www.PacktPub.com

AJAX Whiteboard

This mini-book is an additional resource for *AJAX and PHP: Building Responsive Web Applications* (Packt Publishing, 2006). Updated versions of this document, along with other useful resources, can be found at `http://ajaxphp.packtpub.com`.

This case study was written by Bogdan Brinzarea, co-author of *AJAX and PHP: Building Responsive Web Applications*.

Bogdan Brinzarea has a strong background in Computer Science holding a Masters and Bachelor Degree at the Automatic Control and Computers Faculty of the Politehnica University of Bucharest, Romania, and also an Auditor diploma at the Computer Science department at Ecole Polytechnique, Paris, France.

His main interests cover a wide area from embedded programming, distributed and mobile computing and new web technologies. Currently, he is employed as an Alternative Channels Specialist at Banca Romaneasca, Member of National Bank of Greece, where he is responsible for the Internet Banking project and coordinates other projects related to security applications and new technologies to be implemented in the banking area.

A Few Words on AJAX

AJAX is a complex phenomenon that means different things to different people. Computer users appreciate that their favorite websites are now friendlier and feel more responsive. Web developers learn new skills that empower them to create sleek web applications with little effort. Indeed, everything sounds good about AJAX!

At its roots, AJAX is a mix of technologies that lets you get rid of the evil page reload, which represents the dead time when navigating from one page to another. Eliminating page reloads is just one step away from enabling more complex features into websites, such as real-time data validation, drag and drop, and other tasks that weren't traditionally associated with web applications. Although the AJAX ingredients are mature (the `XMLHttpRequest` object, which is the heart of AJAX, was created by Microsoft in 1999), their new role in the new wave of web trends is very young, and we'll witness a number of changes before these technologies will be properly used to the best benefit of the end users. At the time of writing this book, the "AJAX" name is about just one year old.

AJAX isn't, of course, the answer to all the Web's problems, as the current hype around it may suggest. As with any other technology, AJAX can be overused, or used the wrong way. AJAX also comes with problems of its own: you need to fight with browser inconsistencies, AJAX-specific pages don't work on browsers without JavaScript, they can't be easily bookmarked by users, and search engines don't always know how to parse them. Also, not everyone likes AJAX. While some are developing enterprise architectures using JavaScript, others prefer not to use it at all. When the hype is over, most will probably agree that the middle way is the wisest way to go for most scenarios.

In *AJAX and PHP: Building Responsive Web Applications* we took a pragmatic and safe approach, by teaching relevant patterns and best practices that we think any web developer will need sooner or later. We teach you how to avoid the common pitfalls, how to write efficient AJAX code, and how to achieve functionality that is easy to integrate into current and future web applications, without requiring you to rebuild the whole solution around AJAX. You'll be able to use the knowledge you learn from this book right away, into your PHP web applications.

Introducing the AJAX Whiteboard

Although it isn't apparent, AJAX can draw as well, and in this case study we'll see how. Our goal here is to build an online whiteboard application, where your visitors can draw and publicly express their artistic skills.

Implementing a whiteboard certainly isn't the best real-world scenario for using AJAX, because technologies such as Flash or Java do a much better job at handling complex graphics. However, implementing a whiteboard with AJAX allows us to study some of the more sensitive areas of AJAX development, such as:

- Drawing graphics using specific algorithms and `div` elements
- Efficiently packaging data for client-server communication
- Synchronizing the same view among many clients
- Optimizing the database storage and SQL queries for increasing performance

For the whiteboard application we'll build in this chapter, we have the following requirements:

- Responsiveness is a key factor: We seek for low times of response from a real-time application.
- Support for concurrent access, enabling collaborative work: Two or more users need to be able to access the application at the same time.
- Data sharing: All participants can interact with the whiteboard differently, yet they must share the same view of the board, just as with the blackboard used in a classroom. (All of us need to have the same view in order to participate in the classroom, don't we?)

One might say that these are the characteristics of a desktop application, more or less. One well-known example of a whiteboard is NetMeeting from Microsoft. Popular instant messenger applications such as Yahoo Messenger and MSN Messenger support this feature as well.

After having established the base for developing this application, we need to examine the context: What does JavaScript have to say about graphics?

JavaScript and HTML Graphics

JavaScript isn't very effective at drawing with HTML because HTML wasn't built to draw graphics; hence we need to find "special" ways to draw into the web page.

It's worth taking a look at the **Vector Graphics Library** that can be found at `http://www.walterzorn.com/jsgraphics/jsgraphics_e.htm`. This library provides a large number of basic functions for drawing, such as: `setColor`, `setStroke`, `drawLine`, `drawPolyline`, `drawRect`, `fillRect`, `drawPolygon`, `fillPolygon`, `drawEllipse`, `fillEllipse`, `setFont`, `drawString`, and `drawRectString`. These functions provide most functionality that you need for basic drawings.

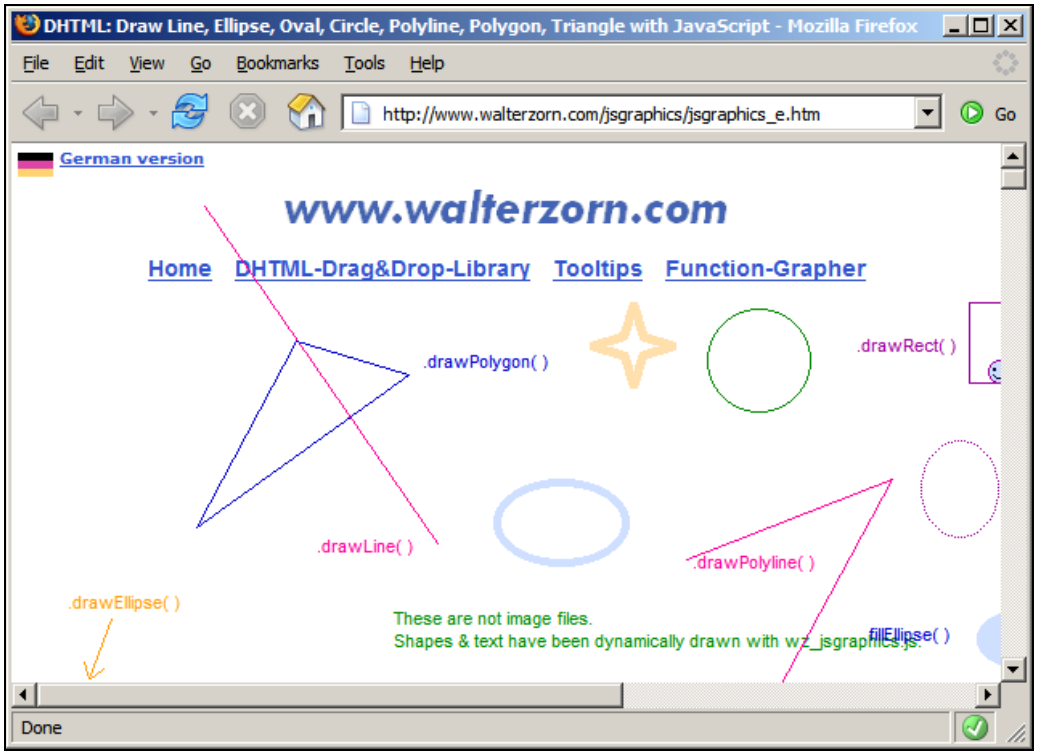

Figure 1: Vector Graphics Library

The technique behind this library resides on building many `div` elements that form the desired shapes. Besides using fast Bresenham algorithms to compute the shapes, it also tries to minimize the number of `div` elements that are used for creating a shape by grouping pixels together.

The alternative to using `div` elements in drawing is using **Scalable Vector Graphics** (**SVG**). You'll learn how to work with SVG in Chapter 7, *AJAX Real-Time Charting with SVG*.

Implementing the AJAX Whiteboard

Before getting our hands on the source code and on the theoretical aspects of this application, let's see what the main technical features are:

- Generate `div` elements on the client to enable drawing
- Use a database back end to store the pixel coordinates
- Allow drawing with multiple colors
- Make the solution work with multiple users at the same time
- Implement a design that permits easy extension

In order to have this example working, you need to enable support for the GD PHP library. The installation instructions in Appendix A include support for the GD library.

The application you'll create will look as the one in Figure 2. You can access its online version at `http://ajaxphp.packtpub.com/`. The application works much more smoothly in Firefox, which apparently has a faster DOM. (Remember though that it will never work as well as a Java or Flash whiteboard. The scope of the case study is to analyze various programming techniques.)

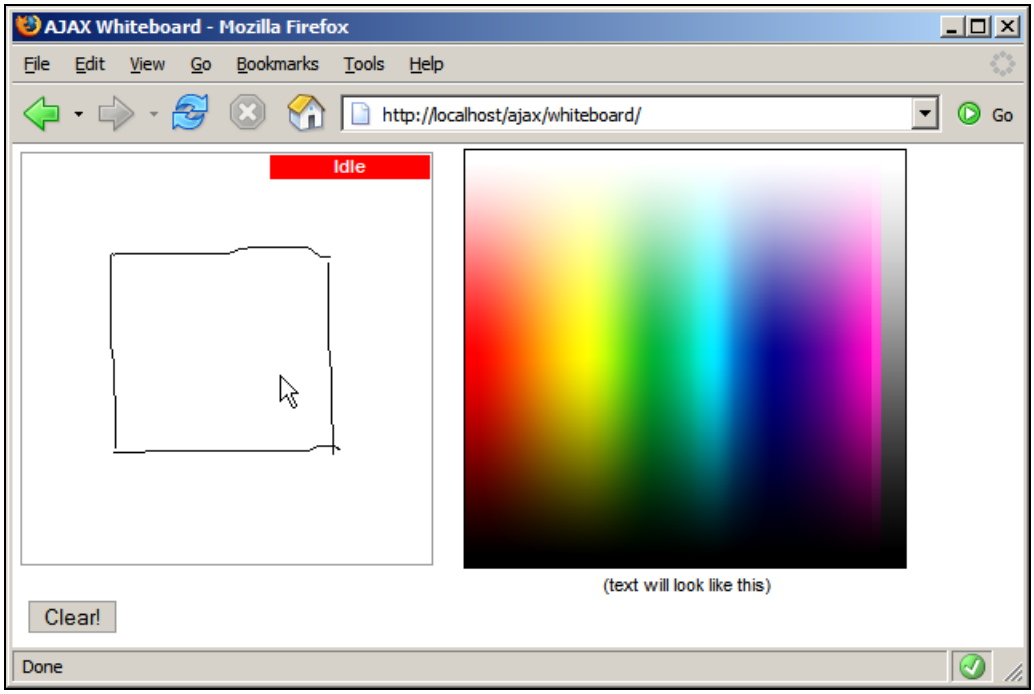

Figure 2: AJAX Whiteboard in action

Now, let's get to work and see what can be done with AJAX for the graphics.

Time for Action—AJAX Whiteboard

> This example assumes that you have installed and configured your system as shown in Appendix A of *AJAX and PHP: Building Responsive Applications*. The book's Appendices can be freely downloaded from the book's mini-site at `http://ajaxphp.packtpub.com`.

1. Connect to the `ajax` database, and create a table named `whiteboard` with the following code:

```
CREATE TABLE whiteboard
(
  whiteboard_id int NOT NULL auto_increment,
  session_id char(32) NOT NULL,
  color char(6) NOT NULL,
  offsetx1 int NOT NULL,
  offsetx2 int NOT NULL,
  offsety1 int NOT NULL,
  offsety2 int NOT NULL,
  length int,
  PRIMARY KEY (whiteboard_id)
) TYPE=MEMORY;
```

2. In your `ajax` folder, create a new folder named `whiteboard`.
3. Copy the `palette.png` file from the code download to the `whiteboard` folder.
4. In the `whiteboard` folder, create a file named `config.php`, and add the database configuration code to it (change these values to match your configuration):

```
<?php
// defines database connection data
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

5. Now add the standard error-handling file, `error_handler.php`:

```
<?php
// set the user error handler method to be error_handler
set_error_handler('error_handler', E_ALL);
// error handler function
function error_handler($errNo, $errStr, $errFile, $errLine)
{
  // clear any output that has already been generated
  if(ob_get_length()) ob_clean();
  // output the error message
  $error_message = 'ERRNO: ' . $errNo . chr(10) .
                   'TEXT: ' . $errStr . chr(10) .
                   'LOCATION: ' . $errFile .
                   ', line ' . $errLine;
  echo $error_message;
  // prevent processing any more PHP scripts
  exit;
}
?>
```

6. Now create the server script `whiteboard.php`, and add this code to it:

```
<?php
// reference the file containing the Whiteboard class
require_once('whiteboard.class.php');
// create a new Whiteboard instance
$wb = new Whiteboard();
// clear the whiteboard if it's too loaded
$wb->checkLoad();
// retrieve the parameters from the request
$mode = $_POST['mode'];
$session_id = $_POST['session_id'];
// if the client connects for the first time, we generate its session id
if($session_id == '')
    $session_id = md5(uniqid());
// initialize the last known ID to 0
$last_id = 0;
// if the operation is DeleteAndRetrieve
if($mode == 'DeleteAndRetrieve')
  // clear the whiteboard
  $wb->clearWhiteboard();
// if the operation is SendAndRetrieve
elseif($mode == 'SendAndRetrieve')
{
  // retrieve the new lines
  $lines = $_POST['lines'];
  // retrieve the id of the last line
  $last_id = $_POST['last_id'];
  // insert the new lines
  $wb->insertLines($lines, $session_id);
}
// if the operation is Retrieve
elseif($mode == 'Retrieve')
{
  // retrieve the id of the last line
  $last_id = $_POST['last_id'];
}
// clear the output
if(ob_get_length()) ob_clean();
// headers are sent to prevent browsers from caching
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // time in the past
header('Last-Modified: ' . gmdate( 'D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
// send the latest lines to client
echo $wb->getNewLines($last_id, $session_id);
?>
```

7. Create another file named `whiteboard.class.php`, and add this code to it:

```
<?php
// load configuration file
require_once('config.php');
// load error handling file
require_once('error_handler.php');
// class handles server-side whiteboard support functionality
class Whiteboard
{
  // database handler
  private $mMysqli;
  // define the maximum total length of all lines in the table
  private $mMaxLoad = 3000;

  /* constructor opens database connection */
  function __construct()
```

```
{
  $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                                    DB_DATABASE);
}

/* destructor, closes database connection */
function __destruct()
{
  $this->mMysqli->close();
}

/*
  The checkLoad method clears the whiteboard table if the total length
  of all lines in it is bigger than the predefined value
*/
public function checkLoad()
{
  // build the SQL query to get the total length of all lines
  $check_load = 'SELECT SUM(length) total_length FROM whiteboard';
  // execute the SQL query
  $result = $this->mMysqli->query($check_load);
  $row = $result->fetch_array(MYSQLI_ASSOC);
  // if the total length of all lines exceeds the maximum total length
  // we delete all the entries in the table
  if($row['total_length'] > $this->mMaxLoad)
  {
    // clear the whiteboard
    $this->clearWhiteboard();
    // flag that we cleared the whiteboard
    return true;
  }
  else
    return false; // we didn't clear the whiteboard
}

/*
  The insertLines method inserts new lines into the database
  - $lines contains the lines as received by the server as a
           string with separators
  - $session_id  contains the id of the client's session
*/
public function insertLines($lines, $session_id)
{
  // check to see if there are new lines sent
  if($lines)
  {
    // the lines are comma separated
    $array_lines = explode(',', $lines);
    // process each line
    for($i=0; $i<count($array_lines); $i++)
    {
      // each line is received in the form:
      // color:offsetx1:offsety1:offsetyx2:offsety2
      list($color, $offsetx1, $offsety1, $offsetx2, $offsety2) =
        explode(':', $array_lines[$i]);
      // escape the input data
      $color = $this->mMysqli->real_escape_string($color);
      $offsetx1 = $this->mMysqli->real_escape_string($offsetx1);
      $offsetx2 = $this->mMysqli->real_escape_string($offsetx2);
      $offsety1 = $this->mMysqli->real_escape_string($offsety1);
      $offsety2 = $this->mMysqli->real_escape_string($offsety2);
      // build the SQL query to insert a new line
      $insert_line = 'INSERT INTO whiteboard ' . '
      (offsetx1, offsety1, offsetx2, offsety2, length, color, session_id) '
       .'VALUES (' . $offsetx1 . ',' . $offsety1 . ',' . $offsetx2 . ','
       . $offsety2 . ',' .
```

```
          sqrt(pow(($offsetx1-$offsetx2), 2) + pow(($offsety1-$offsety2), 2))
          . ',"'.$color. '","' . $session_id . '")';
        // execute the SQL query
        $this->mMysqli->query($insert_line);
      }
    }
  }

  /*
    The getNewLines method returns the lines that appeared since the last
    update
    - $id contains the id of the last updated line
    - $session_id contains the id of the client's session
  */
  public function getNewLines($id, $session_id)
  {
    // escape the variable data
    $id = $this->mMysqli->real_escape_string($id);
    $session_id = $this->mMysqli->real_escape_string($session_id);
    // retrieve the latest ID in the database
    $last_id = $this->getLastId();
    // build the SQL query to get the latest lines
    $get_lines =
      'SELECT whiteboard_id, color, offsetx1, offsety1, offsetx2, offsety2 '
      . 'FROM whiteboard ' .
      'WHERE whiteboard_id IN ' .
      '  (SELECT MAX(whiteboard_id) ' .
      '   FROM whiteboard ' .
      '   WHERE whiteboard_id> ' . $id .
      '   GROUP BY offsetx1, offsety1, offsetx2, offsety2) ' .
      'AND session_id<>"' . $session_id . '" ' .
      'ORDER BY whiteboard_id ASC';
    // execute the SQL query
    $result = $this->mMysqli->query($get_lines);
    // build the XML response
    $response = '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
    $response .= '<response>';
    // get the last id in the database
    $response .= '<last_id>' . $last_id . '</last_id>';
    // send back the session id
    $response .= '<session_id>' . $session_id . '</session_id>';
    // retrieve all lines and send them back to the client
    while($row = $result->fetch_array(MYSQLI_ASSOC))
    {
      // get the details
      $id = $row['whiteboard_id'];
      $offsetx1 = $row['offsetx1'];
      $offsety1 = $row['offsety1'];
      $offsetx2 = $row['offsetx2'];
      $offsety2 = $row['offsety2'];
      $color = $row['color'];
      // generate the XML element
      $response .= '<id>' . $id . '</id>' .
                   '<color>' . $color . '</color>' .
                   '<offsetx1>' . $offsetx1 . '</offsetx1>' .
                   '<offsety1>' . $offsety1 . '</offsety1>' .
                   '<offsetx2>' . $offsetx2 . '</offsetx2>' .
                   '<offsety2>' . $offsety2 . '</offsety2>';
    }
    // close the database connection as soon as possible
    $result->close();
    // finish the XML response
    $response .= '</response>';
    // return the response
    return $response;
```

```
  }

  /*
     The clearWhiteboard method truncates the data table
  */
  public function clearWhiteboard()
  {
    // build the SQL query to truncate the whiteboard table
    $clear_wb = 'TRUNCATE TABLE whiteboard';
    // execute the SQL query
    $this->mMysqli->query($clear_wb);
  }

  /*
    The getLastId method returns the most recent whiteboard_id
  */
  private function getLastId()
  {
    // build the SQL query to retrieve the last id in the whiteboard table
    $get_last_id = 'SELECT whiteboard_id ' .
                   'FROM whiteboard ' .
                   'ORDER BY whiteboard_id DESC ' .
                   'LIMIT 1';
    // execute the SQL query
    $result = $this->mMysqli->query($get_last_id);
    // check to see if there are any results
    if($result->num_rows > 0)
    {
      // fetch the row containing the result
      $row = $result->fetch_array(MYSQLI_ASSOC);
      // return the xml element
      return $row['whiteboard_id'];
    }
    else
      // there are no records in the database so we return 0 as the id
      return '0';
  }
//end class Whiteboard
}
?>
```

8. Create another file named `get_color.php`, and add this code to it:

```
<?php
// the name of the image file
$imgfile='palette.png';
// load the image file
$img=imagecreatefrompng($imgfile);
// obtain the coordinates of the point clicked by the user
$offsetx=$_GET['offsetx'];
$offsety=$_GET['offsety'];
// get the clicked color
$rgb = ImageColorAt($img, $offsetx, $offsety);
$r = ($rgb >> 16) & 0xFF;
$g = ($rgb >> 8) & 0xFF;
$b = $rgb & 0xFF;
// return the color code
printf('#%02s%02s%02s', dechex($r), dechex($g), dechex($b));
?>
```

9. Create a new file named `index.html`, and add this code to it:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>AJAX Whiteboard</title>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="whiteboard.css" rel="stylesheet" type="text/css" />
    <script type="text/javascript" src="whiteboard.js"></script>
  </head>
  <body>
    <noscript>
      This application requires JavaScript!!
    </noscript>
    <div id="whiteboard" onmousemove="javascript:handleMouseMove(event);"
                         onmouseout="javascript:handleMouseOut(event);">
    </div>
    <div id="status"></div>
    <table id="content">
      <tr>
        <td id="emptycell">
        </td>
        <td id="colorpicker">
          <img src="palette.png" id="palette" alt="Color Palette"
               border="1" onclick="getColor(event);"/>
          <br />
          <input id="color" type="hidden" readonly="true" value="#000000" />
          <span id="sampleText">
            (text will look like this)
          </span>
        </td>
      </tr>
    </table>
    <div>
      <input type="button" onclick="clearWhiteboard();" value="Clear!" />
    </div>
  </body>
</html>
```

10. Create another file named `whiteboard.css`, and add this code to it:

```
body
{
  font-family: helvetica, sans-serif;
  margin: 0px;
  padding: 0px;
  font-size: 11px
}

table.content
{
  width: 100%;
  height: 100%;
  margin-left: 5px;
  margin-top: 5px;
  border-spacing: 0px;
  padding: 0px;
  border: 0px
}

#emptycell
{
  width: 275px;
  height: 250px
}

#status
{
  left: 160px;
  top: 6px;
  background: red;
  color: white;
```

```
  font-family: helvetica, sans-serif;
  font-weight: bold;
  border: white 1px solid;
  border-spacing: 0px;
  padding: 0px;
  width: 100px;
  height: 15px;
  position: absolute;
  visibility: hidden;
  text-align: center
}

#whiteboard
{
  left: 5px;
  top: 5px;
  border: #999 1px solid;
  border-spacing: 0px;
  padding: 0px;
  width: 256px;
  height: 256px;
  position: absolute
 }

.simple
{
  background-color: white;
  width: 1px;
  height: 1px;
  font-size: 1px;
  margin: 0px;
  padding: 0px;
  position: absolute;
  z-index: 1
}

input
{
  margin-left: 10px;
  border: #999 1px solid
}

#colorpicker
{
  text-align: center
}
```

11. Create another file named `whiteboard.js`, and add this code to it:

```
/* URL to the page that updates the whiteboard */
var whiteboardURL = "whiteboard.php";
/* URL to the page that retrieves the requested RGB color of a point */
var getColorURL = "get_color.php";
/* mouse coordinates in page */
var mouseX = 0;
var mouseY = 0;
/* flag that indicates if the user has pressed the mouse (drawing) */
var isDrawing = false;
/* flag that specifies if the clear button has been pressed */
var isWhiteboardErased = false;
/* the id of the last line drawn as a server update in the client's
whiteboard*/
var lastDrawnLineId = 0;
/* the id in the server's database of the last line received as update */
var lastDbLineId = 0;
```

```
/* the id of the session as generated at the first request to the server
*/
var sessionId = "";
/* the array containing the updated lines */
var wbUpdatedLinesArray;
/* the number of lines in the wbUpdatedLinesArray */
var linesCount = 0;
/* the whiteboard object */
var oWhiteboard;
/* the control containing the RGB color */
var oColor;
/* whiteboard's position and dimensions */
var wbOffsetLeft;
var wbOffsetTop;
var wbOffsetWidth;
var wbOffsetHeight;
/* the RGB code of the current color */
var currentColor = "#000000";
/* when set to true, display detailed error messages */
var debugMode = true;
/* the status message object */
var oStatus;
/* the onload event is overwritten by our init function */
window.onload = init;
/* the start point offset coordinates for a line */
var lineStartPointOffsetX=0;
var lineStartPointOffsetY=0;
/* the XMLHttpRequest object used to send and retrieve updated lines */
var xmlHttpUpdateWhiteboard= createXmlHttpRequestObject();
/* the XMLHttpRequest object to retrieve the selected color */
var xmlHttpGetColor = createXmlHttpRequestObject();

/* creates an XMLHttpRequest instance */
function createXmlHttpRequestObject()
{
  // will store the reference to the XMLHttpRequest object
  var xmlHttp;
  // this should work for all browsers except IE6 and older
  try
  {
    // try to create XMLHttpRequest object
    xmlHttp = new XMLHttpRequest();
  }
  catch(e)
  {
    // assume IE6 or older
    var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                    "MSXML2.XMLHTTP.5.0",
                                    "MSXML2.XMLHTTP.4.0",
                                    "MSXML2.XMLHTTP.3.0",
                                    "MSXML2.XMLHTTP",
                                    "Microsoft.XMLHTTP");
    // try every prog id until one works
    for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
    {
      try
      {
        // try to create XMLHttpRequest object
        xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
      }
      catch (e) {}
    }
  }
  // return the created object or display an error message
  if (!xmlHttp)
    alert("Error creating the XMLHttpRequest object.");
```

```
  else
    return xmlHttp;
}

/*
  function that shows the status of the whiteboard
*/

function showStatus(message)
{
  // update the status message
  oStatus.innerHTML = message;
  oStatus.style.visibility = "visible";
}

/*
 function that draws a line using the Bresenham algorithm
*/
function lineBresenham(x0, y0, x1, y1,color)
{
  var dy = y1 - y0;
  var dx = x1 - x0;
  var stepx, stepy;
  if (dy < 0)
  {
    dy = -dy;
    stepy = -1;
  }
  else
  {
    stepy = 1;
  }
  if (dx < 0)
  {
    dx = -dx;
    stepx = -1;
  }
  else
  {
    stepx = 1;
  }
  dy <<= 1;
  dx <<= 1;
  createPoint(x0, y0,color);
  if (dx > dy)
  {
    fraction = dy - (dx >> 1);
    while (x0 != x1)
    {
      if (fraction >= 0)
      {
        y0 += stepy;
        fraction -= dx;
      }
      x0 += stepx;
      fraction += dy;
      if(isInWhiteboard(x0,y0))
        createPoint(x0, y0,color);
      else
        return;
    }
  }
  else
  {
    fraction = dx - (dy >> 1);
    while (y0 != y1)
```

```
      {
        if (fraction >= 0)
        {
          x0 += stepx;
          fraction -= dy;
        }
        y0 += stepy;
        fraction += dx;
        if(isInWhiteboard(x0, y0))
          createPoint(x0, y0,color);
        else
          return;
      }
    }
  }

  /*
    function that checks if a point is within the whiteboard's boundaries
  */
  function isInWhiteboard(x,y)
  {
    return (x < wbOffsetWidth-1 && x > 1 && y < wbOffsetHeight-1 && y > 1);
  }

  /*
    function that handles the mouseout event
  */
  function handleMouseOut(e)
  {
    // get mouse coordinates
    getMouseXY(e);
    // compute the current point's offset coordinates
    lineStopPointOffsetX = mouseX - wbOffsetLeft;
    lineStopPointOffsetY = mouseY - wbOffsetTop;
    // check to see if the event occurs in the whiteboard
    // when the mouse is over other divs
    if(isInWhiteboard(lineStopPointOffsetX, lineStopPointOffsetY))
      return;
    // check to see if we are drawing
    if(isDrawing == true)
    {
      // reset the drawing flag
      isDrawing = false;
      // draw the line by clipping it to the whiteboard's boundaries
      lineBresenham(lineStartPointOffsetX, lineStartPointOffsetY,
                    lineStopPointOffsetX, lineStopPointOffsetY,
                    currentColor);
      // add the line
      addLine(lineStartPointOffsetX, lineStartPointOffsetY,
              lineStopPointOffsetX, lineStopPointOffsetY,
              currentColor);
      // the start point of the new line is the stop point of the last line
      lineStartPointOffsetX = lineStopPointOffsetX;
      lineStartPointOffsetY = lineStopPointOffsetY;
    }
  }

  /*
   function for handling the mousedown event
  */
  function handleMouseDown(e)
  {
    // set the flag for drawing
    isDrawing = true;
    // retrieve the event object
    if(!e) e = window.event;
```

```
  // get mouse coordinates
  getMouseXY(e);
  //set the start point's offset coordinates for the new line
  lineStartPointOffsetX = mouseX - oWhiteboard.offsetLeft;
  lineStartPointOffsetY = mouseY - oWhiteboard.offsetTop;
}

/*
 function for handling the mouseup event
*/
function handleMouseUp(e)
{
  // set the flag for drawing
  isDrawing = false;
  // retrieve the event object
  if(!e) e = window.event;
  // get mouse coordinates
  getMouseXY(e);
  // set the stop point for the line
  lineStopPointOffsetX = mouseX - oWhiteboard.offsetLeft;
  lineStopPointOffsetY = mouseY - oWhiteboard.offsetTop;
  // draw the current line
  lineBresenham(lineStartPointOffsetX, lineStartPointOffsetY,
                lineStopPointOffsetX, lineStopPointOffsetY, currentColor);
  // add the current line
  addLine(lineStartPointOffsetX, lineStartPointOffsetY,
          lineStopPointOffsetX, lineStopPointOffsetY, currentColor);
}

/*
    the function handles the mousemove event inside the whiteboard
*/
function handleMouseMove(e)
{
  // check to see if we are drawing
  if(isDrawing)
  {
    // retrieve the event object
    if(!e) e = window.event;
    // retrieve the mouse coordinates
    getMouseXY(e);
    // set the stop point's offset coordinates for the current line
    lineStopPointOffsetX = mouseX - oWhiteboard.offsetLeft;
    lineStopPointOffsetY = mouseY - oWhiteboard.offsetTop;
    // draw the current line
    lineBresenham(lineStartPointOffsetX, lineStartPointOffsetY,
                  lineStopPointOffsetX, lineStopPointOffsetY, currentColor);
    // add the line to the array of lines
    addLine(lineStartPointOffsetX,lineStartPointOffsetY,
            lineStopPointOffsetX,lineStopPointOffsetY,currentColor);
    // set the start point's offset coordinates as
    // the current stop point's offset coordinates
    lineStartPointOffsetX=lineStopPointOffsetX;
    lineStartPointOffsetY=lineStopPointOffsetY;
  }
}

/*
  the function adds a line to the array of drawn lines
*/
function addLine(offsetX1, offsetY1, offsetX2, offsetY2, color)
{
  var newLine = color.substring(1,7) + ":" + offsetX1 + ":" +
                offsetY1 + ":" + offsetX2 + ":" + offsetY2;
  wbUpdatedLinesArray[linesCount++] = newLine;
}
```

```
/*
  function that draws a point on the whiteboard
*/
function createPoint(offsetX, offsetY, color)
{
  // create a new div for the point
  oDiv = document.createElement("div");
  // set the attributes
  oDiv.className = "simple";
  // set its color
  if(color == "")
    newColor = currentColor;
  else
    newColor = color;
  // change color
  oDivStyle = oDiv.style;
  oDivStyle.backgroundColor = newColor;
  // set its position
  oDivStyle.left = offsetX + "px";
  oDivStyle.top = offsetY + "px";
  // add the point to the whiteboard
  oWhiteboard.appendChild(oDiv);
}

/*
 function that initiates the whiteboard
*/
function init()
{
  // initiate the whiteboard, color and status objects
  oWhiteboard = document.getElementById("whiteboard");
  oColor = document.getElementById("color");
  oStatus= document.getElementById("status");
  // retrieve whiteboard dimensions and position
  wbOffsetWidth = oWhiteboard.offsetWidth;
  wbOffsetHeight = oWhiteboard.offsetHeight;
  wbOffsetLeft = oWhiteboard.offsetLeft;
  wbOffsetTop = oWhiteboard.offsetTop;
  // initialize the array of updated lines and the new lines count
  wbUpdatedLinesArray = new Array();
  linesCount = 0;
  // handle events
  oWhiteboard.setAttribute("onmousedown", "handleMouseDown(event);");
  oWhiteboard.setAttribute("onmouseup", "handleMouseUp(event);");
  if(oWhiteboard.onmousedown)
  {
    oWhiteboard.onmousedown=handleMouseDown;
    oWhiteboard.onmouseup=handleMouseUp;
  }
  // start updating the whiteboard
  updateWhiteboard();
}


/*
  function that clears the whiteboard
*/
function clearWhiteboard()
{
  // shows clearing status
  showStatus("Clearing...");
  // clear variables
  lastDrawnLineId = 0;
  linesCount = 0;
  isWhiteboardErased = true;
  // delete all whiteboard's nodes
```

```
  while(oWhiteboard.hasChildNodes())
      oWhiteboard.removeChild(oWhiteboard.lastChild);
}

/*
  the function sends the updated lines to the server
*/
function updateWhiteboard()
{
  // continue only if we have a XMLHttpRequest object to work with
  if(xmlHttpUpdateWhiteboard)
  {
    try
    {
      // let the user know what happens
      showStatus("Updating...");
      // continue only if the XMLHttpRequest object isn't busy
      if (xmlHttpUpdateWhiteboard.readyState == 4 ||
          xmlHttpUpdateWhiteboard.readyState == 0)
      {
        // we build the request's parameters
        params = "";
        // check to see if we erased the whiteboard
        if(isWhiteboardErased == true)
        {
          params += "session_id=" + sessionId + "&mode=DeleteAndRetrieve";
          isWhiteboardErased = false;
        }
        else
        {
          // check to see we have lines to send
          if(linesCount > 0)
          {
            // build the params string
            params="session_id=" + sessionId +
                   "&mode=SendAndRetrieve" +
                   "&last_id=" + lastDrawnLineId +
                   "&lines=";
            // add all lines as parameters
            for(i=0; i<linesCount; i++)
            {
              params += wbUpdatedLinesArray[i];
              params += (i < linesCount-1) ? "," : "";
            }
             // reset the new lines count to 0
            linesCount=0;
          }
          else
          {
            // no lines to send
            params="session_id=" + sessionId +
                   "&mode=Retrieve&" +
                   "last_id=" + lastDrawnLineId;
          }
        }
        // initiate the request
        xmlHttpUpdateWhiteboard.open("POST", whiteboardURL, true);
        xmlHttpUpdateWhiteboard.setRequestHeader("Content-Type",
                                    "application/x-www-form-urlencoded");
        xmlHttpUpdateWhiteboard.onreadystatechange =
                                                handleUpdatingWhiteboard;
        xmlHttpUpdateWhiteboard.send(params);
      }
      // if the XMLHttpRequest object is busy with another request,
      // try again later
      else
```

```
        {
          // we will check again in 1 second
          setTimeout("updateWhiteboard();", 1000);
        }
      }
      catch(e)
      {
        alert("Can't connect to server:\n" + e.toString());
      }
    }
    else
    {
      alert("The XMLHttpRequest object is null !");
    }
  }

  /*
    function that handles the server's response to the whiteboard update
  */
  function handleUpdatingWhiteboard()
  {
    //if the request is completed
    if (xmlHttpUpdateWhiteboard.readyState == 4)
    {
      //if the HTTP response is ok
      if (xmlHttpUpdateWhiteboard.status == 200)
      {
        try
        {
          // process the server's response
          displayUpdates();
        }
        catch(e)
        {
          // display the error message
          alert("Error updating the whiteboard: \n" + e.toString() + "\n" +
                xmlHttpUpdateWhiteboard.responseText);
        }
      }
      else
      {
        alert("There was a problem when updating the whiteboard :\n" +
              xmlHttpUpdateWhiteboard.statusText);
      }
    }
  }

  /*
    display the new lines retrieved from the server
  */
  function displayUpdates()
  {
    // retrieve the response in text format to check if it's an error
    response = xmlHttpUpdateWhiteboard.responseText;
    // server error?
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
      throw(response.length == 0 ? "Can't update the whiteboard!" :
                                   response);
    // update the status message
    showStatus("Drawing...");
    // retrieve the document element
    response = xmlHttpUpdateWhiteboard.responseXML.documentElement;
    // we retrieve from the XML response the parameters
    sessionId =
```

```
    response.getElementsByTagName("session_id").item(0).firstChild.data;
  newLastDbLineId =
             parseInt(response.getElementsByTagName("last_id").item(0).
                      firstChild.data);
  // if the whiteboard has been cleared by another client
  // we need to clear our own whiteabord
  if(newLastDbLineId < lastDbLineId)
  {
    clearWhiteboard(oWhiteboard);
    isWhiteboardErased = false;
  }
  else
  // if new lines have been drawn by others we should also draw them
  if(newLastDbLineId>lastDbLineId)
  {
    // retrieve the lines' parameters
    idArray= response.getElementsByTagName("id");
    colorArray= response.getElementsByTagName("color");
    offsetX1Array= response.getElementsByTagName("offsetx1");
    offsetY1Array=response.getElementsByTagName("offsety1");
    offsetX2Array= response.getElementsByTagName("offsetx2");
    offsetY2Array=response.getElementsByTagName("offsety2");
    // draw the new lines
    if(idArray.length>0)
      updateLines(idArray,colorArray,offsetX1Array,offsetY1Array,
                  offsetX2Array,offsetY2Array);
  }
  // keep the id of the last line in the database
  lastDbLineId = newLastDbLineId;
  // update status message
  showStatus("Idle");
  // restart sequence after 1 second
  setTimeout("updateWhiteboard();", 1000);
}

//   function that draws the lines retrieved from the server
function updateLines(idArray, colorArray, offsetX1Array, offsetY1Array,
                                          offsetX2Array, offsetY2Array)
{
  // we process all the lines
  for(var i=0; i<idArray.length; i++)
  {
    // draw the line
    lineBresenham(parseInt(offsetX1Array[i].firstChild.data),
                  parseInt(offsetY1Array[i].firstChild.data),
                  parseInt(offsetX2Array[i].firstChild.data),
                  parseInt(offsetY2Array[i].firstChild.data),
                  "#"+colorArray[i].firstChild.data);
  }
  // we set the lastDrawnLineId to the value of the id of
  // the last line retrieved from the server
  lastDrawnLineId=idArray[i-1].firstChild.data;
}

/*
  function that computes the mouse coordinates relative to the palette
  and calls the server to retrieve the RGB code
*/
function getColor(e)
{
  // gets current mouse position
  getMouseXY(e);
  // initialize the offset position with
  // the mouse's current position in window
  var offsetX = mouseX;
  var offsetY = mouseY;
```

```
  var oPalette=document.getElementById("palette");
  var oTd=document.getElementById("colorpicker");
  // compute the offset position in our window
  if (window.ActiveXObject)
  {
    offsetX = window.event.offsetX;
    offsetY = window.event.offsetY;
  }
  else
  {
    offsetX -= oPalette.offsetLeft+oTd.offsetLeft;
    offsetY -= oPalette.offsetTop+oTd.offsetTop;
  }
  // continue only if we have valid XMLHttpRequest object
  if(xmlHttpGetColor)
  {
    try
    {
      if (xmlHttpGetColor.readyState == 4 || xmlHttpGetColor.readyState == 0)
      {
        params="?offsetx="+offsetX+"&offsety="+offsetY;
        xmlHttpGetColor.open("GET",getColorURL+params, true);
        xmlHttpGetColor.onreadystatechange = handleGettingColor;
        xmlHttpGetColor.send(null);
      }
    }
    catch(e)
    {
      alert("Can't connect to server:\n" + e.toString());
    }
  }
}

/* function that handles the http response */
function handleGettingColor()
{
  // if the process is completed, decide what to do with the returned data
  if (xmlHttpGetColor.readyState == 4)
  {
    // only if HTTP status is "OK"
    if (xmlHttpGetColor.status == 200)
    {
      try
      {
        //change the color
        changeColor();
      }
      catch(e)
      {
        // display the error message
        alert(e.toString() + "\n" + xmlHttpGetColor.responseText);
      }
    }
    else
    {
      alert("There was a problem retrieving the color:\n" +
            xmlHttpGetColor.statusText);
    }
  }
}

/* function that changes the color used for displaying our messages */
function changeColor()
{
  response=xmlHttpGetColor.responseText;
  // server error?
```

```
    if (response.indexOf("ERRNO") >= 0
        || response.indexOf("error:") >= 0
        || response.length == 0)
      throw(response.length == 0 ? "Can't change color!" : response);
    // change color
    var oSampleText=document.getElementById("sampleText");
    oColor.value=response;
    oSampleText.style.color=response;
    currentColor = "#" + oColor.value.substring(1,7);
}

/* function that computes the mouse's coordinates in page */
function getMouseXY(e)
{
    if(document.all)
    {
      mouseX = window.event.x + document.body.scrollLeft;
      mouseY = window.event.y + document.body.scrollTop;
    }
    else
    {
      mouseX =  e.pageX;
      mouseY =  e.pageY;
    }
}
```

12. Finally, load `http://localhost/ajax/whiteboard/` and test your code. Feel free to even open more browser windows, and draw in all of them with different colors to see that it really works. Figure 3 shows once again this program in action:

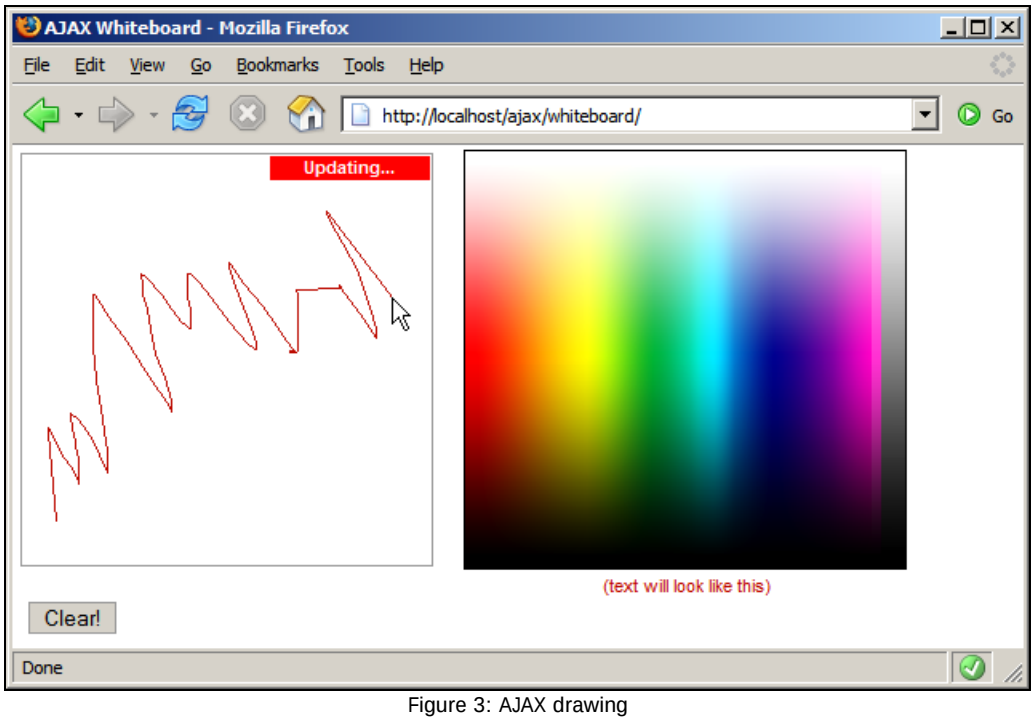

Figure 3: AJAX drawing

What just happened?

Having seen how it looks and what it does, let's see what hides behind these windows!

The database contains a new detail that you haven't met before in this book: the `MEMORY` table type. Tables of this type are stored in memory and have rows of fixed length. These two characteristics allow them to be very fast. These tables can be used only in certain circumstances, because their data is temporary, and they lose their contents if the database server is shut down or restarted.

In the `index.html` file, the `whiteboard div` is used for defining the area in which `mousedown`, `mouseup`, `mousemove`, and `mouseout` events are captured. The last two events are specified as attributes to the HTML tag `div`, the others being added dynamically through JavaScript.

The `status div` element is used for displaying the application's status: `updating` (when a server request is initiated), `drawing` (when the updates are being rendered on the whiteboard), `clearing` (when the clear button is pressed or when somebody else clears it), and `idle` (when the updating phase is completed).

The `palette.png` file allows picking up a color to use when drawing. By treating its `click` event and by using AJAX, we tell the server what coordinate was clicked, and the server responds with the color code.

`whiteboard.css` contains the stylesheet for our application. The `whiteboard` style is the one being applied to our whiteboard. The whiteboard has an absolute position in the page, this solution being chosen in order to avoid cross-browser problems related to relative positions of the mouse within the whiteboard.

The `simple` style is initially applied to all the points that are being drawn on the whiteboard. Each point is represented by a `div` object and the simple style makes it look like a point by giving it a 1 by 1 pixel size. By modifying the width and height attributes, you can easily modify the size of the point in the whiteboard.

Next, we move to `whiteboard.js`, the file containing the JavaScript part for our application. First, you need to see how the whiteboard has been implemented and the reasons for doing it in this way. The sample code that we use for the whiteboard is easy to extend, and can be the base for add-ons containing different vector graphics, such as SVG (an example of using this technology with AJAX is provided in Chapter 7 of *AJAX with PHP: Building Responsive Web Applications*)

For the purpose of this chapter we built a very simple whiteboard, implemented as a table of 256 by 256 pixels. The pixel is also the unit used for drawing on the whiteboard. The code has been written so that the size of the whiteboard could be changed to any value without having to modify the PHP code source. In fact, all you have to do is to play a little with the CSS styles presented above.

A table of 256 by 256 contains 65,536 pixels. Wow, a lot of pixels to draw on! What we want to obtain is an application that draws when the user clicks the mouse and starts fooling with the pointer within the drawing board. As with other drawing programs, the drawing procedure is not called for each pixel. Instead a line is drawn between two consecutive mouse movements, if the left mouse button is pressed, so the drawing is made of lines between consecutive points representing consecutive mouse movements. If we draw slowly the difference is minor, but when

we move the mouse fast in the whiteboard you will get a line between two points that are relatively far away from each other. This is due to the fact that the mouse movement is not detected for each pixel, and the event's frequency isn't that great in JavaScript.

Let's see how the mechanism of updating the whiteboard works. Because we are dealing with a lot of pixels, it's not optimal to retain the status of all the pixels drawn on the whiteboard. Instead, we retain only the lines during two consecutive updates on the server. Each line is represented by the two points. Instead of having 65,537 pixels to retain, we have a maximum of 50 pairs of points (lines) as could be observed when testing the application. This means less than 1% of the initial number of pixels. We could call this performance improvement!

In order to draw the lines between two consecutive points we use the Bresenham algorithm (`http://en.wikipedia.org/wiki/Bresenham's_line_algorithm`) which proves to be the best for drawing lines. The points used to draw lines are simulated as `div` elements of 1 by 1 pixel, as you can see by analyzing the stylesheet file.

Since the JavaScript client knows how to draw lines between points, in the database we only store lines (as two pairs of pixel coordinates), and let the clients do the math and draw them. Each second, an HTTP request is initiated asking the server to send the new lines that were drawn since the last update. The server will reply by sending only the new lines drawn by other users.

Each line is represented by five parameters:

- `offsetX` of the start point of the line
- `offsetY` of the start point of the line
- `offsetX` of the stop point of the line
- `offsetY` of the stop point of the line
- Its color

In the database we also store the line's length.

Another issue that's worth discussing is how the server knows about which client sends the updates. A possible answer would be to use a session ID that uniquely identifies the client. The problem that arose during the tests was that on PHP5 installed as an ISAPI module on IIS5 the session IDs were shared between different clients when the PHP session mechanism was involved. The problem mentioned above occurred only on Mozilla Firefox 1.0.x, while the other browsers were working fine. Instead of using this default mechanism for identifying the client, we came up with another solution: a session ID is generated by the server the first time the client checks for updates; this ID is then sent to the server with each request to identify the client.

The `init` function initiates the array of updated lines, adds mouse events (`mousedown` and `mouseup`) for the whiteboard object, retrieves the whiteboard's position and dimensions, retrieves the status and color object and calls the `updateWhiteboard` function.

The `createPoint` function creates a point as a `div` element that is appended to the whiteboard element. The color and its position are specified as parameters. The position is given as an offset to the whiteboard element.

The `isInWhiteboard` function checks to see if the point given as parameter is inside the whiteboard (if it is not, we don't draw the point). The function that takes care of drawing a line is `lineBresenham`. It receives the endpoints of the line and the color of the line as parameters and draws a line between the endpoints. The `addLine` function is the one that adds a new line to the array of existing lines; this array will be sent to the server on the next update so it can update the other clients.

The drawing begins when the client clicks the mouse and with the mouse button down starts drawing. The drawing ends when the mouse button is no longer clicked or when the mouse exits the whiteboard area. The functions handling these behaviors are: `handleMouseMove`, `handleMouseOut`, `handleMouseDown`, and `handleMouseUp`.

The `handleMouseDown` function occurs when the user presses the mouse button. The function sets the `isDrawing` flag to mark the event and saves the point where the event occurs. This point is in fact one endpoint of the first line that the user draws in the current process. The other endpoints of lines are determined in the other mouse event handling functions.

The function `handleMouseMove` is triggered by the mouse movement inside the whiteboard. If the user is currently drawing, then the current point where the event occurred is the other endpoint of the current line that needs to be drawn. The current line is drawn and added to the array of lines to be sent to the server as updates. The current point is also one endpoint of the next line to be drawn.

The `handleMouseUp` function is triggered when the user releases the mouse button earlier pressed. All we have to do is to mark the event by setting the `isDrawing` flag to false and to draw the last line. The point where this event occurred is one endpoint of the line. The current line is drawn and added to the array of lines to be sent to the server as updates.

The `handleMouseOut` function is triggered when the mouse exits the whiteboard area or when the mouse is over one of the `div` elements inside the whiteboard. So, we need to check if the mouse really exits the whiteboard area or if the mouse is temporally over a `div` element.

The `clearWhiteboard` function resets the whiteboard completely. It sets the status accordingly to "Clearing", sets a flag to notify the server of the event, resets the number of updated lines, and deletes the whiteboard's content.

The `clearWhiteboard` function is responsible for emptying the whiteboard when a user chooses to erase it or when a message from the server says the whiteboard has been erased by someone else. In order to erase the whiteboard, all we need to do is to remove all the child elements of the whiteboard element. Since every point is a child `div` element of the whiteboard, we clear the whiteboard by deleting all its children.

The `updateWhiteboard` function is responsible for sending, receiving, and deleting messages. The function sets the `mode` parameter according to the current operation that is performed (sending and receiving messages, receiving messages, deleting and receiving messages). The other parameters sent to the server are: the client's session ID, the ID of the last line received as update and the clear flag set to true if the whiteboard has been cleared or false otherwise. The change of state of the HTTP request object is handled by the `handleUpdatingWhiteboard` function. A timeout that calls the `updateWhiteboard` procedure is set at 1 second after receiving a response from the server and when sending an update is not possible. The status is updated to "Updating".

The `handleUpdatingWhiteboard` function checks to see when the request to the server is completed and if no errors occurred when the `displayUpdates` function is called.

The `displayUpdates` function deals with the server's response that contains the latest updated lines. The status is changed to "Drawing". The function converts the received XML to arrays of elements. If the ID of the last line inserted in the database by us or by another user is smaller than our previous ID of the last line, it means that the whiteboard has been erased. The client first erases the whiteboard and then the `updateLines` function is called with the updated lines as parameters. At the end of the function the status is set to "Idle", now that all the operations involving the server are over. A timer set at one second reinitiates the cycle.

The `updateLines` function takes the new lines as parameters and updates the whiteboard accordingly. The ID of the last line received as an update is saved in order to know what lines to retrieve during the next request to server.

Let's move on to the server side of the application by first presenting the `whiteboard.php` file.

The server handles clients' updates as mentioned in the following few steps:

1. It calls the `checkLoad` method so that the server checks to see if the total length of all lines in the table is bigger than a maximum total length and if so it erases them. The total length of all lines is calculated by adding the lengths of all the lines inside the whiteboard. (When adding a new line to the database, we also store its length, and that calculated value comes in handy now.)
2. It retrieves the `mode` and `session_id` parameters passed by the client.
3. If the client connects for the first time, the server generates a unique session ID.
4. It takes one of the following actions according to the `mode` parameter:
    - `DeleteAndRetrieve`: The client cleared the whiteboard, and the table that contains the whiteboard as lines is truncated.
    - `SendAndRetrieve`: The new lines updated by the client are inserted in the database.
    - `Retrieve`: The `last_id` of the last line received as an update for the current client is retrieved.
5. It sends back to the client the session ID, the ID of the last line inserted in the database and the latest lines since the last request.

The first step is quite simple, in that it ensures that the total length of all lines on the whiteboard is reasonable at all times. When the total length is too big, the solution becomes very slow because the web browser can't handle the load, so we chose to automatically clear the whiteboard when it becomes too loaded.

The next step is the simplest; it retrieves variables from client's request. No problems here.

Next, we move to the session ID part. As we have seen earlier, instead of relying on the classical solution based on PHP's session ID, we use our own solution. The `uniqid` function is based on the current time in milliseconds. By applying the `md5` hashing function we reach our goal: we get a unique session ID for identifying each client.

The third step is determined by the `mode` parameter.

The `DeleteAndRetrieve` value of the `mode` parameter means that the client has cleared the whiteboard. In this case the server truncates the table containing the whiteboard. This solution has been chosen because it offers us a possibility to easily determine if a clear operation occurred. By truncating a table, the primary key is reset. If we check to see if the records still exist with the ID smaller than the ID of the last updated point by the client, then we know that a clear event occurred.

The `SendAndRetrieve` value of the `mode` parameter instructs the server to insert into the whiteboard table the client's updated lines and to retrieve the `last_id` parameter used to retrieve the last updated lines since the last request.

The `Retrieve` value of the `mode` parameter instructs the server only to retrieve the `last_id` parameter used later to retrieve the lines updated since the last request.

The last step is the one responsible for retrieving the latest lines updated by other clients and for building the response to the client.

The business logic behind the `whiteboard.php` file lies in the `whiteboard.class.php` file. It contains the `Whiteboard` class with several methods that are called from `whiteboard.php` depending on the operation needing to be performed.

The `insertLines` method inserts all new lines in the database. The new lines as well as the client's session ID are passed as parameters. The length of each line is calculated and saved into the table as well.

The `getLines` method retrieves new lines since the last line sent to that client (identified by its session ID and the ID of the last line received) and also gets the ID of the last line inserted in the database (by calling the `getLastId` method). All this information is put together in a XML message that is sent back to the client.

The `clearWhiteboard` method is the one that truncates the data table erasing all the information.

The `checkLoad` method ensures that the total length of the lines remains within reasonable limits. It calculates the total length by summing the lengths of all lines in the whiteboard. If this number exceeds a maximum number established by setting a member variable, the `clearWhiteboard` method is called.

The `config.php` file contains the database configuration parameters and the `error_handler.php` file contains the module for handling errors.

Summary

The chapter started by showing what a whiteboard is and what purposes it has. We learned about the current solutions for vector graphics and about what we could expect in the near feature. Then we developed, step by step, a freehand whiteboard with the possibility of picking a color for drawing.

After having analyzed the source code, the following features can be easily implemented:

- Snap shot of the current whiteboard in an image format such as PNG
- Vector graphics (lines, circles, ellipses, etc.)

These features can be implemented starting from the presented solution. We didn't include any of them in our chapter because we tried to focus on what the goal of this book is, and that's AJAX and because we wanted to have a simple an easy method to follow example.

Check Out the Book!

AJAX and PHP: Building Responsive Web Applications

by Cristian Darie, Bogdan Brinzarea, Filip Chereches-Tosa, Mihai Bucica

Published by Packt

Assuming a basic knowledge of PHP, XML, JavaScript and MySQL, this book will help you understand how the heart of AJAX beats and how the constituent technologies work together. After teaching the foundations, the book will walk you through numerous real-world case studies covering tasks you'll need for your own applications:

- Server-enabled form-validation page
- Online chat collaboration tool
- Customized type-ahead text entry solution
- Real-time charting using SVG
- Database-enabled, editable and customizable data grid
- RSS aggregator application
- Use the script.aculo.us JavaScript toolkit to build a drag&drop enabled sortable list

The appendices guide you through installing your working environment, using powerful tools that enable debugging, improving, and profiling your code and working with XSLT and XPath.

For more information, please visit: `http://www.packtpub.com/ajax_php/book`.