));//Add a new coordinate
            }
         } ;
};
```

Speaking about array, assigning [] to an array is often used as a shorthand to clear it (e.g. arr = [];), but note this creates a new empty array and garbage the old one! It's better to write arr.length = 0; which has the same effect but while re-using the same array object.

Remember that functions are treated as objects as well, so:

```
function getCompare()
{
    return function(a, b) { return a < b; }
}
```

This code will create a new (function) object on every call. Since a function is an object in JavaScript, the garbage collector has to handle those as well. So, try to avoid anonymous function.

Another common source is returning compound results from a function:

```
function getResult()}
{
    return { result: true, value: 'test' }; // creates object every call
}
```

One more example is functions which return functions. Something like this:

```
setTimeout((function (self)
        { return function () {
                self.tick();
        };
})(this), 16);
```

This looks like a reasonable way to call this.tick() every 16 ms. However, it also means every tick it returns a new function! This can be avoided by storing the function permanently, like so:

```
// at startup
this.tickFunc = (function (self)
                { return function () {
                        self.tick();
                };
})(this);

// in the tick() function
setTimeout(this.tickFunc, 16);
```

This re-uses the same function every tick rather than spawning a new one. The same idea can be applied anywhere else functions are returned or otherwise created at runtime.

One more topic: object cloning is a common problem for app developers.
Be very careful when copying anything. Copying big things is generally slow, so don't do it.
for..in loops in JavaScript are particularly bad for this, as they have a devilish specification and will likely never be fast in any engine for arbitrary objects.

When is absolutely needed to copy objects in a performance-critical code, use an array or a custom "copy constructor" function which copies each property explicitly. This is probably the fastest way to do it:

```
function clone(original) {
  this.foo = original.foo;
  this.bar = original.bar;
}
var copy = new clone(original);
```

And finally, the potentially very painful Array.slice:

```
var b = a.slice(1); // creates an entirely new duplicate array
```

Array.slice always creating a new object/array on every call is an important one to watch out for. There's effectively no way to clear an object out of an array without feeling this pain. Whilst modern browsers are doing lots of nice optimizations around this, it's still a killer when done excessively.

Even if you minimize how many objects you create, you'll still be chewing through memory. To keep track of how much you're using, and how hard the garbage collector is working to keep up with you, you can use a pretty simple technique.

The browser give access to the state of the JavaScript heap (the memory allocated to JavaScript objects), through two properties:

window.performance.memory.totalJSHeapSize; // currently used heap memory
window.performance.memory.usedJSHeapSize; // total heap memory

These two values represent how much memory has been allocated to JavaScript, and how much of that allocated memory is currently in use by all the variables/objects.

Even better is using Chrome Developer Tools, that have good support for JavaScript profiling. Profiling starts with obtaining a baseline for the code's current performance, which can be discovered using the Timeline. This will tell how long the application took to run. The Profiles tab then gives a better view into what's happening in the application. The JavaScript CPU profile shows how much CPU time is being used by the code, the CSS selector profile shows how much time is spent processing selectors and Heap snapshots show how much memory is being used by objects.
Using these tools, is possible to isolate, tweak and reprofile to gauge whether changes made to specific functions or operations are improving performance.

5.6. Daily reboot recommendation

In most cases the monitor can run continuously for a lot of days, without any minimal glitch. However, in some cases, due to particular application's features or imperfections, some monitor's components (like the browser) needs a reboot.
Then, in general, for every kind of application, the recommendation is to trigger a monitor reboot every 24h, using the **RST** command included in Rs232LanCommand APIs. Obviously, RST should be scheduled for a very specific timeframe when monitor is not used for its purpose.

5.7. Using GPU: animations

Modern browsers can animate four things really cheaply: position, scale, rotation and opacity.

- Position -> transform: translate (npx, npx)
- Scale -> transform: scale(n)
- Rotation -> transform: rotate (ndeg)
- Opacity -> opacity: 0… 0.99

The process that the browser goes through when new DOM elements should be displayed is pretty simple: calculate the styles that apply to the elements (Recalculate Style), generate the geometry and position for each element (Layout), fill out the pixels for each element into layers (Paint Setup and Paint) and draw the layers out to screen (Composite Layers).
When an element changes, the browser may need to do a layout, which involves calculating the geometry (position and size) of all the elements affected by the change. If an element changes, the geometry of other elements may need to be recalculated. The browser doesn't always need to repaint the entire layer, it tries to be smart about only repainting the part of the DOM that's been invalidated.

The most common cause that forces a repaint is dirtying the DOM by manipulating CSS styles and causing relayout. To achieve smooth animations and maximize the GPU usage, the best way is to change only that CSS properties that affect compositing, like transform and opacity. If only these CSS properties change, it isn't necessary to repaint anything. The browser can just recompose the existing layers that are already sitting on the GPU as textures, but with different compositing properties (e.g. in different positions, with different opacities, etc.).
If part of a layer gets invalidated, it gets repainted and re-uploaded. If its content remains the same but its composited attributes change (e.g. it gets translated or its opacity changes), the browser can leave it on the GPU and recompose to make a new frame.

As should now be clear, the layer-based compositing model has deep implications for rendering performance. Compositing is comparably cheap when nothing needs to be painted, so avoiding repaints of layers is a good overall goal when building applications. It's possible to easily force the creation of layers, defining the DOM structure and separating it into logical graphic layer, i.e. defining a common CSS class .gl, like this

```
.gl {
    -webkit-transform: translate3d(0,0,0);
     transform: translate3d(0,0,0);
}
```

and applying this class to every node that will be animated. Forcing layers to be created ensures both that the layer is painted and ready-to-go as soon as the animation starts (creating and painting a layer is a non-trivial operation and can delay the start of the animation).
But beware just blindly creating too much layers, as they're not free: they take up memory in system RAM and on the GPU and having lots of them can introduce other overhead in the logic keeps track of which are visible. Many layers can also actually increase time spent rasterizing if they layers are large and overlap a lot where they didn't previously, leading to what's sometimes referred to as "overdraw".

As a consequent of the above explanations, please follow these recommendations:

- Show/hide elements with style property **opacity** 0 / 0.99 instead of visibility and display, because they don't use the GPU. Changes to opacity can be handled by the GPU during compositing by simply painting the element texture with a lower alpha value.
- Identify elements that should be animated, and add a common pre-defined CSS class .gl for them, in order to force layer creation at application startup
- Don't manipulate animated elements changing properties like width, height, padding, margin, display border-width, border, top, position, font-size, float, text-align, overflow-y, font-weight, overflow, left, font-family, line-height, vertical-align, right, clearwhite-space, bottom, min-height because they affect layouts
- Don't manipulate animated elements changing properties like color, border-style, visibility, background, text-decoration, background-image, background-position, background-repeat, outline-color, outline, outline-style, border-radius, outline-width, box-shadow, background-size because they cause repaint
- Use absolute position, and use pixel instead of percent in styles. This helps the graphic rendering to avoid further calculation caused by relative positions or dimension changes.

5.8. DOM manipulation

When a browser has to recalculate the positions and geometrics of elements in a document for the purpose of re-rendering it, this is called reflow. Reflow is a user-blocking operation in the browser, so it's helpful to understand how to improve reflow time.
You should batch methods that trigger reflow or that repaint, and use them sparingly. It's important to process off DOM where possible. This is possible using DocumentFragment, a lightweight document object. Think of it as a way to extract a portion of a document's tree, or create a new "fragment" of a document. Rather than constantly adding to the DOM using nodes, we can use document fragments to build up all we need and only perform a single insert into the DOM to avoid excessive reflow.

For example, let's write a function that adds 20 to an element. Simply appending each new div directly to the element could trigger 20 reflows.

```
function addDivs(element) {
  var div;
  for (var i = 0; i < 20; i ++) {
    div = document.createElement('div');
    div.innerHTML = 'Heya!';
    element.appendChild(div);
  }
}
```

To work around this issue, we can use DocumentFragment, and instead, append each of our new divs to this. When appending to the DocumentFragment with a method like appendChild, all of the fragment's children are appended to the element triggering only one reflow.

```
function addDivs(element) {
  var div;
  // Creates a new empty DocumentFragment.
  var fragment = document.createDocumentFragment();
  for (var i = 0; i < 20; i ++) {
    div = document.createElement('a');
    div.innerHTML = 'Heya!';
    fragment.appendChild(div);
  }
  element.appendChild(fragment);
}
```

Creating attached DOM elements is expensive, whereas creating and modifying them while detached and then attaching them yields much better performance.
An ideal way to set up an application is to create at startup all the nodes needed, and then use opacity 0/0.99 to hide/show DOM block or single nodes. When this is not possible, use DocumentFragment and minimize DOM insert and removal and create/destroy tasks.
Also avoid to clone node from existing.

Keep the DOM simple. Applications should be able to work efficiently dealing with small DOM.
Therefore, avoid to build a complex html layout with a lot of nested nodes; most of the times 2 or max 3 nested levels are enough to handle complex scenarios. Remember:

- less nodes make the application's page lightweight and easy to handle.
- less nodes means simple html and simple CSS
- too many nested nodes get the Garbage Collector job complicated, and the risk to generate memory leaks is very high

Last but not least, to foster source code's clearness and readability, the recommendation is to not mix, if possible, style rules into the html, i.e. using style attributes in html nodes. Keep separated the design part from the DOM, use class attributes and always identify isolated block of html subject to animations adding .gl class, as explained in paragraph 5.6.

5.9. XHR

It's recommended to avoid the usage of XHR calls interval above 100ms. In some case this overhead of network usage may cause bad situations.

5.10. App cache

HTML5 provides an application caching mechanism that lets web-based applications run offline. Developers can use the Application Cache (AppCache) interface to specify resources that the browser should cache and make available to offline users. Applications that are cached load and work correctly even if users click the refresh button when they are offline.

Using an application cache gives an application the following benefits:

- Offline browsing: users can navigate a site even when they are offline.
- Speed: cached resources are local, and therefore load faster.
- Reduced server load: the browser only downloads resources that have changed from the server.

In the monitor, the browser App Cache uses a shared pool of TEMPORARY storage that other offline APIs can share, so the cache size limit is about ~10MB. Therefore, due to this limit, please avoid to store videos, images or other big resources that could fill the cache quickly. The App Cache should be used only for store application's files as html, css and js, as well as configuration files (json or xml) and other small resources like fonts.

To enable the application cache for an application, you must include the manifest attribute in the <html> element in your application's pages, as shown in the following example:

```
<html manifest="example.appcache">
  ...
</html>
```

The manifest attribute references a cache manifest file, which is a text file that lists resources (files) that the browser should cache for your application. Obviously, the manifest file must be downloaded with the rest of the application from a remote server or from the USB storage. The manifest attribute in a web application can specify either the relative path of a cache manifest file or an absolute URL. (Absolute URLs must be from the same origin as the application). Usually the file is placed in the root directory of the application
A cache manifest file can have any file extension, but it must be served with the MIME type text/cache-manifest by the web server.

The use of an application cache modifies the normal process of loading a document:

- If an application cache exists, the browser loads the document and its associated resources directly from the cache, without accessing the network. This speeds up the document load time.
- The browser then checks to see if the cache manifest has been updated on the server. This check is performed comparing the old file with the new file. If only one byte is different, a new cache manifest is downloaded and applied. Usually
- If the cache manifest has been updated, the browser downloads a new version of the manifest and the resources listed in the manifest. This is done in the background and does not affect performance significantly.

This is shown and described in detail in the demo called "**Application Cache Events**" in the tools package.

5.10.1. Structure of a manifest file

A simple manifest looks something like this:

```
CACHE MANIFEST
index.html
stylesheet.css
images/logo.png
scripts/main.js
http://www.toshiba.com/scripts/main.js
```

This example will cache four files on the page that specifies this manifest file.

There are a couple of things to note:

- The CACHE MANIFEST string is the first line and is required.
- Files can be from another domain
- If the manifest itself returns a 404 or 410, the cache is deleted.
- If the manifest or a resource specified in it fails to download, the entire cache update process fails. The browser will keep using the old application cache in the event of failure.

This is a more complex example:

```
CACHE MANIFEST
# 2017-02-18:v2

# Explicitly cached 'master entries'.
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js

# Resources that require the user to be online.
NETWORK:
*

# static.html will be served if main.py is inaccessible
# offline.jpg will be served in place of all images in images/large/
# offline.html will be served in place of all other .html files

FALLBACK:
/main.py /static.html
images/large/ images/offline.jpg
```

Lines starting with a '#' are comment lines, but can also serve another purpose. An application's cache is only updated when its manifest file changes. So, for example, if you edit an image resource or change a JavaScript function, those changes will not be re-cached. You must modify the manifest file itself to inform the browser to refresh cached files.

Avoid using a continually-updating timestamp or random string to force updates every time. The manifest is checked twice during an update, once at the start and once after all cached files have been updated. If the manifest has changed during the update, it's possible the browser fetched some files from one version, and other files from another version, so it doesn't apply the cache and retries later.

Although the cache updates, the browser won't use those files until the page is refreshed, because updates happen after the page is loaded from the current version of the cache.

A manifest can have three distinct sections: CACHE, NETWORK, and FALLBACK.

- CACHE: this is the default section for entries. Files listed under this header (or immediately after the CACHE MANIFEST) will be explicitly cached after they're downloaded for the first time.
- NETWORK: files listed in this section may come from the network if they aren't in the cache, otherwise the network isn't used, even if the user is online. You can white-list specific URLs here, or simply "*", which allows all URLs. Most sites need "*".
- FALLBACK: an optional section specifying fallback pages if a resource is inaccessible. The first URI is the resource, the second is the fallback used if the network request fails or errors. Both URIs must from the same origin as the manifest file. You can capture specific URLs but also URL prefixes. "images/large/" will capture failures from URLs such as "images/large/whatever/img.jpg".

These sections can be listed in any order and each section can appear more than one in a single manifest.

5.10.2. Cache status

Once an application is offline it remains cached until one of the following happens:

- FTI or Firmware Update is performed
- The manifest file is modified.
  Note: updating a file listed in the manifest doesn't mean the browser will re-cache that resource. The manifest file itself must be altered.

To know the status of the cache, the window.applicationCache object is the programmatic access the browser's app cache. Its status property is useful for checking the current state of the cache:

```
var appCache = window.applicationCache;

switch (appCache.status) {
  case appCache.UNCACHED: // UNCACHED == 0
    return 'UNCACHED';
    break;
  case appCache.IDLE: // IDLE == 1
    return 'IDLE';
    break;
  case appCache.CHECKING: // CHECKING == 2
    return 'CHECKING';
    break;
  case appCache.DOWNLOADING: // DOWNLOADING == 3
    return 'DOWNLOADING';
    break;
  case appCache.UPDATEREADY:  // UPDATEREADY == 4
    return 'UPDATEREADY';
    break;
  case appCache.OBSOLETE: // OBSOLETE == 5
    return 'OBSOLETE';
    break;
  default:
    return 'UKNOWN CACHE STATUS';
    break;
};
```

5.11. Memory statistics

The browser makes available the following API to monitoring memory usage and availability

- window.performance.memory.jsHeapSizeLimit: is the maximum amount of memory allocable by the browser for JS on the monitor
- window.performance.memory.totalJSHeapSize: current size of the JS heap including free space not occupied by any JS objects.
- window.performance.memory.usedJSHeapSize: the total amount of memory being used by JS

Each API returns the amount of memory in bytes.
From the above description is obvious that usedJsHeapSize cannot be greater than totalJsHeapSize and totalJsHeapSize cannot be great than jsHeapSizeLimit.

These APIs provide the same statistics displayed in the Remote Inspector of the browser (timeline and profiler tabs), and they should be used mainly for monitoring the memory usage during the application execution. In fact, they are the only reliable information available to the application in order to decide when the window.gc() custom API described in paragraph 4.13 should be triggered. In any other case the usage of the gc() API is supposed to be always used in a random and not deterministic way and could cause problems rather than enhancements.
A good way to monitoring memory usage is to observe the ratio between usedJsHeapSize and totalJsHeapSize in relation to the jsHeapSizeLimit. Pay attention that totalJsHeapSize is not a fixed value, it depends on how many objects the application instantiates. Therefore, the memory usage should be always read proportionally to the jsHeapSizeLimit.

5.12. Video Wall synchronization

When you need to chain different monitors in order to create a video wall scenario, you have the option to handle the content synchronization via software rather than through hardware mechanism like Daisy chain.
The synchronization is intended in term of "scenes": a scene is composed by one or more contents, and different scenes synchronize their start time on different monitors.

In the provided tools package, there is available a specific module “es-sync” that makes possible to implement scenes synchronization through different monitors.
The es-sync module strictly cooperates with a socket server integrated in the NodeJs server included in the tools package.
For more details, please take a look at the documentation link “Monitors synchronization” included in the index page documentation of the tools package. Please also check the demo called “**Device Synchronization**”.

5.12.1. Video Offset

When two or more monitors are set up side by side and video synchronization is configured and run, an issue can be noticed: the physical size of the monitor border breaks the continuity of the playout between the videos displayed on each monitor. In this case, an OFFSET parameter can be set using the RS232LanCommand

SETVIDEOWALL RowCount-ColumnCount-Cell-Offset

Example:

SETVIDEOWALL 1-1-1-50

The offset is the size of the video that corresponds to the screen edge size.

Setting the offset makes it possible to have video contents matching their edges on different screens side by side, in order to skip the physical padding due to the monitor borders.
The offset dimension is not pixel-based, there are different offset for different monitor sizes. Please ask to Toshiba for this kind of information.

The offset can be removed setting 0 as the monitor offset. This setting can be changed at run time at any time.

Please also check the demo called "**Device Synchronization with offset**" where this is demonstrated.

5.13. Transition between videos

In order to achieve a perfect switch between two videos avoiding any black gap transition, a specific solution must be adopted. The solution is implemented and described in detail in the demo called "**VideoSwitchDouble**" located in the portfolio folder of the tools package. In case of a single video looping endlessly, please check "**VideoLoopTimeout**" demo in the same tools package's location.

There are four major rules to adhere in order to achieve a perfect video transition with "zero black gap":

1. All video must have the same frame rate. In case of different frame rate, a small black gap will occur, caused by the re-sync of the video decoder
2. Two alternated video tags must be used: one visible tag must play the current video and the other (hidden) tag must load the next video.
3. each video must be first loaded and then played. The "loadedmetadata" event should be used to know when a video is ready to play
4. don't use the "ended" event to get notification about ended video. Ended event is triggered when video is already finished (causing black gap). Instead use setTimeout() with video duration or check video duration listening "canplyathrough" event.

All these rules are well explained in the readme.md files located inside VideoSwitchDouble and VideoLoopTimeout demo in the portfolio folder.

5.14. How to set the STARTURL

To set the starting URL of the monitor's browser there are mainly 3 options:

1. From USB flash drive using OSD menu
2. From USB flash drive using RC's keys combination
3. Using Rs232LanCommand through telnet

5.14.1. From USB flash drive using OSD menu

Please follow these steps:

1. create in a FAT32 formatted USB pen drive, a file named starturl.txt, and type inside this file the new desired URL (e.g.: http://<your_server_host:<your_server_port/some_path/index.html>)
2. plug in the USB pen drive in any USB port available in the back of the monitor
3. push Menu on the remote controller
4. select Signage settings from OSD menu
5. select Link Options
6. select USB Operations
7. wait update operation is done (no feedback is provided)
8. turn off the monitor
9. turn on the monitor
10. the new starting URL is loaded in the browser of the monitor

5.14.2. From USB flash drive using RC's keys combination

Please follow these steps:

1. create in a FAT32 formatted USB pen drive, a file named starturl.txt, and type inside this file the new desired URL (e.g.: http://<your_server_host:<your_server_port/some_path/index.html>)
2. plug in the USB pen drive in any USB port available in the back of the monitor
3. push Menu on the remote controller
4. push 472569 on the remote controller
5. wait update operation is done (no feedback is provided)
6. turn off the monitor
7. turn on the monitor
8. the new starting URL is loaded in the browser of the monitor

5.14.3. Using Rs232LanCommand through Telnet

Please follow these steps:

1. Open a terminal and connect to the monitor with: telnet <monitor_ip> 1986
2. when connection is open, digit SETSTARTURL <your_desired_url>
3. turn off monitor (or give "**TOF"** command from telnet)
4. turn on monitor (or give "**TON on"** command from telnet)

5.15. Firmware update

Applications can trigger a scheduled firmware update through the TSU command, included in the Rs232LanCommand plugin.

If the monitor has been whitelisted on the Toshiba Server and a new firmware version is available, the TSU command searches, downloads and applies immediately the update. When the update id finished, the monitor automatically restarts itself. This operation should be scheduled by the application for a suitable time, when the monitor is not operating.

The command must be invoked like all the other Rs232LanCommand plugin command, i.e.:

```
Rs232LanCommand.execute("TSU")
```

Attaching a listener, you should be able to be notified whether the command is started correctly.

```
Rs232LanCommand.addEventListener("executionResultEvent", function(result){
  console.log(result);
})
```

It should print in console

```
#*Web Software Update search STARTED !!!
```

There is also a different command for firmware upgrade: FSU. It's not very useful because it triggers background updates not performing an immediate update scan. Also, the restart of the monitor should be done manually after the new firmware has been applied and there are no notifications or listeners available to make the application aware of the current status. So please use TSU instead.

5.16. Service workers

Service workers (SW) were born to build offline-first web applications. They are like proxies that sit between the web page and the network, providing cached versions of the resources when no network connectivity is available and giving the ability to intercept network requests and respond to them in different ways.
The service worker script runs in the background making the decision to serve network or cached content.
The older form of browser caching, AppCache, was a more error-prone solution for offline-first ready applications. You can think of service workers as AppCache successor. The service worker API can do what AppCache did, and a whole lot more.
The cache is not solely for offline. It also benefits the user during moments of slow or lowered connectivity. Rather than waiting endless seconds for a resource to load, a previous cache is presented initially.

In the Embedded Signage context, SW are a good solution for:

- make the app working offline
- increase online performance by reducing network requests for certain assets
- provide a customized offline fallback experience
- background syncing

A SW is a file with some JavaScript in it. There are few important things to keep in mind writing this kind of scripts: SW scripts run in a separate thread in the browser from the pages they control. There are ways to communicate between workers and pages, but they execute in a separate scope. That means there is not access to the DOM of those pages, for example. Service workers are not meant to and do not have the ability to manipulate the DOM directly.
Imagine a SW as a sort of script running in a separate tab from the page it affects; this is not at all accurate, but it is a helpful rough metaphor for keeping developers out of confusion.
JavaScript in a SW must not block. Only asynchronous APIs must be used. For example, localStorage cannot be used in a SW (localStorage is a synchronous API).

5.16.1. Lifecycle

A Service Worker has its own lifecycle, which has different steps, each one tied to a specific event (indicated in parentheses):

1. Register('register'): using navigator.serviceWorker.register("serviceWorkerFileName.js") you can register the SW on a browser that supports this API. The command returns a Promise, that you can handle through its .then() and .catch() callbacks.

2. Install('install'): the SW uses this event to open a cache and put in it all the needed resources. The resources are stored using their path. This step will fail if even one resource fails to download. Usually it's used in conjunction with the event.waitUntil() callback to wait for all caching operations to be finished before proceeding.

3. Activate('activate'): this event is fired when the SW starts. Its callback can be used to handle resources caching of a new version of a SW, that has been installed but isn't active yet, usually because an older version of the SW is still active.

4. Fetch('fetch'): this event is used to implement a method for retrieving resources from one of the opened caches (or from the network, if not found locally).

5.16.2. Populating the cache content

To populate a cache, the following command can be used:

```
caches.open('serviceWorkerTest-cache')
    .then(function(cache) {
        cache.add('resource_name');
        cache.add('another_resource_name');
});
```

The cache content isn't automatically updated, this operation must be implemented manually.

5.16.3. Retrieving the cache content

For retrieving a cached resource, the following command can be used:

```
caches.match(event.request)
    .then(function(response) {
        if (!response)
            //Content not found in cache
            return response || fetch(event.request);
        })
        .catch(function(err) {
            //Error
        })
```

5.16.4. Updating an old Service Worker

If you want to create a new version of a SW, you must edit its .js file. The browser will download and install the SW in background. This new version will be kept waiting until all open pages of the application are closed. This can be forced through the command self.skipWaiting(). Use this method with Clients.claim() to ensure that updates to the underlying service worker take effect immediately for both the current client and all other active clients.

5.16.5. Not supported APIs on Chromium v44

As already said, the monitors have Chromium v44 installed. Service Workers API is well supported on version 44, however some small exceptions exist. They are listed below:

cache.addAll()
registration.update()

These APIs are not blocking, there are alternative methods allowing to do the same things. They are just optimization introduced in later releases.

5.16.6. Important notes

Service Workers are domain based. In a single web domain only one SW can run. In case of two or more SW (even operating on different paths) only the first SW installed begins to work, and the other are waiting. Another SW can claim the context, in this case it kills the running SW.

5.17. Video and Audio codec support

The recommended presets for video are the following:

- Codec: h.264
- Container: mp4
- Dimension: same as target monitor (e.g. 1920x1080)

- Bitrate: 30 Mbit
- Framerate: 30fps

Video transcoding can be easily done using ffmpeg. The ffmpeg command to achieve a video with the recommended preset is

ffmpeg -i <original_video> -c:v h264 -r 30 -b:v 30M output.mp4

With the above command Audio presets are kept the same of original video.

In case audio transcoding is needed as well, ffmpeg audio parameter should be used, i.e.

ffmpeg -i <original_video> -c:v h264 -r 30 -b:v 30M -c:a mp3 output.mp4

Below other useful commands to transcode video with some of the supported monitor's video codec

CODEC / Format	**Command**
MPEG1 / mp4	ffmpeg -i <input_video> -c:v mpeg1video -r 30 -b:v 30M mpeg1.mp4
MPEG2 / mp4	ffmpeg -i <input_video> -c:v mpeg2video -r 30 -b:v 30M mpeg2.mp4
MPEG4 / mp4	ffmpeg -i <input_video> -c:v mpeg4 -r 30 -b:v 30M mpeg4.mp4
HEVC (H.265) / mp4	ffmpeg -i <input_video> -c:v hevc -r 30 -b:v 30M h265.mp4
MJPEG / mp4	ffmpeg -i <input_video> -c:v mjpeg -r 30 -b:v 20M mjpeg.mp4
VP8 / mkv	ffmpeg -i <input_video> -c:v vp8 -r 30 -b:v 30M vp8.mkv
VP9 / mkv	ffmpeg -i <input_video> -c:v vp9 -r 30 -b:v 30M vp9.mkv