Pointer Lock API

Quick Guides for Masterminds

J.D Gauchat
www.jdgauchat.com

Cover Illustration by **Patrice Garden**
www.smartcreativz.com

The content of this guide is a collection of excerpts from the book HTML5 for Masterminds. For more information, visit www.formasterminds.com.

The source code for this eBook is available at **www.formasterminds.com**

Copyright Registration Number: 1140725

1st Edition 2018

What is Inside

This guide will teach you how to use the Pointer Lock API to control the mouse. After reading this guide, you will know how to hide the mouse's pointer, and how to determine its position when it is hidden.

About this Guide

This guide is a collection of excerpts from the book HTML5 for Masterminds. The information included in this guide will help you understand a particular aspect of web development, but it will not teach you everything you need to know to develop a website or a web application. If you need a complete course on web development, read our book HTML5 for Masterminds. For more information, visit our website at www.formasterminds.com.

What Do You Need

This guide assumes that you have a basic knowledge of HTML, CSS and JavaScript, and you know how to create files and upload them to a server. If you don't know how to program in HTML, CSS or JavaScript, you can download our guides Introduction to HTML, Introduction to CSS, and Introduction to JavaScript. For a complete course on web development, read our book HTML5 for Masterminds.

IMPORTANT: We recommend you to execute the examples in this guide with the latest versions of Google Chrome and Mozilla Firefox (www.google.com/chrome and www.mozilla.com). You can also check the state of the current implementations at www.caniuse.com. To find examples, resources, links and videos, visit our website at www.formasterminds.com.

The Basics: To create the files for the examples of this guide, you can use a text editor (Notepad or Text Edit), or a professional editor like Atom (www.atom.io). If you do not have a server to test the files, you can install a server in your computer with a package like MAMP (www.mamp.info). For more information, read our free guide Web Development.

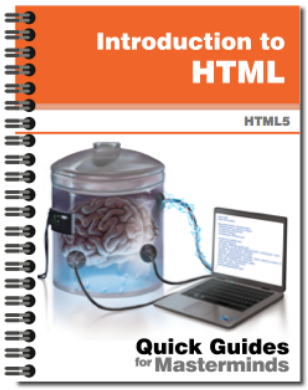
Recommendations

Introduction to JavaScript

Quick Guides for Masterminds

This guide will teach you how to program with JavaScript. After reading this guide, you will know how to create a program in JavaScript, how to define functions and objects, and how to read and modify an HTML document dynamically.

More Information

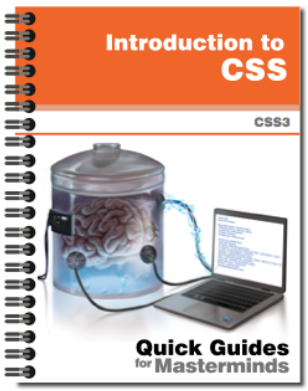


Introduction to HTML

Quick Guides for Masterminds

This guide will teach you how to create your website's documents with HTML. After reading this guide, you will know how to work with HTML elements, how to define a document's structure, and how to organize its content.

More Information



Introduction to CSS

Quick Guides for Masterminds

This guide will teach you how to program CSS Style Sheets to style your documents. After reading this guide, you will know how to style HTML elements, how to modify the styles dynamically, and how to use CSS to design your website or web application.

More Information

More Guides Available at www.formasterminds.com

Table of Contents

Pointer Lock API

Custom Pointer

Visual applications, like those created with the **<canvas>** element or WebGL, sometimes demand the use of other techniques to express the movement of the mouse. There are countless reasons for this. We may need to change the graphic that represents the pointer to indicate a different function for the mouse or hide it to avoid distracting the user from an image or a video. Considering these requirements, browsers include the Pointer Lock API.

Mouse Capture

The Pointer Lock API is a small group of properties, methods, and events that let the application assume control over the mouse. Two methods are provided to lock and unlock the mouse.

- **requestPointerLock()**—This method locks the mouse and binds it to an element. It is available on **Element** objects.
- **exitPointerLock()**—This method unlocks the mouse and makes the mouse pointer visible again. It is available in the **Document** object.

When the **requestPointerLock()** method is called, the graphic representing the pointer (usually a small arrow) is hidden, and the code becomes responsible for providing the visual clues the user needs to interact with the application. The following example illustrates how to take control of the mouse.

Listing 1: *Taking control of the mouse*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pointer Lock API</title>
  <script>
    function initiate() {
      var element = document.getElementById("control");
      element.addEventListener("click", lockmouse);
    }
    function lockmouse(event) {
      var element = event.target;
      element.requestPointerLock();
```

```
    }
    window.addEventListener("load", initiate);
  </script>
</head>
<body>
  <section>
    <p id="control">Click here to lock the mouse</p>
  </section>
</body>
</html>
```

Unless the element requesting control of the mouse is in full screen mode, the **requestPointerLock()** method will return an error when it is called without user intervention. A user gesture, such as the click of the mouse, must precede the action. Considering this, in the code in Listing 1 we add a listener for the **load** event to the window to execute a function called **initiate()** as soon as the document is loaded. This function adds a listener for the **click** event to the **<p>** element. When the user clicks on this element, the **lockmouse()** function is called, and the **requestPointerLock()** method is used to lock the mouse. At that moment, the pointer disappears from the screen, and it is not shown again unless the user switches to another window or cancels the mode pressing the escape key on the keyboard.

Do It Yourself: Create a new HTML file with the document of Listing 1. Open the document in your browser and click on the area occupied by the **<p>** element. You should see the mouse's pointer disappear. Press the Escape key on your keyboard to unlock the mouse.

When the mouse is locked for an element, the rest of the elements do not fire any mouse event until the mouse is unlocked by the application or the mode is canceled by the user. To report what is going on, the API includes the following events.

pointerlockchange—This event is fired on the **Document** object when an element locks or unlocks the mouse.

pointerlockerror—This event is fired on the **Document** object when the attempt to lock the mouse fails.

The code in the following example listens to the **pointerlockchange** event and prints a message on the console every time the condition of the mouse changes.

Listing 2: *Reporting a change in the condition of the mouse*

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <meta charset="utf-8">
  <title>Pointer Lock API</title>
  <script>
    function initiate() {
      var element = document.getElementById("control");
      element.addEventListener("click", lockmouse);
      document.addEventListener("pointerlockchange",
checkmouse);
    }
    function lockmouse(event) {
      var element = event.target;
      element.requestPointerLock();
    }
    function checkmouse() {
      console.log("The condition of the mouse changed");
    }
    window.addEventListener("load", initiate);
  </script>
</head>
<body>
  <section>
    <p id="control">Click here to lock the mouse</p>
  </section>
</body>
</html>
```

Do It Yourself: Update the document in your HTML file with the code in Listing 2. Open the document in your browser and activate the console to see the messages printed by the code. Every time you click on the element or press the Escape key, a message should be shown on the console.

The mouse is locked for a specific element. The API includes the following property to report which element locked the mouse.

pointerLockElement—This property returns a reference to the **Element** object representing the element that locked the mouse or the value **null** if the mouse hasn't been locked.

The **pointerLockElement** property may be used along with the **pointerlockchange** event to determine if the mouse was locked or unlocked.

Listing 3: *Checking the condition of the mouse*

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="utf-8">
  <title>Pointer Lock API</title>
  <script>
    function initiate() {
      var element = document.getElementById("control");
      element.addEventListener("click", lockmouse);
      document.addEventListener("pointerlockchange",
checkmouse);
    }
    function lockmouse(event) {
      var element = event.target;
      element.requestPointerLock();
    }
    function checkmouse() {
      var element = document.getElementById("control");
      if (document.pointerLockElement) {
        console.log("Mouse Locked");
      } else {
        console.log("Mouse Unlocked");
      }
    }
    window.addEventListener("load", initiate);
  </script>
</head>
<body>
  <section>
    <p id="control">Click here to lock the mouse</p>
  </section>
</body>
</html>
```

Every time the mouse is locked or unlocked, either by our code or by the user, the **checkmouse()** function in Listing 3 checks the value of the **pointerLockElement** property. If the property returns a reference to an element, it means that the mouse is currently locked, but if the property returns the value **null**, it means that no element is locking the mouse at the moment.

Do It Yourself: Update the document in your HTML file with the code in Listing 3. Open the document in your browser and activate the console to see the messages. Click on the **<p>** element to deactivate the mouse. You should see the message "Mouse Locked" on the console. Press the Escape key. You should see the message "Mouse Unlocked" on the console.

As we mentioned, if no element has control over the mouse, the **pointerLockElement** property returns the value **null**, so we can use it to decide whether to lock or unlock the mouse depending on the current condition.

Listing 4: *Locking and unlocking the mouse*

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pointer Lock API</title>
  <script>
    var canvas;
    function initiate() {
      var element = document.getElementById("canvas");
      canvas = element.getContext("2d");
      element.addEventListener("click", lockmouse);
      element.addEventListener("mousemove", draw);
    }
    function draw(event) {
      canvas.clearRect(0, 0, 500, 400);
      var posx = event.clientX;
      var posy = event.clientY;

      canvas.beginPath();
      canvas.moveTo(posx, posy - 20);
      canvas.lineTo(posx, posy + 20);
      canvas.moveTo(posx - 20, posy);
      canvas.lineTo(posx + 20, posy);
      canvas.moveTo(posx + 20, posy);
      canvas.arc(posx, posy, 20, 0, Math.PI * 2);
      canvas.stroke();
    }
    function lockmouse(event) {
      var element = event.target;
      if (!document.pointerLockElement) {
        element.requestPointerLock();
      } else {
        document.exitPointerLock();
      }
    }
    window.addEventListener("load", initiate);
  </script>
</head>
<body>
  <section>
```

```
    <canvas id="canvas" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

In this example, we create a small application to show a more realistic approach to the use of this API. We use the **pointerLockElement** property to determine whether the canvas is in control of the mouse or not. Every time the user clicks on the **<canvas>** element, we check this condition and lock or unlock the mouse with **requestPointerLock()** or **exitPointerLock()** accordingly.

When the mouse is locked, the browser assigns control of the mouse to the element that made the request. All the mouse events, such as **mousemove**, **click** or **mousewheel**, are only fired for this element, but events that are related to the mouse's pointer, such as **mouseover** or **mouseout**, are no longer fired. As a result, to interact with the mouse and detect its movements, we have to listen to the **mousemove** event. The code in Listing 2 adds a listener for the **mousemove** event to draw a gun sight on the canvas in the current position of the mouse, determined by the **clientX** and **clientY** properties.

Do It Yourself: Update the document in your HTML file with the code in Listing 4. Open the document in your browser and move the mouse to the area occupied by the **<canvas>** element. You should see a gun sight following the pointer. Click to lock the mouse and fix the gun sight in place. Click again to unlock it.

In Listing 4, the gun sight moves with the mouse pointer, but as soon as the mouse is locked, the gun sight is frozen on the screen. This is because when the mouse is locked, only the values of the **movementX** and **movementY** properties are updated (the rest of the event properties keep the last values stored prior to the activation of the mode). These properties do not return the exact position of the mouse; they return the difference between the previous and the current position. To work with the mouse when it is locked, we must calculate its position from the values returned by these properties.

Listing 5: *Calculating the position of the mouse with* movementX *and* movementY

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Pointer Lock API</title>
  <script>
    var canvas, posx, posy;
    function initiate() {
      var element = document.getElementById("canvas");
```

```
    canvas = element.getContext("2d");
    element.addEventListener("click", lockmouse);
    startmessage();
  }
  function draw(event) {
    canvas.clearRect(0, 0, 500, 400);

    var test1, test2;
    test1 = posx + event.movementX;
    test2 = posy + event.movementY;
    if(test1 > 0 && test1 < 500){
      posx = test1;
    }
    if(test2 > 0 && test2 < 400){
      posy = test2;
    }
    canvas.beginPath();
    canvas.moveTo(posx, posy - 20);
    canvas.lineTo(posx, posy + 20);
    canvas.moveTo(posx - 20, posy);
    canvas.lineTo(posx + 20, posy);
    canvas.moveTo(posx + 20, posy);
    canvas.arc(posx, posy, 20, 0, Math.PI * 2);
    canvas.stroke();
  }
  function lockmouse(event) {
    var element = event.target;
    if (!document.pointerLockElement) {
      element.requestPointerLock();
      posx = event.clientX;
      posy = event.clientY;
      element.addEventListener("mousemove", draw);
    } else {
      document.exitPointerLock();
      element.removeEventListener("mousemove", draw);
      startmessage();
    }
  }
  function startmessage() {
    canvas.clearRect(0, 0, 500, 400);
    canvas.font = "bold 36px verdana, sans-serif";
    canvas.fillText("Click to Play", 120, 180);
  }
  window.addEventListener("load", initiate);
</script>
```

```
</head>
<body>
  <section>
    <canvas id="canvas" width="500" height="400"></canvas>
  </section>
</body>
</html>
```

We made a few changes in this code to increase the level of control and show how to take advantage of this API in certain phases of the application. The **initiate()** function sets the canvas, adds the listener for the **click** event to lock the mouse, and calls the **startmessage()** function to show a welcome message on the screen, but it does not add a listener for the **mousemove** event. This listener is added in the **lockmouse()** function after the mouse is locked, and it is also removed in the same function after the mouse is unlocked. By following this procedure, the **<canvas>** element gets control of the mouse only after the first stage of the application is executed (where we ask the user to click on the screen). If the user returns to this first stage, the mouse is unlocked again, and the listener for the **mousemove** event is removed by the **removeEventListener()** method.

The values returned by the **movementX** and **movementY** properties reflect the change of the position of the mouse. Before the mouse is locked, we capture its coordinates with the **clientX** and **clientY** properties to set the initial position of the gun sight. Usually, this is not necessary, but in an application like this, where the mouse is constantly locked and unlocked, it helps produce a better transition from one state to another. This initial position is stored in the variables **posx** and **posy** and is only modified by the amount of displacement when it does not exceed the limits established by the size of the **<canvas>** element (see the **if** instructions in the **draw()** function). Unless we are just checking the direction of the mouse, this control is necessary to avoid storing values that our application cannot process.

Do It Yourself: Update the document in your HTML file with the code in Listing 5. Open the document in your browser. Click on the canvas to lock the mouse. You should see the gun sight moving with the mouse, but not the mouse's pointer. Click again to go back to the initial screen.

Quick Reference

Methods

requestPointerLock()—This method locks the mouse and binds it to an element. It is available on **Element** objects.

exitPointerLock()—This method unlocks the mouse and makes the mouse pointer visible again. It is available in the **Document** object.

Events

pointerlockchange—This event is fired on the **Document** object when an element locks or unlocks the mouse.

pointerlockerror—This event is fired on the **Document** object when the attempt to lock the mouse fails.

Property

pointerLockElement—This property returns a reference to the **Element** object representing the element that locked the mouse or the value **null** if the mouse hasn't been locked.

For Masterminds

Book Series

for more Books and Quick Guides visit

www.formasterminds.com