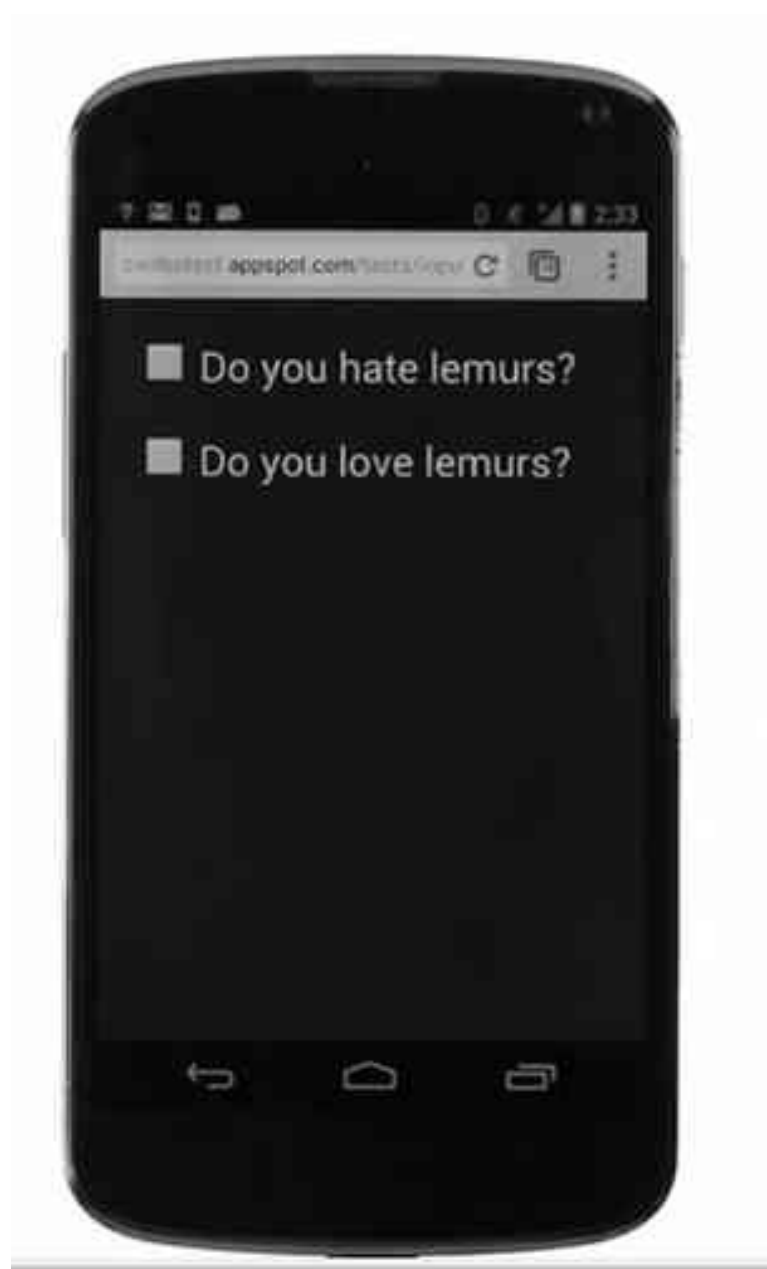




- ▶ The only difference between these two controls is that one of them has a label element around it.
- ▶ But you'll notice, it's much easier to touch the one with the label. The one without a label, you have to actually touch the check box itself, in order to activate the control.



```
<!DOCTYPE html>
```

```
▼ <html>
```

```
▶ <head>...</head>
```

```
▼ <body>
```

```
▼ <p>
```

```
<input type="checkbox">
```

```
" Do you hate lemurs?"
```

```
</p>
```

```
▼ <p>
```

```
▼ <label>
```

```
<input type="checkbox">
```

```
" Do you love lemurs?"
```

```
</label>
```

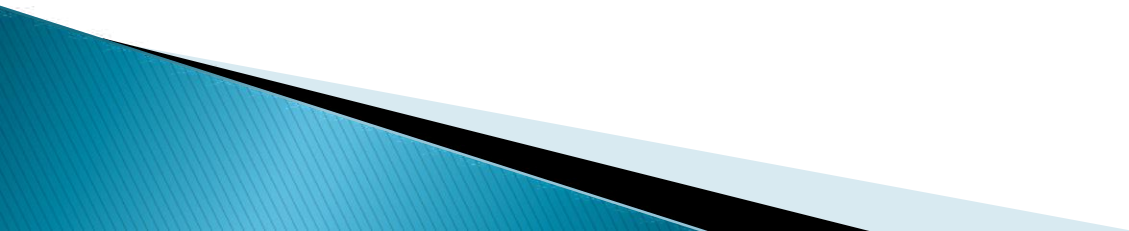
```
</p>
```

```
</body>
```

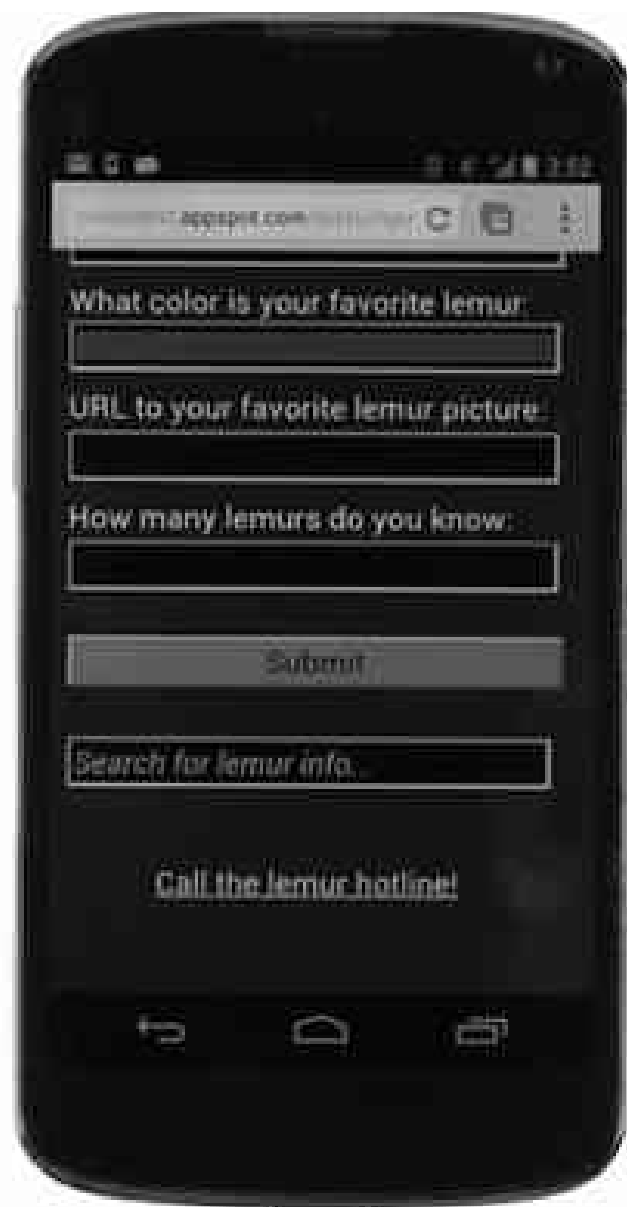
```
</html>
```

telecolon URL

On mobile, you can use a telecolon URL to create a URL that when activated will actually dial a telephone number. This is a really handy feature, particularly for business websites. Customers can instantly dial you.



- ▶ Of course, on desktop, these URLs will typically fail, so you probably want to deactivate them.

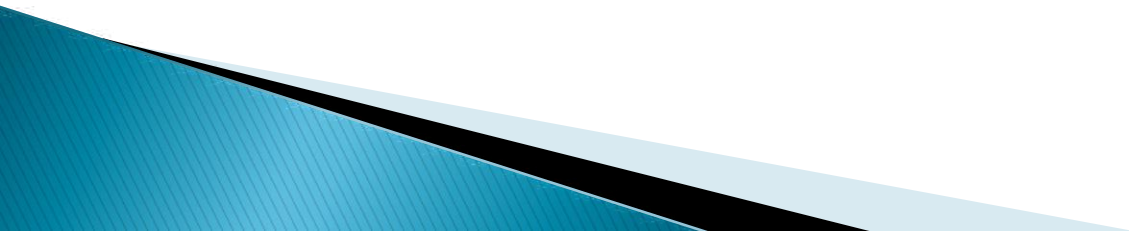


Elements	Resources	Network	Source
<pre><p>Welcome Lemur aficionados!</p> <form id="lemurs"> ▶ <div class="container">...</div> ▶ <div class="container">...</div> ▶ <div class="container">...</div> ▶ <div class="container">...</div> ▶ <div class="container">...</div> ▶ <div class="container">...</div> ▶ <div class="container">...</div> ▶ <div class="container">...</div> <p> </p> ▼ <div class="container"> ▶ <div>...</div> Call th </div> </form> body> ml></pre>			
 > 🔍 html body form#lem			



Sensors on mobile

One of the most amazing things about the mobile platform is the incredible array of sensors these devices typically have.



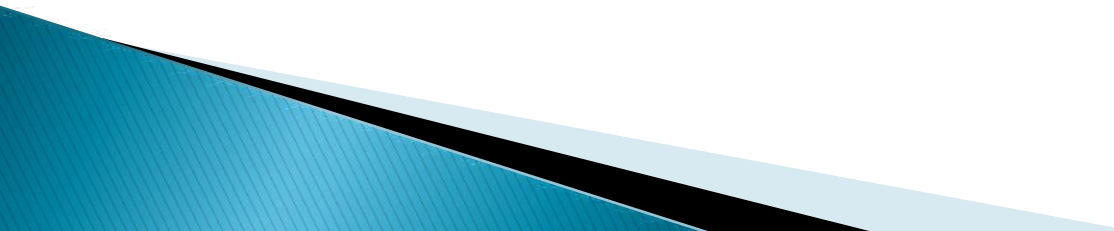
This phone has a camera that can take stills and video, has audio input and output, it has a GPS, a compass, a tilt sensing accelerometer.



A Mobile device has far more sensors than a desktop or laptop typically has.

And all this, in a device that one can typically hold in the palm of his/her hand.

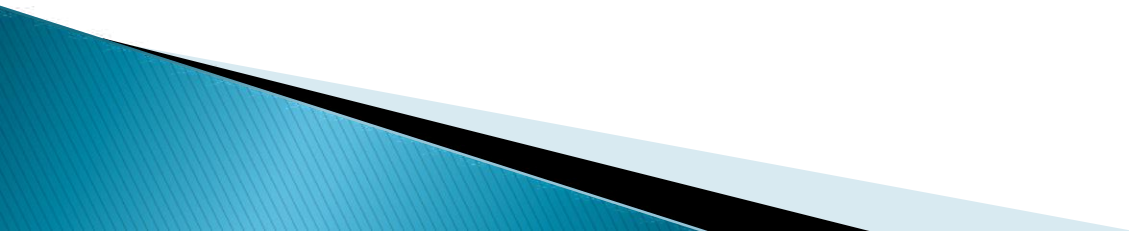
This enables the mobile platform to play host to incredibly engaging and interactive user experiences.



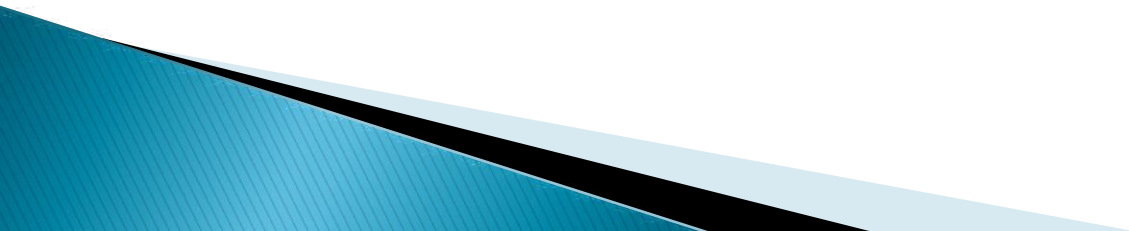
Camera Access

The camera is not just for taking pictures and making videos.

The camera can be a great way to get quick input into the mobile device.



For example you can snap a new profile picture using your mobile camera and upload it



- ▶ You capture a QR code to transfer a hyperlink



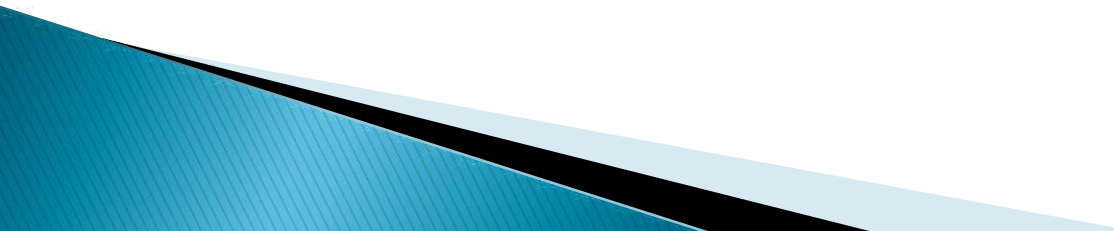
- ▶ You can capture a map or a business card.



Capture Extension

There is a simple way to get access to the camera and the audio input, the capture extension on the accept attribute on the file input element in HTML.

This will give you a button that opens up the system camera or Audio recorder app on a mobile device.



Accept Attribute

The accept attribute takes a comma-separated list of unique content types of files that are acceptable values for the file input type only. The comma separated values can include file extensions, MIME types without extensions,

```
<input type="file" accept="image/*">
```

(images only)

For Mobile Devices

```
<input type="file"  
accept="image/*;capture=camera">
```

```
<input type="file"  
accept="video/*;capture=camcorder">
```

```
<input type="file"  
accept="audio/*;capture=microphone">
```



Drawbacks of Capture Extension



- ▶ The first problem is that this feature only works in mobile, not in desktop. On desktop, you only get the normal file open dialogue. Not access to the webcam.

- ▶ Secondly, the interface is a file open button, which is hard to style and make look the way you might want it to.

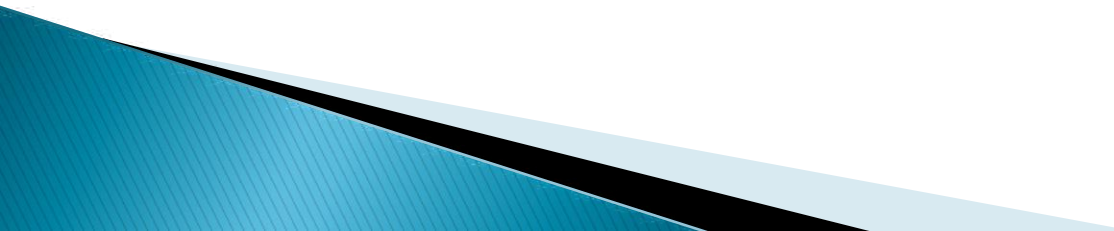
- ▶ Third issue is , taking you to a different application to take a picture or record some audio, makes it kind of hard to keep a consistent flow in your application.

Live Input

We can get Live Input, Audio and Video, directly under our App by using an API called getUserMedia.

Link:–

<https://developer.mozilla.org/en/docs/Web/API/Navigator/getUserMedia>



Example

```
var constraints = {video: true};

// prefix monkeypatching
navigator.getUserMedia = navigator.getUserMedia ||
    navigator.webkitGetUserMedia;

function successCallback(stream) {
    var video = document.querySelector("video");
    video.src = window.URL.createObjectURL(stream);
}

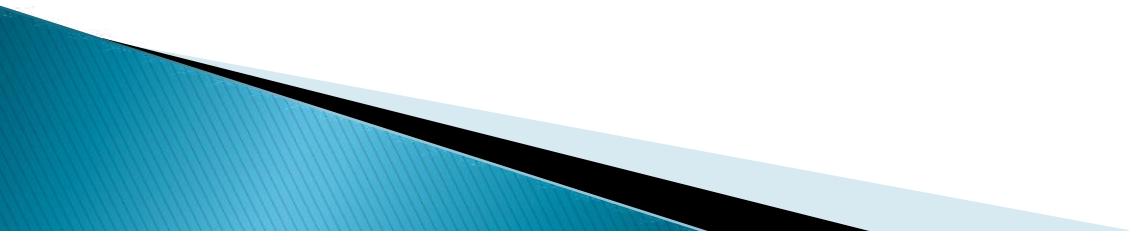
function errorCallback(error) {
    console.log("getUserMedia error: ", error);
}

navigator.getUserMedia(constraints, successCallback,
    errorCallback);
```

This is a simple example where we just call `getUserMedia`, passing some constraints that say we want video, and then we assign the resulting stream to a video element in the page. Of course, this just gives us a rectangle with live Video Input.

```
function snapshot() {  
    var video = document.querySelector('video');  
    var canvas = document.querySelector('canvas');  
  
    // ensure the canvas matches the video dimensions  
    canvas.width = video.videoWidth;  
    canvas.height = video.videoHeight;  
  
    // grab the video and draw it to the canvas  
    var ctx = canvas.getContext('2d');  
    ctx.drawImage(video, 0, 0);  
}
```

We probably want to do something with it. If we want to grab Snapshots like if we want to implement a Camera Application, we can do this with a canvas context and DrawImage. Whenever we want to take a Snapshot, we simply grab the image from the video and we draw it to the camera with DrawImage.



```
function snapshot() {  
    // ensure the canvas and img match the video  
    canvas.width = video.videoWidth;  
    canvas.height = video.videoHeight;  
    img.height = video.videoHeight;  
    img.width = video.videoWidth;  
  
    // grab the video image and draw to the canvas  
    var ctx = canvas.getContext('2d');  
    ctx.drawImage(video, 0, 0);  
  
    // encode the canvas contents to a data url.  
    img.src = canvas.toDataURL('image/png');  
}
```

Of course a canvas is different than an image. You can't directly copy a canvas and paste it somewhere else, or download it to your hard drive or post it to your social networks. Or any of the other typical User flows centered around images, but it turns out, canvas has a handy function to encode itself as an image. The `canvas.toDataURL` method.

Here, instead of displaying the canvas itself, we're copying the canvas contents as a data url to the source of an image tag. And if you want to save these images to the Mobile's local storage.

Navigator.getUserMedia()

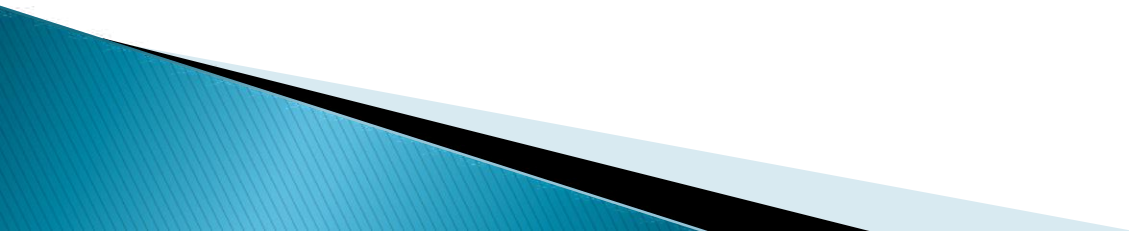
Syntax

```
navigator.getUserMedia(constraints,  
successCallback, errorCallback);
```

Parameters

Constraints

A `MediaStreamConstraints` object specifying the types of media to request, along with any requirements for each type.



- ▶ To *require* a capability, use the keywords min, max, or exact (a.k.a. min == max). The following demands a minimum resolution of 1280x720

```
1 {  
2   audio: true,  
3   video: {  
4     width: { min: 1280 },  
5     height: { min: 720 }  
6   }  
7 }
```

successCallback

When the call succeeds, the function specified in the successCallback is invoked with the MediaStream object that contains the media stream. You may assign that object to the appropriate element and work with it, as shown in the following example:

```
function(stream) {  
    var video = document.querySelector('video');  
    video.src = window.URL.createObjectURL(stream);  
    video.onloadedmetadata = function(e) {  
        // Do something with the video here.  
    };  
}
```

errorCallback

When the call fails, the function specified in the errorCallback is invoked with a `MediaStreamError` object as its sole argument

Error	Description
PermissionDeniedError	Permission to use a media device was denied by the user or the system.
NotFoundError	No media tracks of the type specified were found that satisfy the constraints specified.