

# How-To Create mobile web applications with HTML5 and its new visual UI features

**Add Canvas, CSS3, and more semantic elements to mobile web apps**

---

HTML5 is a popular technology since its early days and for a very good reason. It brought desktop application-like capabilities to the web browser — not just on the desktop but also on mobile devices.

In this five-part series, we will take a look at some of the popular features that are part of HTML5. In each part, we will also write some code to showcase these features and see how they work on both desktop and mobile browsers.

HTML5 comes with plenty of new features for mobile Web applications, including visual ones that usually make the most impact. Canvas is the most eye-catching of the new UI capabilities, providing full 2-D graphics in the browser. In this article you learn to use Canvas as well as some of the other new visual elements in HTML 5 that are more subtle but make a big difference for mobile users.

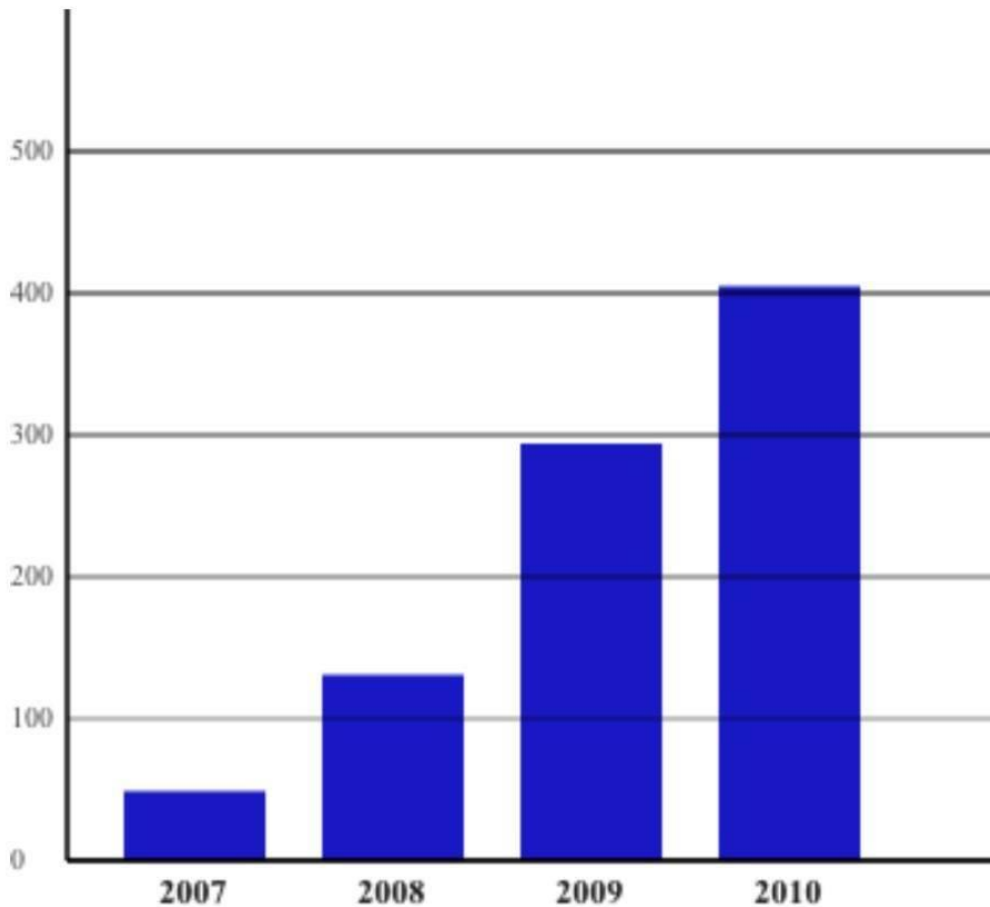
## Prerequisites

In this tutorial, we will develop a web application to showcase the visual features introduced in HTML5. The code we will see uses core web technologies like HTML, Cascading Style Sheets (CSS), and JavaScript. To test the app on desktop and mobile, we recommend using the latest web browser versions.

## Getting graphical with Canvas

Canvas offers a native drawing API for web browsers. Using the Canvas API, you can draw different shapes like lines, circles, and polygons, and fill them with colors and gradients. You can create text and perform various geometric transformations. It can be used for things like data visualization, animation, photo, and video processing. The best thing is that Canvas is well supported by the recent versions of nearly all popular browsers.

Let's take a look at how to create a graph using Canvas. Figure 1 shows a screen capture of the app: a bar graph of yearly results.



The graph is generated using the Canvas API. Listing 1 shows the HTML code used to create this graph.

Listing 1. HTML for bar graph

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>HTML 5 Reports</title>
    <script type="text/javascript">
      const initialize = () => {
        const data = [
          { year: "2007", sales: 49 },
          { year: "2008", sales: 131 },
          { year: "2009", sales: 294 },
          { year: "2010", sales: 405 }
        ];
        const report = { x: "year", y: "sales", data };
      };
    </script>
  </head>
  <body>
    <div>
      <canvas>
        <!-- Canvas API code for bar graph -->
      </canvas>
    </div>
  </body>
</html>
```

```

        generateGraph(report, 350, 300);
    };

    </script>
</head>
<body onload="initialize()">
    <canvas id="graph"></canvas>
</body>
</html>
Show more

```

The body of the document contains a Canvas element. When the page is loaded, the `initialize()` function is called. This function passes the static report data to the `generateGraph` function, which is the function that generates the bar graph, as shown in Listing 2.

Listing 2. The `generateGraph` function

```

const generateGraph = (report, maxWidth, maxHeight) => {
    const { data } = report;
    const canvas = document.getElementById("graph");
    const axisBuffer = 20;
    canvas.height = maxHeight + 100;
    canvas.width = maxWidth;
    const context = canvas.getContext("2d");

    const width = 50;
    const buffer = 20;
    const i = 0;
    let x = buffer + axisBuffer;
    context.font = "bold 12px sans-serif";
    context.textAlign = "start";

    data.forEach(item => {
        context.fillStyle = "rgba(0, 0, 200, 0.9)";
        context.fillRect(
            x,
            maxHeight - item[report.y] / 2,
            width,
            item[report.y] / 2
        );
        context.fillStyle = "rgba(0, 0, 0, 0.9)";
        context.fillText(item[report.x], x + width / 4, maxHeight + 15);
        x += width + buffer;
    });

    // draw the horizontal axis
    context.moveTo(axisBuffer, maxHeight);
    context.lineTo(axisBuffer + maxWidth, maxHeight);

```

```

    context.strokeStyle = "black";
    context.stroke();

    // draw the vertical axis
    context.moveTo(axisBuffer, 0);
    context.lineTo(axisBuffer, maxHeight);
    context.stroke();

    // draw gridlines
    const lineSpacing = 50;
    const numLines = maxHeight / lineSpacing;
    let y = lineSpacing;
    context.font = "10px sans-serif";
    context.textBaseline = "middle";

    for (let i = 0; i < numLines; i++) {
        context.strokeStyle = "rgba(0,0,0,0.25)";
        context.moveTo(axisBuffer, y);
        context.lineTo(axisBuffer + maxWidth, y);
        context.stroke();
        context.fillStyle = "rgba(0,0,0, 0.75)";
        context.fillText("" + 2 * (maxHeight - y), 0, y);
        y += lineSpacing;
    }
};

```

Show more

In the `generateGraph` function, you first set up the objects needed to create the report, like the width and height of the canvas, and padding variables. You also create the Canvas context object, which will be used for the actual drawing. Then you draw the bar graphs seen in Figure 1 by iterating over the report data. You set the `fillStyle` property using the `rgba` notation to set the color and the alpha value (transparency of color is discussed in a later section). After setting the `fillStyle` property, create the bar graph for the data point using the `fillRect` API. Here, we specify the `(x, y)` starting point of the rectangle and its height and width. Next, re-define the `fillStyle` to print some text as part of the report. Use the `fillText` API to draw text on the canvas. This API takes the `(x, y)` starting point and the text. You then do this for each of the data points, creating a bar graph with a label below it.

Next, draw the horizontal and vertical axes. For each of these, use the `moveTo` API to set the point from which you will begin to draw a line. Then use the `lineTo` API to draw a line from the starting point to the end point passed in to the `lineTo` call and call the `stroke` API to draw the line. Next, you draw the gridlines along with their labels. Draw evenly spaced lines using the same combination of `moveTo`, `lineTo`, and `stroke`.

In this example, we have discussed some of the commonly used Canvas APIs. There are many other functions that are part of the Canvas API, allowing you to do some amazing things.

## The wonderful world of CSS

When talking about HTML5, you may instantly think about the HTML tags and many new tags introduced as part of the specification, but the HTML tag is just part of the story. JavaScript and CSS are equally important parts to web development. Many new user interface elements were introduced as part of CSS 3.0. In Figure 2, you can see a web page using some of the CSS 3.0 features.

## The World of CSS3

nth-child

gradients

alpha

text effects

reflections

transformations

What's your name?

Let's look at the code (in Listing 3) that generated this web page.

Listing 3. Code to demonstrate new CSS capabilities

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">

function init() {
  var i = 0;
  var row = {};
  var cell = {};
  var topics = [
    "nth-child",
    "gradients",
    "alpha",
    "text effects",
    "reflections",
    "transformations"
  ];
  for (i = 0; i < topics.length; i++) {
    row = document.createElement("tr");
    cell = document.createElement("td");
    cell.appendChild(document.createTextNode(topics[i]));
    row.appendChild(cell);
    document.getElementById("dtable").appendChild(row);
  }
}

    </script>
    <style type="text/css">
```

```

        header > h1{
        color: yellow;
        background: linear-gradient(to right, red , yellow);
        }
        table tr:nth-child(odd) {
        color: navy;
        }
        table tr:nth-child(even) {
        color: green;
        }
        input[type="text"]{
        background: rgba(150, 30, 30, 0.5);
        }
    </style>
</head>
<body onload="init()">
    <header>
        <h1>The World of CSS3</h1>
    </header>
    <table id="dtable"></table>
    <div id="formSection">
        <label for="name">What's your name?</label>
        <input type="text" id="name"></input>
        <button id="rtBtn" onclick="rotate()">Rotate</button>
    </div>
</body>
</html>
Show more

```

Inside the body of the page, the first thing to notice is the `header` tag, a new HTML element introduced as part of HTML5. This element can be used to define a header of a page or a section.

If you look at the style element in the code above, we are using CSS selector `header > h1` to style the heading text. This will make the text yellow and the background with a linear gradient from red to yellow. While these styles work well with the latest version of most browsers, some browsers, especially the older versions, may require a browser specific prefix for some of these styles to work properly. After the header, we have a table with ID `dtable`. Use the `css nth-child` selector. Here, the CSS displays odd table rows in a navy color and even rows in a green color. In the past, this would have required custom JavaScript code, but it can now be done easily with CSS selectors.

The last visual element on the page is the red-colored text field with the label `What's your name?` and a **Rotate** button. Here you have used a selector that targets a specific attribute of the element. The selector `input[type="text"]` is only applicable to input elements with type text.

Let's see the code in Listing 4 to see what the **Rotate** button does.

Listing 4. JavaScript rotation function using CSS

```
function rotate() {
```

```

    document.getElementById("formSection").style["transform"] = "rotateZ(-5deg)";
    document.getElementById("formSection").style["transition"] = "transform 2s ease-in-out";
    document.getElementById("rtBtn").innerHTML = "Undo";
    document.getElementById("rtBtn").onclick = function() {
        document.getElementById("formSection").style["transform"] = "";
        document.getElementById("rtBtn").innerHTML = "Rotate";
        document.getElementById("rtBtn").setAttribute("onclick",
"rotate()");
    };
}

```

Show more

The button calls function `rotate` when clicked. This function uses JavaScript to change the CSS that is applied to the `formSection` div. When clicked, the button will rotate the div by five degrees counter-clockwise. This is achieved using the `transform` style with the value `rotateZ(-5deg)`. We also set the `transform` style to `transform 2s ease-in-out` to make the rotate take two seconds, starting slowly, accelerating, then slowing down again. In Figure 3, you can see the rotated fields with a button to undo the rotation.

## The World of CSS3

[nth-child](#)  
[gradients](#)  
[alpha](#)  
[text effects](#)  
[reflections](#)  
[transformations](#)

What's your name?

Now let's look at some of the interesting layout, image and text effects introduced in CSS in Figure 4. The code for this is in Listing 5.

## The Gettysburg Address

Abraham Lincoln, Gettysburg, PA. November 19, 1863

Four score and seven years ago ...

Now we are engaged in a great civil war, testing whether that nation, or any nation, so conceived and so	dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a	portion of that field, as a final resting place for those who here gave their lives that that nation	might live. It is altogether fitting and proper that we should do this.
--	--	--	---

Listing 5. Code to demonstrate new CSS capabilities

```
<!DOCTYPE html>
<html>

<head>
  <style type="text/css">
    h2 {
      -webkit-text-fill-color: blue;
      -webkit-text-stroke-color: yellow;
      -webkit-text-stroke-width: 1.5px;
      background: -webkit-gradient( radial, 430 50, 0, 430 50, 200,
from(red), to(#000));
      -webkit-box-reflect: below 5px -webkit-gradient(linear, left
top, left bottom, from(transparent), color-stop(0.5, transparent),
to(white));
    }

    h3 {
      color: rgba(0, 0, 255, 0.75);
      background: rgba(255, 255, 0, 0.5);
    }

    .xyz {
      text-shadow: #6374ab 4px -4px 2px;
      white-space: nowrap;
      width: 14em;
      height: 2em;
      overflow: hidden;
      text-overflow: ellipsis;
      border: 1px solid #bbb;
      border-radius: 9px;
      background-color: #fff;
    }

    .abc {
      border: 1px solid #000;
      border-radius: 9px;
      column-count: 4;
      column-rule: 1px solid #a00;
      column-gap: 0.75em;
    }
  </style>
</head>

<body onload="init()">
  <h2>The Gettysburg Address</h2>
  <h3>Abraham Lincoln, Gettysburg, PA. November 19, 1863</h3>
  <div class="xyz">
```



```
    Four score and seven years ago our fathers brought forth on this
    continent a new nation, conceived in liberty, and dedicated to the
    proposition that all men are created equal.
```

```
</div>
```

```
<div class="abc">
```

```
    Now we are engaged in a great civil war, testing whether that
    nation, or any nation, so conceived and so dedicated, can long endure. We
    are met on a great battle-field of that war. We have come to dedicate a
    portion of that field, as a final resting place for those who here gave
    their lives that that nation might live. It is altogether fitting and
    proper that we should do this.
```

```
</div>
```

```
</body>
```

```
</html>
```

Show more

Let's walk element-by-element through this code. First, we have a heading "The Gettysburg Address" with the following styles:

1. We used `-webkit-text-fill-color`, `-webkit-text-stroke-color`, and `-webkit-text-stroke-width` styles to create the blue-inside-the-yellow effect.
2. A red and black background is placed behind the text by setting the background style `-webkit-gradient`. In the previous example, we used linear gradient, but here we are using radial gradient.
3. Finally, we applied a reflection to the heading by setting the `-webkit-box-reflect` style. This is set to reflect the heading five pixels below it, with a gradient effect applied to the reflection, making the reflection seem to fade away.

For the next heading, we applied simple styling with text and background color. We used the `rgba` function to specify the red-green-blue values and an alpha transparency value. An alpha transparency value of 0.0 is transparent, and a value of 1.0 is completely opaque.

Next, we have a paragraph with a border around the text. The rounded corners for the border is achieved using `border-radius` style. We also applied a shadow to the text by using the `text-shadow` style. Finally, as you can see, the text is quite long for the height and width set for the parent div; by setting the `text-overflow` style, we get a nice ellipsis (...), instead of truncating the text.

Finally, you can see how the last paragraph is displayed as four columns, which is achieved by using the `column-count` style. Each column also has a column separator using the `column-rule` style.

Along with the features we have discussed, there are a lot of new features introduced in CSS 3.0 to help achieve amazing visual effects, which work across different browsers, without the need to write complicated custom code.

## New semantics

Many new elements were added as part of the HTML5 specification. While some of these are visual elements, others add extra semantic meaning without any change in look and feel. These new

semantics are important to make the web apps accessible for non-visual users, including anyone using assistive technologies like screen readers and even programs like search engine crawlers.

Figure 5 shows a web page using some of the new semantic elements.



## There are a lot of new markup elements in HTML5

Sunday, May 16

Did you notice that we just had two H1's? But it's cool!

If this page was really popular, I'd put an ad here and make some serious cash

This example has a header element, a nav element, an article, a section, and an aside element. These elements do not cause any special rendering. They just add semantic value, and you can use them to write CSS that gives them visual treatments to match that semantic value. The code for the page shown in Figure 4 is in Listing 6.

Listing 6. New semantic elements in HTML5

```
<!DOCTYPE html>
<html>

<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Get the latest markup</title>
</head>

<body>
  <header style="border: 1px dotted #000;border-radius: 3px;">
    <hgroup align="center">
      <h1>Real documents have headers</h1>
      <h2>Even if they don't say so</h2>
    </hgroup>
    <hgroup>
      <nav style="column-count:3; column-rule: 1px solid #a00;">
        <a href="new-css.html">CSS3</a>
        <br />
        <a href="report.html">Canvas</a>
        <br />
        <a href="elements.html">Markup</a>
      </nav>
    </hgroup>
  </body>
</html>
```

```

</header>
<article>
  <h1>There are a lot of new markup elements in HTML5</h1>
  <time datetime="2010-05-16" pubdate>Sunday, May 16</time>
  <section>
    Did you notice that we just had two H1's? But it's cool!
  </section>
  <aside style="white-space:nowrap;overflow:hidden;text-
overflow:ellipsis;">
    If this page was really popular, I'd put an ad here and make
    some serious cash
  </aside>
</article>
</body>

</html>
Show more

```

In the code above, we have used a number of new elements previously mentioned. We have applied CSS styles to create a box with rounded corners around the header and to create separators for the nav element. We also used the text overflow styling on the aside element. Using the HTML5 semantic elements with CSS 3.0, we have not only created a much more meaningful markup, but also styled them without much effort.

Now let's look at Figure 6 for visual elements introduced as part of HTML5.

Settings

Username

Name

Email

Phone

Date of birth

Preferred Contact Method

1. ☐ Email

2. ☐ Phone

Preferred external temperature

Favorite color

Where'd you hear about us?

The screen in Figure 6 uses many new form elements available in HTML5. Although these may look like existing elements, they provide better visual representation and user experience based on a user's device and browser.

Let's look at the above web page on the iPhone Safari browser in Figure 7.

8:33



html5-examples.surge.sh



Settings

Username

Name

Email

Phone

Date of birth

Preferred Contact Method

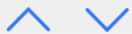
- ☐ Email
- ☐ Phone

Preferred external temperature

Favorite color

Where'd you hear about us?

Save



Clear Done

22	October	2015
23	November	2016
24	December	2017
25	January	2018
26	February	2019
27	March	2020
28	April	2021
29	May	2022
30	June	2023

As you can see, the date field that looked like a normal form field is rendered differently on an iPhone, giving users a better experience. The code used to generate the form is shown in Listing 7.

Listing 7. HTML5 form elements in code

```
<form id="settings">
  <fieldset id="inputs" style="border: 1px solid #000;border-radius:
6px;">
    <legend>Settings</legend>
    <label for="name">Username</label>
    <input id="name" name="name" type="text" required autofocus />
    <br />
    <label for="name">Name</label>
    <input id="name" name="name" type="text" placeholder="First and
last name" required />
    <br />
    <label for="email">Email</label>
    <input id="email" name="email" type="email"
placeholder="example@domain.com" required />
    <br />
    <label for="phone">Phone</label>
    <input id="phone" name="phone" type="tel" placeholder="Eg.
+447500000000" required />
    <br />
    <label for="dob">Date of birth</label>
    <input id="dob" name="dob" type="date" required />
    <fieldset style="border: 1px dotted #000; border-radius: 6px">
      <legend>Preferred Contact Method</legend>
      <ol>
        <li>
          <input id="emailMeth" name="contactMethod"
type="radio" />
          <label for="emailMeth">Email</label>
        </li>
        <li>
          <input id="phoneMeth" name="contactMethod"
type="radio" />
          <label for="phoneMeth">Phone</label>
        </li>
      </ol>
    </fieldset>
    <label for="climate">Preferred external temperature</label>
    <input id="climate" name="climate" type="range" min="50" max="100"
step="5" value="70" />
    <br />
    <label for="color">Favorite color</label>
    <input id="color" name="color" type="color" />
    <br />
    <label for="referrer">Where'd you hear about us?</label>
```

```

        <input type="url" name="refUrl" id="referrer" list="urls" />
        <datalist id="urls">
            <option label="TechCrunch" value="http://www.techcrunch.com/">
</option>
            <option label="ReadWrite Web"
value="http://www.readwriteweb.com/">
</option>
            <option label="Engadget" value="http://www.engadget.com/">
</option>
            <option label="Ajaxian" value="http://www.ajaxian.com/">
</option>
        </datalist>
        <br />
        <button type="button" onclick="checkInputs()">Save</button>
    </fieldset>
</form>
Show more

```

We have used many new HTML5 features in Listing 7. We used the required attribute to mark a field as required, which can also be used during form validation. We have also used the autofocus attribute that allows us to focus a field automatically when page loads.

For many of the form elements, we have used the placeholder attribute to provide a hint describing the element. We have also used new types for the input fields like email, phone, date, range, color, and URL. The input fields with these new types might look the same, but they provide different experiences to users based on their devices and browsers. For example, we looked at how the input field of date type looks on an iPhone in Figure 7.

## Summary

In this tutorial, we looked at many of the visual features of HTML5, including Canvas, CSS 3.0 features, and new HTML visual and semantic elements. Using these features, we can build web applications with amazing user interface and experiences across different browsers without the need to write complicated custom code. Using these visual elements along with the other features discussed in previous parts of this series, you can build web applications that are on par with native applications.

Courtesy: <https://developer.ibm.com/tutorials/x-html5mobile5/>

Modified: 2021.10.06.8.07.PM

Dököll Solutions, Inc