farbtastic / farbtastic.js

```
/**
 * Farbtastic Color Picker 1.2
 * © 2008 Steven Wittens
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 */

jQuery.fn.farbtastic = function (callback) {
  $.farbtastic(this, callback);
  return this;
};

jQuery.farbtastic = function (container, callback) {
  var container = $(container).get(0);
  return container.farbtastic || (container.farbtastic = new jQuery._farbtastic(contain
}

jQuery._farbtastic = function (container, callback) {
  // Store farbtastic object
  var fb = this;

  // Insert markup
  $(container).html('<div class="farbtastic"><div class="color"></div><div class="wheel
  var e = $('.farbtastic', container);
  fb.wheel = $('.wheel', container).get(0);
  // Dimensions
  fb.radius = 84;
  fb.square = 100;
  fb.width = 194;

  // Fix background PNGs in IE6
  if (navigator.appVersion.match(/MSIE [0-6]\./)) {
    $('*', e).each(function () {
      if (this.currentStyle.backgroundImage != 'none') {
        var image = this.currentStyle.backgroundImage;
        image = this.currentStyle.backgroundImage.substring(5, image.length - 2);
        $(this).css({
          'backgroundImage': 'none',
          'filter': "progid:DXImageTransform.Microsoft.AlphaImageLoader(enabled=true, s
        });
      }
    });
  }

  /**
   * Link to the given element(s) or callback.
   */
  fb.linkTo = function (callback) {
    // Unbind previous nodes
    if (typeof fb.callback == 'object') {
      $(fb.callback).unbind('keyup', fb.updateValue);
    }

    // Reset color
    fb.color = null;

    // Bind callback or elements
    if (typeof callback == 'function') {
      fb.callback = callback;
    }
    else if (typeof callback == 'object' || typeof callback == 'string') {
      fb.callback = $(callback);
      fb.callback.bind('keyup', fb.updateValue);
      if (fb.callback.get(0).value) {
        fb.setColor(fb.callback.get(0).value);
      }
    }
    return this;
  }
  fb.updateValue = function (event) {
    if (this.value && this.value != fb.color) {
      fb.setColor(this.value);
    }
  }

  /**
   * Change color with HTML syntax #123456
   */
  fb.setColor = function (color) {
    var unpack = fb.unpack(color);
    if (fb.color != color && unpack) {
      fb.color = color;
    fb.rgb = unpack;
    fb.hsl = fb.RGBToHSL(fb.rgb);
    fb.updateDisplay();
  }
  return this;
}

/**
 * Change color with HSL triplet [0..1, 0..1, 0..1]
 */
fb.setHSL = function (hsl) {
  fb.hsl = hsl;
  fb.rgb = fb.HSLToRGB(hsl);
  fb.color = fb.pack(fb.rgb);
  fb.updateDisplay();
  return this;
}

/////////////////////////////////////////////////////

/**
 * Retrieve the coordinates of the given event relative to the center
 * of the widget.
 */
fb.widgetCoords = function (event) {
  var x, y;
  var el = event.target || event.srcElement;
  var reference = fb.wheel;

  if (typeof event.offsetX != 'undefined') {
    // Use offset coordinates and find common offsetParent
    var pos = { x: event.offsetX, y: event.offsetY };

    // Send the coordinates upwards through the offsetParent chain.
    var e = el;
    while (e) {
      e.mouseX = pos.x;
      e.mouseY = pos.y;
      pos.x += e.offsetLeft;
      pos.y += e.offsetTop;
      e = e.offsetParent;
    }

    // Look for the coordinates starting from the wheel widget.
    var e = reference;
    var offset = { x: 0, y: 0 }
    while (e) {
      if (typeof e.mouseX != 'undefined') {
        x = e.mouseX - offset.x;
        y = e.mouseY - offset.y;
```

```
          break;
        }
        offset.x += e.offsetLeft;
        offset.y += e.offsetTop;
        e = e.offsetParent;
      }

      // Reset stored coordinates
      e = el;
      while (e) {
        e.mouseX = undefined;
        e.mouseY = undefined;
        e = e.offsetParent;
      }
    }
    else {
      // Use absolute coordinates
      var pos = fb.absolutePosition(reference);
      x = (event.pageX || 0*(event.clientX + $('html').get(0).scrollLeft)) - pos.x;
      y = (event.pageY || 0*(event.clientY + $('html').get(0).scrollTop)) - pos.y;
    }
    // Subtract distance to middle
    return { x: x - fb.width / 2, y: y - fb.width / 2 };
  }

  /**
   * Mousedown handler
   */
  fb.mousedown = function (event) {
    // Capture mouse
    if (!document.dragging) {
      $(document).bind('mousemove', fb.mousemove).bind('mouseup', fb.mouseup);
      document.dragging = true;
    }

    // Check which area is being dragged
    var pos = fb.widgetCoords(event);
    fb.circleDrag = Math.max(Math.abs(pos.x), Math.abs(pos.y)) * 2 > fb.square;

    // Process
    fb.mousemove(event);
    return false;
  }

  /**
   * Mousemove handler
   */
  fb.mousemove = function (event) {
    // Get coordinates relative to color picker center
    var pos = fb.widgetCoords(event);
    // Set new HSL parameters
    if (fb.circleDrag) {
      var hue = Math.atan2(pos.x, -pos.y) / 6.28;
      if (hue < 0) hue += 1;
      fb.setHSL([hue, fb.hsl[1], fb.hsl[2]]);
    }
    else {
      var sat = Math.max(0, Math.min(1, -(pos.x / fb.square) + .5));
      var lum = Math.max(0, Math.min(1, -(pos.y / fb.square) + .5));
      fb.setHSL([fb.hsl[0], sat, lum]);
    }
    return false;
  }

  /**
   * Mouseup handler
   */
  fb.mouseup = function () {
    // Uncapture mouse
    $(document).unbind('mousemove', fb.mousemove);
    $(document).unbind('mouseup', fb.mouseup);
    document.dragging = false;
  }

  /**
   * Update the markers and styles
   */
  fb.updateDisplay = function () {
    // Markers
    var angle = fb.hsl[0] * 6.28;
    $('.h-marker', e).css({
      left: Math.round(Math.sin(angle) * fb.radius + fb.width / 2) + 'px',
      top: Math.round(-Math.cos(angle) * fb.radius + fb.width / 2) + 'px'
    });

    $('.sl-marker', e).css({
      left: Math.round(fb.square * (.5 - fb.hsl[1]) + fb.width / 2) + 'px',
      top: Math.round(fb.square * (.5 - fb.hsl[2]) + fb.width / 2) + 'px'
    });

    // Saturation/Luminance gradient
    $('.color', e).css('backgroundColor', fb.pack(fb.HSLToRGB([fb.hsl[0], 1, 0.5])));

    // Linked elements or callback
    if (typeof fb.callback == 'object') {
      // Set background/foreground color
      $(fb.callback).css({
        backgroundColor: fb.color,
        color: fb.hsl[2] > 0.5 ? '#000' : '#fff'
      });

      // Change linked value
      $(fb.callback).each(function() {
        if (this.value && this.value != fb.color) {
          this.value = fb.color;
        }
      });
    }
    else if (typeof fb.callback == 'function') {
      fb.callback.call(fb, fb.color);
    }
  }

  /**
   * Get absolute position of element
   */
  fb.absolutePosition = function (el) {
    var r = { x: el.offsetLeft, y: el.offsetTop };
    // Resolve relative to offsetParent
    if (el.offsetParent) {
      var tmp = fb.absolutePosition(el.offsetParent);
      r.x += tmp.x;
      r.y += tmp.y;
    }
    return r;
  };

  /* Various color utility functions */
  fb.pack = function (rgb) {
    var r = Math.round(rgb[0] * 255);
    var g = Math.round(rgb[1] * 255);
    var b = Math.round(rgb[2] * 255);
    return '#' + (r < 16 ? '0' : '') + r.toString(16) +
           (g < 16 ? '0' : '') + g.toString(16) +
           (b < 16 ? '0' : '') + b.toString(16);
  }

  fb.unpack = function (color) {
    if (color.length == 7) {
      return [parseInt('0x' + color.substring(1, 3)) / 255,
        parseInt('0x' + color.substring(3, 5)) / 255,
        parseInt('0x' + color.substring(5, 7)) / 255];
    }
    else if (color.length == 4) {
      return [parseInt('0x' + color.substring(1, 2)) / 15,
        parseInt('0x' + color.substring(2, 3)) / 15,
        parseInt('0x' + color.substring(3, 4)) / 15];
    }
  }
```

```
fb.HSLToRGB = function (hsl) {
  var m1, m2, r, g, b;
  var h = hsl[0], s = hsl[1], l = hsl[2];
  m2 = (l <= 0.5) ? l * (s + 1) : l + s - l*s;
  m1 = l * 2 - m2;
  return [this.hueToRGB(m1, m2, h+0.33333),
      this.hueToRGB(m1, m2, h),
      this.hueToRGB(m1, m2, h-0.33333)];
}

fb.hueToRGB = function (m1, m2, h) {
  h = (h < 0) ? h + 1 : ((h > 1) ? h - 1 : h);
  if (h * 6 < 1) return m1 + (m2 - m1) * h * 6;
  if (h * 2 < 1) return m2;
  if (h * 3 < 2) return m1 + (m2 - m1) * (0.66666 - h) * 6;
  return m1;
}

fb.RGBToHSL = function (rgb) {
  var min, max, delta, h, s, l;
  var r = rgb[0], g = rgb[1], b = rgb[2];
  min = Math.min(r, Math.min(g, b));
  max = Math.max(r, Math.max(g, b));
  delta = max - min;
  l = (min + max) / 2;
  s = 0;
  if (l > 0 && l < 1) {
    s = delta / (l < 0.5 ? (2 * l) : (2 - 2 * l));
  }
  h = 0;
  if (delta > 0) {
    if (max == r && max != g) h += (g - b) / delta;
    if (max == g && max != b) h += (2 + (b - r) / delta);
    if (max == b && max != r) h += (4 + (r - g) / delta);
    h /= 6;
  }
  return [h, s, l];
}

// Install mousedown handler (the others are set on the document on-demand)
$('*', e).mousedown(fb.mousedown);

  // Init color
fb.setColor('#000000');

// Set linked elements/callback
if (callback) {
  fb.linkTo(callback);
}
```

```
}
```