Mediastream Image Capture

W3C Editor's Draft 14 May 2014

This version:
http://gmandyam.github.com/image-capture
Latest published version:
http://www.w3.org/TR/image-capture/
Latest editor's draft:
http://gmandyam.github.com/image-capture

Editor:
Giridhar Mandyam, Qualcomm Innovation Center, Inc

Abstract

This document specific the takePhoto() and grabFrame() methods, and corresponding camera settings for use with MediaStreams as defined in Media Capture and Streams [GETUSERMEDIA].

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Comments on this document are welcomed.

This document was published by the Web Real-Time Communication Working Group and Device APIs Working Group as an Editor's Draft. If you wish to make comments regarding this document, please send them to public-media-capture@w3.org (subscribe, archives). All comments are welcome.

Publication as an Editor's Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures (Web Real-Time Communication Working Group, Device APIs Working Group) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Introduction

The API defined in this document taks a valid MediaStream and returns an encoded image in the form of a Blob (as defined in [FILE-API]). The image is provided by the capture device that provides the MediaStream. Moreover, picture-specific settings can be optionally provided as arguments that can be applied to the image being captured.

2. Image Capture API

WebIDL

```
[Constructor(VideoStreamTrack track)]
interface ImageCapture : EventTarget {
    readonly    attribute PhotoOptions     photoOptions;
    readonly    attribute VideoStreamTrack videoStreamTrack;
    readonly    attribute MediaStream      previewStream;
                attribute EventHandler     onphoto;
                attribute EventHandler     onerror;
                attribute EventHandler     onoptions;
                attribute EventHandler     onframe;
    void setOptions (PhotoSettings? photoSettings);
    void takePhoto ();
    void grabFrame ();
};
```

2.1 Attributes

`onerror` of type EventHandler,
: Register/unregister for Image Capture error events of type `ImageCaptureErrorEvent`. The handler should expect to get an `ImageCaptureError` object as its first parameter.

`onframe` of type EventHandler,
: Register/unregister for frame capture events of type `FrameGrabEvent`. The handler should expect to get a `FrameGrabEvent` object as its first parameter.

`onoptions` of type EventHandler,
: Register/unregister for photo settings change events of type `SettingsChangeEvent`.

`onphoto` of type EventHandler,
: Register/unregister for photo events of type `BlobEvent`. The handler should expect to get a `BlobEvent` object as its first parameter.

`photoOptions` of type *PhotoOptions*, readonly
: Describes current photo settings

`previewStream` of type MediaStream, readonly
: The MediaStream that provides a camera preview

`videoStreamTrack` of type VideoStreamTrack, readonly
: The MediaStreamTrack passed into the constructor

2.2 Methods

`grabFrame`

When the `grabFrame()` method of an `ImageCapture` object is invoked, then if the `readyState` of the `VideoStreamTrack` provided in the contructor is not "live", the UA MUST fire an `ImageCaptureErrorEvent` event at the `ImageCapture` object with a new `ImageCaptureError` object whose `code` is set to INVALID_TRACK. If the UA is unable to execute the `grabFrame()` method for any other reason, then the UA MUST fire an `ImageCaptureErrorEvent` event at the `ImageCapture` object with a new `ImageCaptureError` object whose `code` is set to FRAME_ERROR. Otherwise it MUST queue a task, using the DOM manipulation task source, that runs the following steps:

1. Gather data from the `VideoStreamTrack` into an `ImageData` object (as defined in [CANVAS-2D]) containing a single still frame in RGBA format. The `width` and `height` of the `ImageData` object are derived from the constraints of the `VideoStreamTrack`.
2. Raise a `FrameGrabEvent` event containing the `ImageData` to the `onframe` event handler (if specified). {Note: `grabFrame()` returns data only once upon being invoked.}

No parameters.
Return type: `void`

`setOptions`

When the `setOptions()` method of an `ImageCapture` object is invoked, then a valid `PhotoSettings` object MUST be passed in the method to the `ImageCapture` object. If the UA can successfully apply the settings, then the UA MUST fire a `SettingsChangeEvent` event at the `onoptions` event handler (if specified). If the UA cannot successfully apply the settings, then the UA MUST fire an `ImageCaptureErrorEvent` at the `ImageCapture` object whose `code` is set to OPTIONS_ERROR.

Parameter	Type	Nullable	Optional	Description
photoSettings	`PhotoSettings`	✓	✗	

Return type: `void`

`takePhoto`

When the `takePhoto()` method of an `ImageCapture` object is invoked, then if the `readyState` of the `VideoStreamTrack` provided in the constructor is not "live", the UA MUST fire an `ImageCaptureErrorEvent` event at the `ImageCapture` object with a new `ImageCaptureError` object whose `code` is set to INVALID_TRACK. If the UA is unable to execute the `takePhoto()` method for any other reason (for example, upon invocation of multiple takePhoto() method calls in rapid succession), then the UA MUST fire an `ImageCaptureErrorEvent` at the `ImageCapture` object with a new `ImageCaptureError` object whose `code` is set to PHOTO_ERROR. Otherwise it MUST queue a task, using the DOM manipulation task source, that runs the following steps:

1. Gather data from the `VideoStreamTrack` into a `Blob` containing a single still image. The method of doing this will depend on the underlying device. Devices may temporarily stop streaming data, reconfigure themselves with the appropriate photo settings, take the photo, and then resume streaming. In this case, the stopping and restarting of streaming SHOULD cause `mute` and `unmute` events to fire on the Track in question.
2. Raise a `BlobEvent` event containing the `Blob` to the `onphoto` event handler (if specified).

No parameters.
Return type: `void`

3. `FrameGrabEvent`

WebIDL

```
[Constructor(DOMString type, optional FrameGrabEventInit frameGrabInitDict)]
interface FrameGrabEvent : Event {
    readonly    attribute ImageData imageData;
};
```

3.1 Attributes

`imageData` of type ImageData, readonly

Returns an `ImageData` object whose `width` and `height` attributes indicates the dimensions of the captured frame.

3.2 `FrameGrabEventInit` Dictionary

WebIDL

```
dictionary FrameGrabEventInit : EventInit {
    ImageData imageData;
};
```

3.2.1 Dictionary `FrameGrabEventInit` Members

`imageData` of type ImageData

An `ImageData` object containing the data to deliver via this event.

4. `ImageCaptureErrorEvent`

WebIDL

```
[Constructor(DOMString type, optional ImageCaptureErrorEventInit imageCaptureErrorInitDict)]
interface ImageCaptureErrorEvent : Event {
    readonly    attribute ImageCaptureError imageCaptureError;
};
```

4.1 Attributes

`imageCaptureError` of type *ImageCaptureError*, readonly
: Returns an `ImageCaptureError` object whose `code` attribute indicates the type of error occurrence.

4.2 `ImageCaptureErrorEventInit` Dictionary

WebIDL

```
dictionary ImageCaptureErrorEventInit : EventInit {
    ImageCaptureError imageCaptureError;
};
```

4.2.1 Dictionary `ImageCaptureErrorEventInit` Members

`imageCaptureError` of type *ImageCaptureError*
: an `ImageCaptureError` object containing the data to deliver via this event.

5. `BlobEvent`

WebIDL

```
[Constructor(DOMString type, optional BlobEventInit blobInitDict)]
interface BlobEvent : Event {
    readonly    attribute Blob data;
};
```

5.1 Attributes

`data` of type Blob, readonly
: Returns a `Blob` object whose type attribute indicates the encoding of the blob data. An implementation must return a Blob in a format that is capable of being viewed in an HTML `<img>` tag.

5.2 `BlobEventInit` Dictionary

WebIDL

```
dictionary BlobEventInit : EventInit {
    Blob data;
};
```

5.2.1 Dictionary `BlobEventInit` Members

`data` of type Blob
: A `Blob` object containing the data to deliver via this event.

6. `SettingsChangeEvent`

WebIDL

```
[Constructor(DOMString type, optional SettingsChangeEventInit photoSettingsInitDict)]
interface PhotoSettingsEvent : Event {
    readonly    attribute PhotoSettings photoSettings;
};
```

6.1 Attributes

`photoSettings` of type *PhotoSettings*, readonly
: Returns a `PhotoSettings` object whose type attribute indicates the current photo settings.

6.2 SettingsChangeEventInit Dictionary

```
WebIDL

dictionary SettingsChangeEventInit : EventInit {
    PhotoSettings photoSettings;
};
```

6.2.1 Dictionary SettingsChangeEventInit Members

`photoSettings` of type *PhotoSettings*
    A `PhotoSettings` object containing the data to deliver via this event.

7. ImageCaptureError

The `ImageCaptureError` object is passed to an `onerror` event handler of an `ImageCapture` object if an error occurred when the object was created or any of its methods were invoked.

```
WebIDL

[NoInterfaceObject]
interface ImageCaptureError {
    const unsigned short FRAME_ERROR = 1;
    const unsigned short OPTIONS_ERROR = 2;
    const unsigned short PHOTO_ERROR = 3;
    const unsigned short ERROR_UNKNOWN = 4;
    readonly    attribute unsigned short code;
    readonly    attribute DOMString      message;
};
```

7.1 Attributes

`code` of type unsigned short, readonly
    The `code` attribute returns the appropriate code for the error event, derived from the constants defined in the `ImageCaptureError` interface.

`message` of type DOMString, readonly
    The `message` attribute must return an error message describing the details of the error encountered.

7.2 Constants

`ERROR_UNKNOWN` of type unsigned short
    An `ImageCaptureError` object must set its `code` value to this constant if an error occurred due to indeterminate cause upon invocation of any method of the `ImageCapture` interface.

`FRAME_ERROR` of type unsigned short
    An `ImageCaptureError` object must set its `code` value to this constant if an error occurred upon invocation of the `grabFrame()` method of the `ImageCapture` interface.

`OPTIONS_ERROR` of type unsigned short
    An `ImageCaptureError` object must set its `code` value to this constant if an error occurred upon invocation of the `setOptions()` method of the `ImageCapture` interface.

`PHOTO_ERROR` of type unsigned short
    An `ImageCaptureError` object must set its `code` value to this constant if an error occurred upon invocation of the `takePhoto()` method of the `ImageCapture` interface.

8. MediaSettingsRange

```
WebIDL

interface MediaSettingsRange {
    readonly    attribute unsigned long max;
    readonly    attribute unsigned long min;
    readonly    attribute unsigned long initial;
};
```

8.1 Attributes

`initial` of type unsigned long, readonly
    The current value of this setting

max of type unsigned long, readonly
The maximum value of this setting

min of type unsigned long, readonly
The minimum value of this setting

9. MediaSettingsItem

The MediaSettingsItem interface is now defined, which allows for a single setting to be managed.

WebIDL

```
interface MediaSettingsItem {
    readonly    attribute any value;
};
```

9.1 Attributes

value of type any, readonly
Value of current setting.

10. PhotoOptions

The PhotoOptions attribute of the ImageCapture object provides the photo-specific settings options and current settings values. The following definitions are assumed for individual settings and are provided for information purposes:

1. *Autofocus* is a setting that enables the camera hardware to automatically focus on a selected part of the imaging area.
2. *White balance mode* is a setting that cameras use to adjust for different color temperatures. Color temperature is the temperature of background light (measured in Kelvin normally). This setting can also be automatically determined by the implementation. If 'automatic' mode is selected, then the Kelvin setting for White Balance Mode may be overridden. Typical temprature ranges for different modes are provided below:

Mode	Kelvin range
incandescent	2500-3500
fluorescent	4000-5000
warm-fluorescent	5000-5500
daylight	5500-6500
cloudy-daylight	6500-8000
twilight	8000-9000
shade	9000-10000

3. *Exposure* is the amount of light allowed to fall on the photographic medium. Auto-exposure mode is a camera setting where the exposure levels are automatically adjusted by the implementation based on the subject of the photo.
4. *Exposure Compensation* is a numeric camera setting that adjusts the exposure level from the current value used by the implementation. This value can be used to bias the exposure level enabled by auto-exposure.
5. The *ISO* setting of a camera describes the sensistivity of the camera to light. It is a numeric value, where the lower the value the greater the sensitivity. This setting in most implementations relates to shutter speed, and is sometimes known as the ASA setting.
6. *Red Eye Reduction* is a feature in cameras that is designed to limit or prevent the appearance of red pupils ("Red Eye") in photography subjects due prolonged exposure to a camera's flash.
7. *Brightness* refers to the numeric camera setting that adjusts the perceived amount of light emitting from the photo object. A higher brightness setting increases the intensity of darker areas in a scene while compressing the intensity of brighter parts of the scene.
8. *Contrast* is the numeric camera setting that controls the difference in brightness between light and dark areas in a scene. A higher contrast setting reflects an expansion in the difference in brightness.
9. *Saturation* is a numeric camera setting that controls the intensity of color in a scene (i.e. the amount of gray in the scene). Very low saturation levels will result in photo's closer to black-and-white.
10. *Sharpness* is a numeric camera setting that controls the intensity of edges in a scene. Higher sharpness settings result in higher edge intensity, while lower settings result in less contrast and blurrier edges (i.e. soft focus).
11. *Zoom* is a numeric camera setting that controls the focal length of the lens. The setting usually represents a ratio, e.g. 4 is a zoom ratio of 4:1. The minimum value is usually 1, to represent a 1:1 ratio (i.e. no zoom).

WebIDL

```
interface PhotoOptions {
                attribute MediaSettingsItem  autoWhiteBalanceMode;
                attribute MediaSettingsRange whiteBalanceMode;
                attribute ExposureMode       autoExposureMode;
                attribute MediaSettingsRange exposureCompensation;
                attribute MediaSettingsRange iso;
                attribute MediaSettingsItem  redEyeReduction;
                attribute MediaSettingsRange brightness;
                attribute MediaSettingsRange constrast;
                attribute MediaSettingsRange saturation;
                attribute MediaSettingsRange sharpness;
```

```
                    attribute MediaSettingsRange imageHeight;
                    attribute MediaSettingsRange imageWidth;
                    attribute MediaSettingsRange zoom;
                    attribute boolean            autofocus;
};
```

10.1 Attributes

`autoExposureMode` of type ExposureMode,
: This reflects the current auto exposure mode setting. Values are of type `ExposureMode`.

`autoWhiteBalanceMode` of type *MediaSettingsItem*,
: This reflects whether automated White Balance Mode selection is on or off, and is boolean - on is true

`autofocus` of type boolean,
: This reflects the current autofocus setting. *false* means autofocus is disabled.

`brightness` of type *MediaSettingsRange*,
: This reflects the current brightness setting of the camera and permitted range. Values are numeric.

`constrast` of type *MediaSettingsRange*,
: This reflects the current contrast setting of the camera and permitted range. Values are numeric.

`exposureCompensation` of type *MediaSettingsRange*,
: This reflects the current exposure compensation setting and permitted range. Values are numeric.

`imageHeight` of type *MediaSettingsRange*,
: This reflects the image height range supported by the UA and the current height setting.

`imageWidth` of type *MediaSettingsRange*,
: This reflects the image width range supported by the UA and the current width setting.

`iso` of type *MediaSettingsRange*,
: This reflects the current camera ISO setting and permitted range. Values are numeric.

`redEyeReduction` of type *MediaSettingsItem*,
: This reflects whether camera red eye reduction is on or off, and is boolean - on is true

`saturation` of type *MediaSettingsRange*,
: This reflects the current saturation setting of the camera and permitted range. Values are numeric.

`sharpness` of type *MediaSettingsRange*,
: This reflects the current sharpness setting of the camera and permitted range. Values are numeric.

`whiteBalanceMode` of type *MediaSettingsRange*,
: This reflects the current white balance mode setting. Values are of type `WhiteBalanceModeEnum`.

`zoom` of type *MediaSettingsRange*,
: This reflects the zoom value range supported by the UA and the current zoom setting.

11. `ExposureMode`

WebIDL

```
enum ExposureModeEnum {
    "frame-average",
    "center-weighted",
    "spot-metering"
};
```

Enumeration description

`frame-average`	Average of light information from entire scene
`center-weighted`	Sensitivity concentrated towards center of viewfinder
`spot-metering`	Spot-centered weighting

12. `PhotoSettings`

The `PhotoSettings` object is optionally passed into the `ImageCapture.setOptions()` method in order to modify capture device settings specific to still imagery. Each of the attributes in this object are optional.

WebIDL

```
dictionary PhotoSettings {
    attribute boolean       autoWhiteBalanceMode;
    attribute unsigned long whiteBalanceMode;
    attribute any           autoExposureMode;
    attribute unsigned long exposureCompensation;
    attribute unsigned long iso;
    attribute boolean       redEyeReduction;
    attribute unsigned long brightness;
    attribute unsigned long constrast;
    attribute unsigned long saturation;
    attribute unsigned long sharpness;
    attribute unsigned long imageHeight;
    attribute unsigned long imageWidth;
    attribute unsigned long zoom;
    attribute boolean       autofocus;
};
```

12.1 Dictionary `PhotoSettings` Members

`autoExposureMode` of type attribute any
: This reflects the desired auto exposure mode setting. Acceptable values are of type `ExposureModeEnum`.

`autoWhiteBalanceMode` of type attribute boolean
: This reflects whether automatic White Balance Mode selection is desired.

`autofocus` of type attribute boolean
: This reflects the desired autofocus setting.

`brightness` of type attribute unsigned long
: This reflects the desired brightness setting of the camera.

`constrast` of type attribute unsigned long
: This reflects the desired contrast setting of the camera.

`exposureCompensation` of type attribute unsigned long
: This reflects the desired exposure compensation setting.

`imageHeight` of type attribute unsigned long
: This reflects the desired image height.

`imageWidth` of type attribute unsigned long
: This reflects the desired image width.

`iso` of type attribute unsigned long
: This reflects the desired camera ISO setting.

`redEyeReduction` of type attribute boolean
: This reflects whether camera red eye reduction is desired

`saturation` of type attribute unsigned long
: This reflects the desired saturation setting of the camera.

`sharpness` of type attribute unsigned long
: This reflects the desired sharpness setting of the camera.

`whiteBalanceMode` of type attribute unsigned long
: This reflects the desired white balance mode setting.

`zoom` of type attribute unsigned long
: This reflects the desired zoom value.

13. Promise Extensions to `ImageCapture`

If the User Agent supports *Promises*, then the following may be used. Any *Promise* object is assumed to have *resolver* object, with *resolve()* and *reject()* methods associated with it. {NOTE: The `setOptions()` method is not recast as a *Promise* due to the possibility that its associated event handler `onoptions` may be repeatedly invoked.}

WebIDL

```
[Constructor(VideoStreamTrack track)]
interface ImageCapture : EventTarget {
    readonly    attribute PhotoOptions     photoOptions;
    readonly    attribute VideoStreamTrack videoStreamTrack;
    readonly    attribute MediaStream      previewStream;
                attribute EventHandler     onerror;
                attribute EventHandler     onoptions;
    void    setOptions (PhotoSettings? photoSettings);
    Promise takePhoto ();
    Promise grabFrame ();
};
```

13.1 Attributes

`onerror` of type EventHandler,
: Register/unregister for Image Capture error events of type `ImageCaptureErrorEvent`. The handler should expect to get an `ImageCaptureError` object as its first parameter.

`onoptions` of type EventHandler,
: Register/unregister for photo settings change events of type `SettingsChangeEvent`.

`photoOptions` of type *PhotoOptions*, readonly
: Describes current photo settings

`previewStream` of type MediaStream, readonly
: The MediaStream that provides a camera preview

`videoStreamTrack` of type VideoStreamTrack, readonly
: The MediaStreamTrack passed into the constructor

13.2 Methods

`grabFrame`

When the `grabFrame()` method of an `ImageCapture` object is invoked, a new *Promise* object is returned. If the `readyState` of the `VideoStreamTrack` provided in the contructor is not "live", the UA MUST return an `ImageCaptureErrorEvent` event to the *resolver* object's *reject()* method with a new `ImageCaptureError` object whose `code` is set to INVALID_TRACK. If the UA is unable to execute the `grabFrame()` method for any other reason, then the UA MUST return an `ImageCaptureErrorEvent` event to the *resolver* object's *reject()* method with a new `ImageCaptureError` object whose `code` is set to FRAME_ERROR. Otherwise it MUST queue a task, using the DOM manipulation task source, that runs the following steps:

1. Gather data from the `VideoStreamTrack` into an `ImageData` object (as defined in [CANVAS-2D]) containing a single still frame in RGBA format. The `width` and `height` of the `ImageData` object are derived from the constraints of the `VideoStreamTrack`.
2. Return a `FrameGrabEvent` event containing the `ImageData` to the *resolver* object's *resolve()* method. {Note: `grabFrame()` returns data only once upon being invoked.}

No parameters.
Return type: `Promise`

`setOptions`

When the `setOptions()` method of an `ImageCapture` object is invoked, then a valid `PhotoSettings` object MUST be passed in the method to the `ImageCapture` object. If the UA can successfully apply the settings, then the UA MUST fire a `SettingsChangeEvent` event at the `onoptions` event handler (if specified). If the UA cannot successfully apply the settings, then the UA MUST fire an `ImageCaptureErrorEvent` at the `ImageCapture` object whose `code` is set to OPTIONS_ERROR.

Parameter	Type	Nullable	Optional	Description
photoSettings	**`PhotoSettings`**	✓	✗	

Return type: `void`

`takePhoto`

When the `takePhoto()` method of an `ImageCapture` object is invoked, a new *Promise* object is returned. If the `readyState` of the `VideoStreamTrack` provided in the constructor is not "live", the UA MUST return an `ImageCaptureErrorEvent` event to the *resolver* object's *reject()* method with a new `ImageCaptureError` object whose `code` is set to INVALID_TRACK. If the UA is unable to execute the `takePhoto()` method for any other reason (for example, upon invocation of multiple takePhoto() method calls in rapid succession), then the UA MUST return an `ImageCaptureErrorEvent` event to the *resolver* object's *reject()* method with a new `ImageCaptureError` object whose `code` is set to PHOTO_ERROR. Otherwise it MUST queue a task, using the DOM manipulation task source, that runs the following steps:

1. Gather data from the `VideoStreamTrack` into a `Blob` containing a single still image. The method of doing this will depend on the underlying device. Devices may temporarily stop streaming data, reconfigure themselves with the appropriate photo settings, take the photo, and then resume streaming. In this case, the stopping and restarting of streaming SHOULD cause `mute` and `unmute` events to fire on the Track in question.
2. Return a `BlobEvent` event containing the `Blob` to the *resolver* object's *resolve()* method.

No parameters.
Return type: `Promise`

14. Examples

14.1 Taking a picture if Red Eye Reduction is activated

EXAMPLE 1

```
navigator.getUserMedia({video: true}, gotMedia, failedToGetMedia);

  function gotMedia(mediastream) {
     //Extract video track.
     var videoDevice = mediastream.getVideoTracks()[0];
     // Check if this device supports a picture mode...
     var captureDevice = new ImageCapture(videoDevice);
     if (captureDevice) {
           captureDevice.onphoto = showPicture;
           if (captureDevice.photoOptions.redEyeReduction) {
              captureDevice.setOptions({redEyeReductionSetting:true});
              }
           else
              console.log('No red eye reduction');
           captureDevice.onoptions = function(){
              if (captureDevice.photoOptions.redEyeReduction.value)
                  captureDevice.takePhoto();
              }
           }
       }

   function showPicture(e) {
      var img = document.querySelector("img");
      img.src = URL.createObjectURL(e.data);
      }

   function failedToGetMedia{
      console.log('Stream failure');
      }
```

14.2 Grabbing a Frame for Post-Processing

EXAMPLE 2

```
navigator.getUserMedia({video: true}, gotMedia, failedToGetMedia);

  function gotMedia(mediastream) {
     //Extract video track.
     var videoDevice = mediastream.getVideoTracks()[0];
     // Check if this device supports a picture mode...
     var captureDevice = new ImageCapture(videoDevice);
     if (captureDevice) {
           captureDevice.onframe = processFrame;
           captureDevice.grabFrame();
           }
       }

   function processFrame(e) {
      imgData = e.imageData;
      width = imgData.width;
      height = imgData.height;
      for (j=3; j < imgData.width; j+=4)
           {
           // Set all alpha values to medium opacity
           imgData.data[j] = 128;
           }
      // Create new ImageObject with the modified pixel values
      var canvas = document.createElement('canvas');
      ctx = canvas.getContext("2d");
      newImg = ctx.createImageData(width,height);
      for (j=0; j < imgData.width; j++)
           {
           newImg.data[j] = imgData.data[j];
           }
      // ... and do something with the modified image ...
      }

   function failedToGetMedia{
      console.log('Stream failure');
      }
```

14.3 Repeated grabbing of a frame

EXAMPLE 3

```
<html>
  <body>
  <p><canvas id="frame"></canvas></p>
```

```
    <button onclick="stopFunction()">Stop frame grab</button>
    <script>
    var canvas = document.getElementById('frame');
    navigator.getUserMedia({video: true}, gotMedia, failedToGetMedia);

    function gotMedia(mediastream) {
       //Extract video track.
       var videoDevice = mediastream.getVideoTracks()[0];
       // Check if this device supports a picture mode...
       var captureDevice = new ImageCapture(videoDevice);
       var frameVar;
       if (captureDevice) {
             captureDevice.onframe = processFrame;
             frameVar = setInterval(captureDevice.grabFrame(), 1000);
             }
        }

     function processFrame(e) {
         imgData = e.imageData;
         canvas.width = imgData.width;
         canvas.height = imgData.height;
         canvas.getContext('2d').drawImage(imgData, 0, 0,imgData.width,imgData.height);
         }

     function stopFunction(e) {
         clearInterval(myVar);
         }
    </script>
    </body>
    </html>
```

14.4 Taking a picture if Red Eye Reduction is activated using promises

EXAMPLE 4

```
  navigator.getUserMedia({video: true}, gotMedia, failedToGetMedia);

    function gotMedia(mediastream) {
       //Extract video track.
       var videoDevice = mediastream.getVideoTracks()[0];
       // Check if this device supports a picture mode...
       var captureDevice = new ImageCapture(videoDevice);
       if (captureDevice) {
             if (captureDevice.photoOptions.redEyeReduction) {
                captureDevice.setOptions({redEyeReductionSetting:true});
                }
             else
                console.log('No red eye reduction');
             captureDevice.onoptions = function(){
                if (captureDevice.photoOptions.redEyeReduction.value)
                    captureDevice.takePhoto().then(showPicture(blob),function(error){alert("Failed to take photo");});
                }
             }
        }

     function showPicture(e) {
        var img = document.querySelector("img");
        img.src = URL.createObjectURL(e.data);
        }

     function failedToGetMedia{
        console.log('Stream failure');
        }
```

A. References

A.1 Normative references

[CANVAS-2D]
Rik Cabanier; Eliot Graff; Jay Munro; Tom Wiltzius; Ian Hickson. *HTML Canvas 2D Context*. 6 August 2013. W3C Candidate Recommendation. URL: http://www.w3.org/TR/2dcontext/

[FILE-API]
Arun Ranganathan; Jonas Sicking. *File API*. 12 September 2013. W3C Last Call Working Draft. URL: http://www.w3.org/TR/FileAPI/

[GETUSERMEDIA]
Daniel Burnett; Adam Bergkvist; Cullen Jennings; Anant Narayanan. *Media Capture and Streams*. 3 September 2013. W3C Working Draft. URL: http://www.w3.org/TR/mediacapture-streams/