



Recommended Revisions to the W3C Javascript Media Capture API

August 19, 2012

**Qualcomm Innovation Center, Inc.
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.**

**Qualcomm Innovation Center, Inc.
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.**

**Copyright © 2012 Qualcomm Innovation Center, Inc.
All rights not expressly granted are reserved.**

Redistribution and use in source form and compiled forms (SGML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

Redistributions in source form must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

This documentation is provided by Qualcomm Innovation Center, Inc. "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and non-infringement are disclaimed. In no event shall Qualcomm Innovation Center, Inc. be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this documentation, even if advised of the possibility of such damage.

This technical data may be subject to U.S. and international export, re-export or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Other product and brand names may be trademarks or registered trademarks of their respective owners.

Contents

1 Chapter 1	4
1.1 Purpose	4
1.2 References	4
2 Camera Capabilities	5
2.1 getUserMedia() Method Modifications.....	6
2.2 LocalMediaStream Modifications	7
2.3 Sample Code.....	9

1 Chapter 1

The World Wide Web Consortium (W3C) is the most widely-recognized industry standards development organization (SDO) that focuses on web technologies. There are several Working Groups (WG's) that exist in the W3C. The focus of this document is the ongoing activity in the W3C Media Capture Task Force, which is a joint effort of the W3C Devices API (DAP) and Web Real Time Communications (WebRTC) Working Groups. Currently, the primary specification for a Javascript enabler for media capture is entitled "getUserMedia: Getting access to local devices that can generate multimedia streams." This specification is considered necessary in order to implement local media capture that may be used for two-way real time communication between browsers (which is covered in the WebRTC Working Group's deliverables). However, the current specification as written does not necessarily consider advanced camera features beyond the standard web camera (which is generally not considered as capable an image capture device as the cameras integrated with most mobile devices). The API changes proposed in this document are meant to ensure that the getUserMedia specification can take full advantage of all functionality associated with image capture on advanced handheld devices.

1.1 Purpose

This document is intended to be used as input into the W3C Media Capture Task Force for potential changes to the getUserMedia API.

1.2 References

The following are reference documents:

- *getUserMedia: Getting access to local devices that can generate multimedia streams*. W3C Editor's Draft. June 25, 2012. <http://dev.w3.org/2011/webrtc/editor/getusermedia.html>.
- *WebRTC 1.0: Real-Time Communication Between Browsers*. W3C Editor's Draft. April 27, 2012. <http://dev.w3.org/2011/webrtc/editor/webrtc.html>.

2 Camera Capabilities

The media capture capability encompassed in the current `getUserMedia` definition was primarily targeted to complete the necessary functionality for the WebRTC specification, and was originally defined in the WebRTC specification. Near the end of 2011, `getUserMedia` specification work was removed from the W3C WebRTC Working Group and handed to the W3C Media Capture Task Force. Nevertheless, `getUserMedia` must still satisfy the basic requirements of WebRTC, which are for the ability to create streams between the browser and local capture devices (microphone, camera, etc.) that are suitable to be added on to a `peerConnection` (the Javascript object that represents a UDP socket connection between browsers). Moreover, although WebRTC does not exclude mobile browser implementations, the initial target is desktop browsers. Personal computers generally do not have sophisticated cameras. Smartphone cameras, which come with a suite of more sophisticated user controls, are not well-represented by the current `getUserMedia` specification. As a result, the still image capture scenario is not well-addressed currently by `getUserMedia`.

`getUserMedia` starts with the definition of the `MediaStream` object. The `MediaStream` object represents streams of audio and/or video content. The source of these streams is not specified. The WebIDL description of the interface is

```
interface MediaStream {
    readonly attribute DOMString          label;
    readonly attribute MediaStreamTrackList audioTracks;
    readonly attribute MediaStreamTrackList videoTracks;
    attribute boolean                     ended;
    attribute function?                   onEnded;
};
```

Each `MediaStream` object is assigned a unique *label* by the browser. The object is a grouping of audio and video tracks of type `MediaStreamTrackList`. A `MediaStream` object can be queried to see if the stream has ended based on the *ended* attribute, and a function can be defined as a handler when the *onEnded* event occurs.

A *LocalMediaStream* is a `MediaStream` object with a `stop()` method defined. It is meant to represent a `MediaStream` originating from a local device.

A `MediaStreamTrack` object is the browser representation of a media source. It is defined as

```
interface MediaStreamTrack {
    readonly attribute DOMString          kind;
    readonly attribute DOMString          label;
    attribute boolean                     enabled;
    const unsigned short LIVE = 0;
    const unsigned short MUTED = 1;
    const unsigned short ENDED = 2;
    readonly attribute unsigned short readyState;
    attribute Function?                  onmute;
    attribute Function?                  onunmute;
    attribute Function?                  onended;
};
```

The *kind* attribute refers to the nature of the `MediaStreamTrack` and can be “audio” or “video”. The *label* is an optional identifier (e.g. “Internal microphone”) assigned by the browser for the `MediaStreamTrack`. A `MediaStreamTrack` can be enabled or disabled through the *enabled* attribute. The *readyState* attribute can have the integer value representations for LIVE, MUTED and ENDED. Finally, handler functions can be defined for the events *onmute*, *onunmute* and *onended*.

The interface `MediaStreamTrackList` that is part of a `MediaStream` object is meant to define one or more audio and/or video tracks (`MediaStreamTrack`'s) associated with the `MediaStream` content:

```
interface MediaStreamTrackList {
    readonly attribute unsigned long          length;
    MediaStreamTrack item (unsigned long index);
    void add (MediaStreamTrack track);
    void remove (MediaStreamTrack track);
    attribute Function?                       onaddtrack;
    attribute Function?                       onremovetrack;
};
```

length is the number of tracks in the `MediaStreamTrackList`. *item* returns the `MediaStreamTrack` at the given index indicated by this value. *add()* and *remove()* are methods defined on `MediaStreamTrackList` to add or remove specific `MediaStreamTracks`. Finally, handler functions can be defined for the events *onaddtrack* and *onremovetrack*.

Given that `MediaStream` and `LocalMediaStream` are now defined, the interface *getUserMedia()* can also be specified with respect to these two objects. The formal interface is `NavigatorUserMedia`:

```
interface NavigatorUserMedia {
    void getUserMedia (MediaStreamConstraints? constraints,
        NavigatorUserMediaSuccessCallback? successCallback, optional
        NavigatorUserMediaErrorCallback? errorCallback);
};
```

The *getUserMedia()* method takes a *MediaStreamOptions* parameter that encompasses *audio* and *video* boolean variables (i.e. an audio-only stream would just define options as `audio:true`). *getUserMedia()* passes a `LocalMediaStream` object to the success callback function. The error code of “1”, which represents “Permission Denied”, is currently all that is passed to the error callback function.

2.1 *getUserMedia()* Method Modifications

Since mobile devices can have more than one camera, it is proposed that the `MediaStreamConstraints` dictionary remain as is:

```
dictionary MediaStreamOptions {
    (boolean or MediaTrackConstraints) audio;
    (boolean or MediaTrackConstraints) video;
};
```

The `MediaTrackConstraints` dictionary will continue to be leveraged, with modifications:

```
dictionary MediaTrackConstraints {
    MediaTrackConstraintSet? mandatory = null;
```

```
MediaTrackConstraintSet[]? optional = null;
};
```

Firstly, `MediaTrackConstraints` should be leveraged to specify the target device. For instance, on portable devices with two cameras, there must be a constraint that specifies which camera will be the source of the `localMediaStream`. One attribute, `cameraSelect`, is proposed with valid values of 'front' and 'back'. If it is listed among the mandatory `MediaTrackConstraints` and the specified camera is not available, then creation of the stream will fail. If it is specified as part of the optional constraints and the specified camera is not available, then the stream can still be created to the device's only camera if it exists. Valid `MediaStreamConstraints` are still being defined in the W3C, but additional constraints have been proposed such as (min-) `maxBandwidth`, (min-) `maxHeight`, (min-) `maxWidth`, and (min-) `maxFrameRate`. One example of a `getUserMedia` call would be

```
navigator.getUserMedia({video:{mandatory:{minHeight:320,minWidth:640,cameraSelect:'front'},optional:[{minFrameRate:10}]}, successCallback, errorCallback);
```

In addition, a new error code is defined to be passed to the error callback function:

```
interface NavigatorUserMediaError {
  const unsigned short PERMISSION_DENIED = 1;
  const unsigned short CAMERA_UNAVAILABLE = 2;
  readonly attribute unsigned short code;
};
```

The `CAMERA_UNAVAILABLE` error code will be returned if the requested `LocalMediaStream` object cannot be created due to the lack of access to the specified camera if the `camera_select` attribute in the `MediaStreamConstraints` is mandatory. This could occur on handheld single camera devices if the `MediaStreamOption` is set to *video* (implying a stream to a non-existent front-facing camera). The preferred behavior of the user agent in the case of this condition is to not query the end user for camera access permission – therefore there can be no situations when the requested camera is unavailable where the user can explicitly deny permission.

2.2 LocalMediaStream Modifications

A new method to enable image capture, `captureImage()` is defined for the `LocalMediaStream` object:

```
interface LocalMediaStream : MediaStream {
  void stop ();
  void captureImage(captureImageSuccessCallback? successCallback, captureImageErrorCallback? errorCallback);
  void getFeatureKeyRange(getFeatureKeyRangeSuccessCallback? successCallback, getFeatureKeyRangeErrorCallback? errorCallback);
  void getFeatureKeys(FeatureKeys? feature_keys, getFeatureKeysSuccessCallback? successCallback, getFeatureKeysErrorCallback? errorCallback);
  void setFeatureKeys(FeatureKeys? feature_keys, setFeatureKeysSuccessCallback? successCallback, setFeatureKeysErrorCallback? errorCallback);
};
callback captureImageSuccessCallback =
```

```

    void (ArrayBuffer array);
callback captureImageErrorCallback =
    void (captureImageError image_error);
callback getFeatureKeyRangeSuccessCallback =
    void (FeatureKeyRangeArray range_array);
callback getFeatureKeyRangeErrorCallback =
    void ();
callback getFeatureKeysSuccessCallback =
    void ();
callback getFeatureKeysErrorCallback =
    void ();
callback setFeatureKeysSuccessCallback =
    void ();
callback setFeatureKeysErrorCallback =
    void ();

interface captureImageError {
    const DOMString type_mismatch_err = "TYPE_MISMATCH_ERR";
    const DOMString range_err = "RANGE_ERR";
    const DOMString invalid_values_err = "INVALID_VALUES_ERR";
    const DOMString not_found_err = "NOT_FOUND_ERR";
    const DOMString security_err = "SECURITY_ERR";
    const DOMString unknown_err = "UNKNOWN_ERR";
    const DOMString io_err = "IO_ERR";
    const DOMString stream_close_err = "STREAM_CLOSE_ERR";
    const DOMString camera_paused_err = "CAMERA_PAUSED_ERR";
    const DOMString camera_not_ready_err = "CAMERA_NOT_READY_ERR";
    const DOMString camera_capture_err = "CAMERA_CAPTURE_ERR";
    const DOMString camera_connect_err = "CAMERA_CONNECT_ERR";
    const DOMString camera_preview_err = "CAMERA_PREVIEW_ERR";
    const DOMString camera_feature_err = "CAMERA_FEATURE_ERR";
    const DOMString camera_info_err = "CAMERA_INFO_ERR";
    readonly attribute DOMString;
};

```

`captureImage()` will return a binary `ArrayBuffer` to the success callback function. The error codes for capture image are currently undefined. When invoking `getFeatureKeyRange` on a `LocalMediaStream`, the returned associative array is of type *FeatureKeyRangeArray*. It returns the settings options available for various feature keys.

```

dictionary FeatureKeyRangeArray {
    readonly DOMSTRING[]    aspect_ratio_settings;
    readonly double[]       brightness_range;
    readonly double[]       contrast_range;
    readonly DOMSTRING[]    effects_options;
    readonly boolean        face_detection_available;
    readonly DOMSTRING[]    flash_settings;
    readonly DOMSTRING[]    focus_settings;
    readonly boolean        geotagging_available;
    readonly boolean        hand_jitter_reduction_available;
    readonly boolean        high_dynamic_range_available;
    readonly boolean        image_stablization_available;
    readonly DOMSTRING[]    iso_settings;
    readonly DOMSTRING[]    quality_settings;
};

```

```

    readonly double[] resolution_video_range;
    readonly double[] resolution_picture_range;
    readonly double[] saturation_range;
    readonly DOMString[] self_timer_settings;
    readonly double[] sharpness_range;
    readonly boolean shutter_sound_available;
    readonly boolean skin_color_enhancement_available;
    readonly boolean timestamp_available;
    readonly boolean white_balance_mode_available;
    readonly boolean zero_shutter_lag_available;
    readonly double[] zoom_range;
    readonly double[] zoom_level_digital_range;
};

```

`getFeatureKeys` and `setFeatureKeys` refer to operations on an array of `FeatureKeys`. `FeatureKeys` represent features of the camera that are settable, and the values must be within the acceptable settings provided in the `FeatureKeyRangeArray` returned from the `getFeatureKeyRange` call:

```

dictionary FeatureKeys {
    DOMSTRING aspectRatio;
    double brightness;
    double contrast;
    DOMSTRING effects;
    boolean faceDetection;
    DOMSTRING flash;
    DOMSTRING focus;
    boolean geotagging;
    boolean handJitterReduction;
    boolean highDynamicRange;
    boolean imageStablization;
    DOMSTRING iso;
    DOMSTRING quality;
    double resolutionVideo;
    double resolutionPicture;
    double saturation;
    DOMSTRING selfTimer;
    double sharpness;
    boolean shutterSound;
    boolean skinColorEnhancement;
    boolean timestamp;
    boolean whiteBalanceMode;
    boolean zeroShutterLag;
    double zoom;
    double zoomLevelDigital;
};

```

The error codes for `getFeatureKeys` and `setFeatureKeys` is TBD.

2.3 Sample Code

The following code displays a preview image next to a capture image.

```

...
<section id="splash">
    <p id="errorMessage">Loading...</p>

```

```
</section>
<section id="app" hidden>
  <p><video id="monitor" autoplay></video> <canvas
id="photo"></canvas></p>
  <p><input value="Click for pic" type=button value="&#x1F4F7;"
onclick="snapshot()">
</section>

<script>

if (navigator.getUserMedia) {

  navigator.getUserMedia({video:{mandatory:{minHeight:300,minWidth:600,cameraSelect:'front'}}, gotStream, noStream);

  var video = document.getElementById('monitor');
  var canvas = document.getElementById('photo');
  var localstream;

  function gotStream(stream) {

    video.src = URL.createObjectURL(stream);
    video.onerror = function () {
      stream.stop();
      streamError();
    };
    localstream = stream;
    document.getElementById('splash').hidden = true;
    document.getElementById('app').hidden = false;
  }

  function noStream() {
    document.getElementById('errorMessage').textContent =
'No camera available.';
  }

  function streamError() {
    document.getElementById('errorMessage').textContent =
'Camera error.';
  }

  function snapshot() {
    image_buffer = new ArrayBuffer;
    var image = document.createElement('img');
    //image.height = from camera settings
    //image.width = from camera settings
    localstream.captureImage(image_capture_function);
    function image_capture_function(img) {
      image_buffer = img;
    }
    var builder = new Blob([image_buffer])
    image.src = window.URL.createObjectURL(builder);
    canvas.getContext('2d').drawImage(image, 0, 0, image.width
, image.height);
  }
}
```

```
    } else {  
        document.getElementById('errorMessage').textContent = 'No  
native camera support available.';  
    }  
</script>
```