


williammalone

-
-
-
-
-



▶ article contents

- [Define Our Objective](#)
- [Prepare HTML5 Canvas: Markup](#)
- [Prepare HTML5 Canvas: Scripting](#)
- [Create a Simple Drawing "Canvas"](#)
- [Simple Drawing Canvas Demo](#)
- [Add Colors](#)
- [Demo Colors](#)
- [Add Sizes](#)
- [Demo Sizes](#)
- [Add Tools](#)
- [Demo Tools](#)
- [Add Outline](#)
- [Demo Outline](#)
- [Final Details](#)
- [Complete Demo](#)
- [Download Source Code](#)
- [Related Articles](#)
- [Share this Article](#)
- ○ 



 **followme** ◀

 **subscribe** ◀

relatedarticles ◀

- [HTML5 Game Character](#)



- [HTML5 Canvas Paint Bucket](#)



- [HTML5 Canvas Example](#)



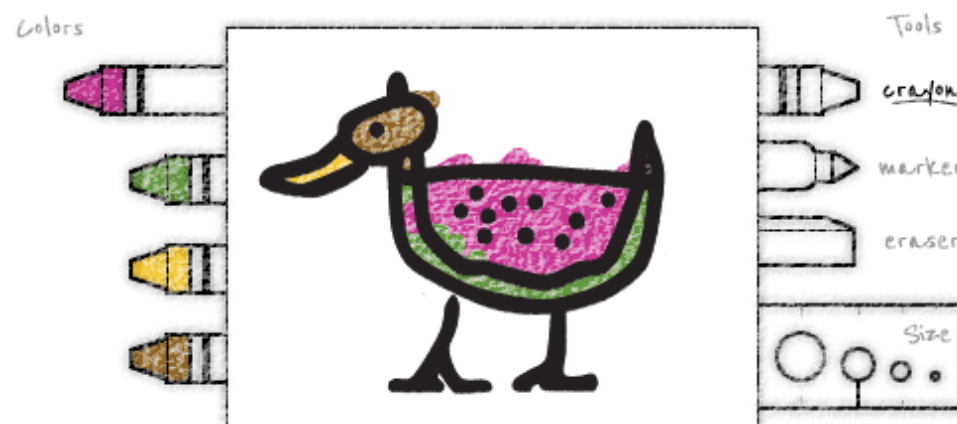
relatedbriefs ◀

- [How do you draw a rectangle on HTML5 canvas?](#)

Create a Drawing App with HTML5

Canvas and JavaScript

by William Malone



This tutorial will take you step by step through the development of a simple web drawing application using HTML5 canvas and its partner JavaScript. The aim of this article is to explore the process of creating a simple app along the way learn:

- How to draw dynamically on HTML5 canvas
- The future possibilities of HTML5 canvas
- The current browser compatibility of HTML5 canvas

Each step includes a working demo; if you want to skip ahead:

- [Simple Demo](#)
- [Colors Demo](#)
- [Sizes Demo](#)
- [Tools Demo](#)
- [Outline Demo](#)
- [Complete Demo](#)

Define Our Objective

Let's create a web app where the user can dynamically draw on an HTML5 canvas. What will our users use? A coloring book comes to mind; that means crayons. Our first tool is a crayon. Although the real world doesn't agree, I think we should be able to erase crayons. Our second tool will be an eraser (sorry reality). And because I have always have been a Sharpie® fan our final tool will be a marker.

Our tools could use colors (except maybe our eraser). Let's keep it simple, so we give our user 4 different colors to choose from.

Similarly let's also give our user 4 different sizes to draw with, because we can. To recap our app should have the following:

- 3 tools: crayon, marker, eraser
- 4 colors to choose from (except eraser)
- 4 sizes to choose from

Like a coloring book, let's give our user something to "color". I have chosen a favorite of mine: [Watermelon Duck](#) by Rachel Cruthirds.

Prepare HTML5 Canvas: Markup

We only need a line of markup; everything else will be in scripting.

```
<canvas id="canvasInAPerfectWorld" width="490" height="220"> </canvas>
```

Wait... HTML5 is still new and some browsers (psst... that means you Internet Explorer) don't support the canvas tag, so let's use this line of markup instead:

```
<div id="canvasDiv"> </div>
```

Prepare HTML5 Canvas: Scripting

To prepare our canvas, we would hope to use:

```
context = document.getElementById('canvasInAPerfectWorld').getContext("2d");
```

However IE doesn't know what the canvas tag means, and if we used that in our markup, IE would serve us an entrée of error. Instead we create a canvas element in JavaScript and append it to our div we called canvasDiv.

```
var canvasDiv = document.getElementById('canvasDiv');
canvas = document.createElement('canvas');
canvas.setAttribute('width', canvasWidth);
canvas.setAttribute('height', canvasHeight);
canvas.setAttribute('id', 'canvas');
canvasDiv.appendChild(canvas);
if(typeof G_vmlCanvasManager != 'undefined') {
    canvas = G_vmlCanvasManager.initElement(canvas);
}
context = canvas.getContext("2d");
```

For Internet Explorer compatibility we will also have to include an additional script: [ExplorerCanvas](#).

Create a Simple Drawing "Canvas"

Before we add any options, let's tackle the basics of dynamically drawing on an HTML5 canvas. It will consist of 4 mouse events and two functions: addClick to record mouse data and redraw which will draw that data.

Mouse Down Event: When the user clicks on canvas we record the position in an array via the addClick function. We set the boolean paint to true (we will

see why in a sec). Finally we update the canvas with the function redraw.

```
$('#canvas').mousedown(function(e){  
  var mouseX = e.pageX - this.offsetLeft;  
  var mouseY = e.pageY - this.offsetTop;  
  
  paint = true;  
  addClick(e.pageX - this.offsetLeft, e.pageY - this.offsetTop);  
  redraw();  
});
```

Mouse Move Event: Just like moving the tip of a marker on a sheet of paper, we want to draw on the canvas when our user is pressing down. The boolean paint will let us know if the virtual marker is pressing down on the paper or not. If paint is true, then we record the value. Then redraw.

```
$('#canvas').mousemove(function(e){  
  if(paint){  
    addClick(e.pageX - this.offsetLeft, e.pageY - this.offsetTop, true);  
    redraw();  
  }  
});
```

Mouse Up Event: Marker is off the paper; paint boolean will remember!

```
$('#canvas').mouseup(function(e){  
  paint = false;  
});
```

Mouse Leave Event: If the marker goes off the paper, then forget you!

```
$('#canvas').mouseleave(function(e){  
  paint = false;  
});
```

Here is the addClick function that will save the click position:

```
var clickX = new Array();
var clickY = new Array();
var clickDrag = new Array();
var paint;

function addClick(x, y, dragging)
{
    clickX.push(x);
    clickY.push(y);
    clickDrag.push(dragging);
}
```

The redraw function is where the magic happens. Each time the function is called the canvas is cleared and everything is redrawn. We could be more efficient and redraw only certain aspects that have been changed, but let's keep it simple.

We set a few stroke properties for the color, shape, and width. Then for every time we recorded as a marker on paper we are going to draw a line.

```
function redraw(){
    canvas.width = canvas.width; // Clears the canvas

    context.strokeStyle = "#df4b26";
    context.lineJoin = "round";
    context.lineWidth = 5;

    for(var i=0; i < clickX.length; i++)
    {
        context.beginPath();
        if(clickDrag[i] && i){
            context.moveTo(clickX[i-1], clickY[i-1]);
        }else{
```

```
        context.moveTo(clickX[i]-1, clickY[i]);  
    }  
    context.lineTo(clickX[i], clickY[i]);  
    context.closePath();  
    context.stroke();  
}  
}
```

Simple Drawing Canvas Demo

Give it a try:

Clear the canvas:

Add Colors

Let's give our user some color choices. To do so, all we need to do is add declare a few global variables and update our redraw function.

Declare four color variables: colorPurple, colorGreen, colorYellow, colorBrown

with corresponding hex color values, a variable to store the current color: curColor, and an array to match the chosen color when the user clicked the canvas clickColor.

```
var colorPurple = "#cb3594";  
var colorGreen = "#659b41";  
var colorYellow = "#ffcf33";  
var colorBrown = "#986928";
```

```
var curColor = colorPurple;  
var clickColor = new Array();
```

The addClick function needs to be updated to record the chosen color when the user clicks.

```
function addClick(x, y, dragging)  
{  
    clickX.push(x);  
    clickY.push(y);  
    clickDrag.push(dragging);  
    clickColor.push(curColor);  
}
```

Since the color can vary now, we need to update the redraw function so it references the color that was active when the user clicked. We move the line about strokeStyle into the for loop and assign it to color value in the new array clickColor that corresponds to the user's clickage.

```
function redraw(){  
    /* context.strokeStyle = "#df4b26"; */  
    context.lineJoin = "round";  
    context.lineWidth = 5;  
  
    for(var i=0; i < clickX.length; i++)  
    {
```

```
context.beginPath();  
if(clickDrag[i] && i){  
    contex.moveTo(clickX[i-1], clickY[i-1]);  
}else{  
    context.moveTo(clickX[i]-1, clickY[i]);  
}  
context.lineTo(clickX[i], clickY[i]);  
context.closePath();  
context.strokeStyle = clickColor[i];  
context.stroke();  
}  
}
```

Demo Colors

Try it with color choices:

- Clear the canvas:
- Choose a color:

Add Sizes

Just like we added colors, let's add some sizes to chose from: "small", "normal", "large", and "huge".

We need a couple more global variables: clickSize and curSize.

```
var clickSize = new Array();  
var curSize = "normal";
```

The addClick function needs to be updated to record the chosen size when the user clicks.

```
function addClick(x, y, dragging)  
{  
  clickX.push(x);  
  clickY.push(y);  
  clickDrag.push(dragging);  
  clickColor.push(curColor);  
  clickSize.push(curSize);  
}
```

Update the redraw function to handle the new sizes.

```
function redraw(){  
  context.lineJoin = "round";  
  class="highlight delete"> /* context.lineWidth = 5; */  
  for(var i=0; i < clickX.length; i++)  
  {  
    context.beginPath();  
    if(clickDrag[i] && i){  
      context.moveTo(clickX[i-1], clickY[i-1]);  
    }else{
```

```
        context.moveTo(clickX[i]-1, clickY[i]);
    }
    context.lineTo(clickX[i], clickY[i]);
    context.closePath();
    context.strokeStyle = clickColor[i];
    context.lineWidth = radius;
    context.stroke();
}
}
```

Demo Sizes

Try it with different sizes:

- Clear the canvas:
- Choose a color:
- Choose a size:

Add Tools

Crayon, Marker, Eraser. Three tools. Let's make them.

The two global variables we need are: clickTool and curTool.

```
var clickTool = new Array();  
var curTool = "crayon";
```

The addClick function needs to be updated to record the chosen tool when the user clicks.

```
function addClick(x, y, dragging)  
{  
  clickX.push(x);  
  clickY.push(y);  
  clickDrag.push(dragging);  
  if(curTool == "eraser"){  
    clickColor.push("white");  
  }else{  
    clickColor.push(curColor);  
  }  
  clickColor.push(curColor);  
  clickSize.push(curSize);  
}
```

Update the redraw function to handle the new tools.

```
function redraw(){  
  context.lineJoin = "round";  
  for(var i=0; i < clickX.length; i++)  
  {  
    context.beginPath();  
    if(clickDrag[i] && i){  
      context.moveTo(clickX[i-1], clickY[i-1]);  
    }else{  
      context.moveTo(clickX[i]-1, clickY[i]);  
    }  
  }
```

```
}
context.lineTo(clickX[i], clickY[i]);
context.closePath();
context.strokeStyle = clickColor[i];
context.lineWidth = radius;
context.stroke();
}
if(curTool == "crayon") {
    context.globalAlpha = 0.4;
    context.drawImage(crayonTextureImage, 0, 0, canvasWidth, canvasHeight);
}
context.globalAlpha = 1;
}
```

Demo Tools

Try it with different tools:

- Clear the canvas:
- Choose a color:
- Choose a size:
- Choose a tool:

Add Outline

Coloring books provide an outline of something to color: a cute puppy or a hopping bunny. I chose a watermelon duck.

First declare a variable `outlineImage`.

```
var outlineImage = new Image();
```

Load the outline image.

```
function prepareCanvas(){  
  ...  
  
  outlineImage.src = "images/watermelon-duck-outline.png";  
}
```

Update the `redraw` function to draw the outline image using the canvas context's `drawImage` method. Its parameters are the image object we loaded, the position we want to draw the image, and the dimensions of the image.

```
function redraw(){
```

...

```
context.drawImage(outlineImage, drawingAreaX, drawingAreaY, drawingAreaWidth, drawingAreaHeight);  
}
```

Demo Outline

Give it try:

- Clear the canvas:
- Choose a color:
- Choose a size:
- Choose a tool:



Final Details

We are almost there! These last details are optional: restrict the drawing area to a rectangle on the canvas and use the rest of the canvas for our gui (aka buttons).

Let's mask the canvas so all drawing is within a the drawing area. We use the clip method with the save and restore methods. This method is not currently supported by Internet Explorer.

```
function redraw()
{
    ...

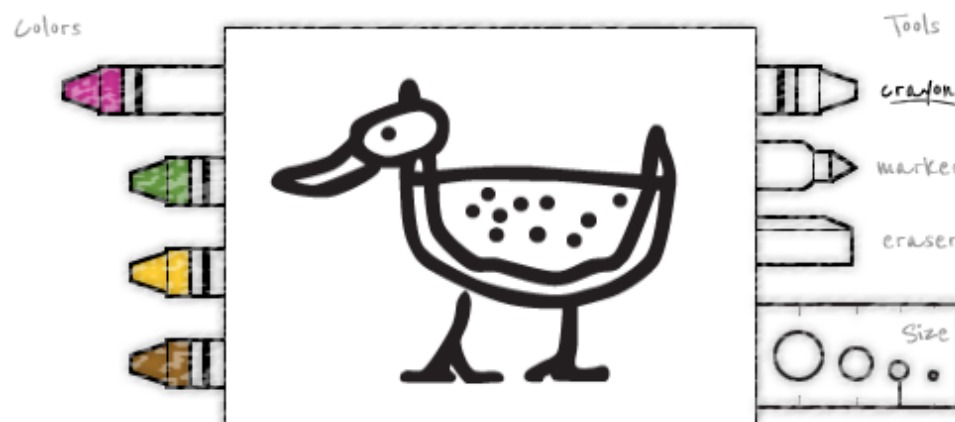
    context.save();
    context.beginPath();
    context.rect(drawingAreaX, drawingAreaY, drawingAreaWidth, drawingAreaHeight);
    context.clip();

    var radius;
    var i = 0;
    for(; i < clickX.length; i++)
    {
        ...
    }
    context.restore();
    ...
} // end redraw function
```

The last detail is to move all those buttons on our canvas. It involves loading images and displaying them based on our user interaction. I used standard JavaScript techniques so I won't bore you with the details (but it is included in the source code if you are interested). And there you have it!

Complete Demo

Now that we have created an HTML5 canvas drawing app, lets take a break and draw!



Download Source Code

- Download HTML5 Canvas Drawing App (zip format): [html5-canvas-drawing-app.zip](#)

Related Articles

- [HTML5 Game Character](#)
- [HTML5 Canvas Paint Bucket](#)
- [HTML5 Canvas Example](#)

Share this Article





-
-
-
-
-

[Home](#) | [Articles](#) | [Briefs](#) | [Works](#) | [Resume](#) | [About](#) | [Contact](#) | [RSS](#) | [Sitemap](#)

This site williammalone.com and its contents [copyright](#) © 2012 [William Malone](#). It is not affiliated with, sponsored by, or endorsed by any entity other than yours truly: [William Malone](#).