

Chapter 4.

SECTION II: BUILDING AJAX-FRIENDLY APPLICATIONS

Text Suggest

Any programming technique requires something called the 'AhhHa' quotient to help get it universally adopted. The Text Suggest or Auto Suggest is one such Ajax application that has helped Ajax blend into the main stream of web based programming rapidly.

Text still remains the most powerful way for humans to communicate with computers. With the advent of Instant Messaging, Blogging, Email and so on a user is required to type in data faster and in greater quantity than ever before. A significant number of users are now quick on the keyboard. However, there are still many users whose workflow experience on the web is simply a Click → Execute type of experience.

Typing in data quickly is an issue for users with limited typing experience and also with users having certain physical disabilities. Even for a fast typist, typing speed remains an issue, as most people think faster than they can type. Additionally, people often make mistakes when they type.

The solution to this problem is to craft a mechanism that suggests words or phrases which are likely to complete what the user's typing. If appropriate, the user can select the suggestion, thus avoid typing it in.

A drop down, combo box, is a type of data field used in traditional GUI forms. It is a combination of a text input with a dropdown list. A user can type any text and the current selection in the list will track what's been typed so far. The user is also free to choose an element from the drop down list, which will then be posted back to the text field. In some cases, the elements in drop down list constrain data, while in other cases it's there to provide suggestions only to the user.

Text Suggest can be considered as sort of "advanced combo box" become popular in recent years that comprises of free text entry, with some suggestions for completion at any time. Text Suggest became popular with I.E. 5, which auto-completed fields based on user history

Text Suggest works as follows:

- ❑ A standard **input field** is used along with an initially invisible **DIV** element that is created to contain suggestions as they are retrieved from the Web server. The input field needs an event handler to monitor the text it contains. This ensures, the list always highlights whichever suggestion is matching
- ❑ Upon each key press within the input field, the Browser checks whether anything has changed since the last request. If so, the Web server is passed a partial query as a GET-based XMLHttpRequest call and the changed information in the input field is passed to it
- ❑ The Web server then produces an ordered list of suggestions
- ❑ At the Browser end, the callback function picks up the suggestions and displays them to the user, in a format that allows them to be selected. Each entry has an event handler if clicked, the input field will be altered

The basic idea is that as a character is typed, the application suggests terms that came up as search results at the Web server. The first suggestion is filled into the textbox as a character is typed while a list of several suggestions appears as a dropdown list beneath the textbox.

This concept has been used in desktop applications for some time. The technique has been mapped to the Web purely because of Ajax.

Implementing Text Suggest in GUI

In order to understand the functionality of Text Suggest the best way is to implement it in GUI. The following part of the chapter demonstrates the implementation of Text Suggest in BookMaster GUI.

The focus:

- ❑ To suggest the book names starting with the first letter entered in the text box
- ❑ Once the suggested book name is selected retrieve that book's details such as the Image of its cover page, publisher's name, cost of the book and a synopsis from the BookMaster table on demand

The names of the books will be suggested as soon as the first letter is known. This will be done asynchronously. The book details will be fetched and displayed asynchronously when the name of the book is selected from the suggestion made. This is where Ajax is used.

After the page is crafted it will appear as shown in diagram 4.1 and diagram 4.2.

4. TEXT SUGGEST

Type in the first few characters of a book name:

Book Details:

Diagram 4.1: The HTML page with a text box

If the letter keyed in the text box produces a single suggestion then that suggestion appears in the text box suggestion along with the details of that particular book under **Book Details:** as shown in diagram 4.2.

Type in the first few characters of a book name:

Book Details:

Publisher Name:	SPD	
Book Cost:	550	

Book Synopsis:
 The book has been written to provide genuine domain knowledge to programmers who wish to learn web based, application development, using PHP as a front-end programming tool, Apache as the web server and Oracle 10g as a DBMS of choice all run on Linux.

Learning web development is done through a set of examples and is finally strongly reinforced by the development of a Personnel Management System.

Diagram 4.2: Single Suggestion

If the letter keyed in the text box produces more than one suggestion then a list of suggestions is displayed below the text box as shown in diagram 4.3.1. The details of a book will be shown under **Book Details:** only if a particular book is selected using left mouse click as shown in diagram 4.3.2.

Type in the first few characters of a book name:

- PHP 5.1 For Beginners
- Professional Oracle Projects

Book Details:

Diagram 4.3.1: Multiple suggestions

Type in the first few characters of a book name:

- PHP 5.1 For Beginners Left click
- Professional Oracle Projects

Book Details:

Publisher Name:	SPD
Book Cost:	650



Book Synopsis:
The book has been written to provide genuine domain knowledge to programmers who wish to learn web based, application development, using object oriented PHP 5.1 as the programming environment. IIS and Apache as the web servers of choice. With a clutch of DBMS as the data store underlay for the applications developed. All run on M.S. Windows and Linux.

Diagram 4.3.2: Book Details shown on left click

The data required to populate the suggestion text box and display book details will be retrieved from the MySQL table BookMaster. The data population has to be done without refreshing the entire HTML page. The data retrieval from the Web server will be performed asynchronously using Ajax technology. This means any database operation initiated by the user via the GUI will be sent as a request to the Web server as the background process. This is done using JavaScript.

4. TEXT SUGGEST

Expected Program Flow

- ❑ When a letter is typed in the textbox
- ❑ An XMLHttpRequest object is created and configured
- ❑ The XMLHttpRequest object makes a call to a Web server side, PHP script, to retrieve appropriate information from the BookMaster table for display
- ❑ The request is processed by the server side PHP script
- ❑ The server side PHP script returns the result in form of text
- ❑ The XMLHttpRequest object calls the callback() function (i.e. its listener) and processes the result
- ❑ The HTML DOM is updated
- ❑ The Browser immediately recognizes that DOM values has changed and updates that section of the HTML form which is bound to the DOM values. This immediately displays the information returned from the Web server and catches the user's attention. The entire HTML form is not refreshed

The entire system is built using **PHP** as the server-side scripting language, **MySQL** as the database of choice and **Ajax** as the client / server technology base.

Create the following table and save it in the **bookdb** database under the MySQL Db engine:

- ❑ Create the bookdb database (if it is not already existing):

```
CREATE DATABASE bookdb;
```

- ❑ Activate the bookdb database:

```
USE bookdb;
```

- ❑ Create the BookMaster table:

Column Name	Data Type	Width	Description
BookID	Integer		An identity number of the book
PublisherName	Varchar	50	The name of the publisher
BookName	Varchar	200	The name of the book
BookCategory	Varchar	200	The category to which the book belongs
BookSynopsis	Text		A synopsis of the book
BookCost	Integer		The cost of the book
BookPic	Varchar	50	The directory path of the book image

```
CREATE TABLE BookMaster(
    BookID Integer,
    PublisherName Varchar(50),
    BookName Varchar(200),
    BookCategory Varchar(200),
    BookSynopsis Text,
    BookCost Integer,
    BookPic Varchar(50));
```

- Insert the following records into the BookMaster table:

```
INSERT INTO BookMaster VALUES(1, 'SPD', 'Application Development With
Oracle And PHP On Linux For Beginners', 'Linux', 'The book has been written to
provide genuine domain knowledge to programmers who wish to learn web
based, application development, using PHP as a front-end programming tool,
Apache as the web server and Oracle 10g as a DBMS of choice all run on Linux.
<BR><BR>Learning web development is done through a set of examples and
is finally strongly reinforced by the development of a Personnel Management
System.', 550, 'images/image1.jpg');
```

```
INSERT INTO BookMaster VALUES(2, 'SPD', 'PHP 5.1 For Beginners',
'Language', 'The book has been written to provide genuine domain knowledge
to programmers who wish to learn web based, application development, using
object oriented PHP 5.1 as the programming environment. <BR><BR>IIS and
Apache as the web servers of choice. With a clutch of DBMS as the data store
underlay for the applications developed. All run on M.S. Windows and Linux.',
650, 'images/image2.jpg');
```

```
INSERT INTO BookMaster VALUES(3, 'SPD', 'MySQL 5 For Professionals',
'Database', 'The book has been written to provide an excellent grounding to
those who wish to learn ANSI SQL using MySQL 5. This book also has several
illustrative examples, which have a logical link between them.
<BR><BR>Learning of SQL is done through the construction of a Sales
Management System. The skills thus developed are strongly firm up via
hands on exercises based on the construction of a Human Resource
Management System.', 550, 'images/image3.jpg');
```

```
INSERT INTO BookMaster VALUES(4, 'BpB', 'Professional Oracle Projects',
'Database', 'The book has been written to provide genuine domain knowledge to
programmers who wish to learn professional project development using Oracle
Forms 6i / 9iDS (web based) as a front-end and Oracle 9iAS on Red Hat Linux
9.0 as the back-end. <BR><BR>Learning the project development (in Oracle
Forms 6i) is done through the construction of two projects: A Retail Banking
System and A Manufacturing And Sales System.', 195, 'images/image4.jpg');
```

4. TEXT SUGGEST

**INSERT INTO BookMaster VALUES(5, 'SPD', 'Database Concepts And Systems', 'Database', 'The book has been written to provide students an excellent grounding in Database Concepts and Systems. The book has several illustrative examples, which have a logical link between them.

Each set of examples helps build skills that will take the reader to the next set of examples, which in turn leads upwards until a strong programming foundation using ANSI SQL, the natural language of an RDBMS, has been established.', 275, 'images/image5.jpg');**

**INSERT INTO BookMaster VALUES(6, 'BpB', 'SQL, PL/SQL: The Programming Language Of Oracle', 'Database', 'The book has been written to provide an excellent grounding to those who wish to learn SQL and PL/SQL using Oracle.

Learning of SQL & PL/SQL is done through the construction of a Retail Banking System. The skills thus developed are implemented via hands on exercises based on the construction of a Sales Order System.', 250, 'images/image6.jpg');**

**INSERT INTO BookMaster VALUES(7, 'BpB', 'Moving From Windows To Linux', 'Linux', 'This book has been written to provide the reader an insight into how simple it is to use the Linux operating system and programming environment.

This book shows the user, who is comfortable with using Microsoft Windows and associated popular applications, how Linux works and how using it is similar in many ways to their current software. This is the reason the book is named "Moving From Windows To Linux".', 195, 'images/image7.jpg');**

**INSERT INTO BookMaster VALUES(8, 'BpB', 'Using StarOffice 7.0 On Linux', 'Linux', 'The book has been written to provide a sound working knowledge anyone looking for an excellent Office productivity enhancement product in the Linux domain.

the book has covered the functionality of all the menu and tool bar items for each of the major products in StarOffice 7.0. such as StarWriter, StarCalc, Star Impress, Making visiting cards, Creating mailing labels.', 195, 'images/image8.jpg');**

**INSERT INTO BookMaster VALUES(9, 'BpB', 'Programming In Visual Basic 6', 'Language', 'The book has been written to give an excellent grounding to those who wish to learn application development using Visual Basic.

Each set of examples helping build skills that will take the reader to the next set of examples, which in turn lead upwards until a strong programming foundation using Visual Basic has been established.', 150, 'images/image9.jpg');**

**INSERT INTO BookMaster VALUES(10, 'BpB', 'Chronicles Of Java On Linux', 'Language', 'The book has been written to provide excellent grounding to those who wish to learn Java.

Each set of examples helps build skills that will take the reader to the next set of examples, which in turn leads upwards until a strong programming foundation in Java using the natural language of MySQL has been established.', 295, 'images/image10.jpg');**

This process loads BookMaster with the appropriate test data.

Files Used

Form Code Block	index.html	The HTML page having one text box and the Book Details section
JavaScript Code Block	ajaxfunctions.js	<p>Holds generic Ajax functions that allow requesting to and receiving a response from the Web server using Ajax technology. These functions allow:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Creating and instantiating an XMLHttpRequest object <input type="checkbox"/> Sending request and receiving response
PHP Code Block	doSql.php	<p>Establishes a Db connection with the MySQL Db engine</p> <p>Retrieves book names for suggestions</p> <p>Retrieves the details of the book from the MySQL Db engine</p> <p>Forms a table structure using the HTML's TABLE tag, populates the TABLE cells with appropriate data and returns the same to the Browser that demanded it</p>

The Code Spec And Functionality

The index.html File

Code Spec:

```
<HTML>
  <HEAD>
    <TITLE>Auto Text Suggest Using Ajax with PHP & MySQL</TITLE>
    <SCRIPT TYPE="text/javascript" SRC="ajaxfunctions.js"> </SCRIPT>
  </HEAD>
```

Explanation:

A script file named ajaxfunctions.js is included to allow using Ajax functionality.

Code Spec:

```
<BODY BGCOLOR="pink">
  <TABLE WIDTH="90%">
    <TR>
      <TD>
        Type in the first few characters of a book name:
        <INPUT SIZE="100" TYPE="text" ID="txtAutoSuggest"
          NAME="txtAutoSuggest"
          onKeyUp="autocomplete(this, event);"/>
```

4. TEXT SUGGEST

Explanation:

The table's first row holds a text box named txtAutoSuggest where the user can key in characters for suggestions.

Code Spec:

```
<DIV ID="booksSuggested"></DIV>
<BR><BR>
```

Explanation:

A DIV named booksSuggested to display the suggestions if there are multiple suggestions.

Code Spec:

```
<STRONG>Book Details:</STRONG>
<HR>
<DIV ID="bookDetailsPanel"></DIV>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Explanation:

Another DIV named bookDetailsPanel to display the book details for the selected book.

All these DIVs currently hold nothing. The contents of these DIVs will be fetched by the server-side PHP script (doSql.php) and returned to the appropriate Ajax → JavaScript function which in turn will populate the DIV object using its innerHTML property.

The first DIV will hold multiple suggestions for the characters typed in.

The second DIV will hold a table structure populated with appropriate book details.

The ajaxfunctions.js File**Code Spec:**

```
function createRequestObject()
{
  var xmlhttp = false;
  try
  {
    xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (e)
  {
```

```

    try
    {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    catch (E)
    {
        xmlhttp = false;
    }
}
if (!xmlhttp && typeof XMLHttpRequest != 'undefined')
{
    xmlhttp = new XMLHttpRequest();
}
return xmlhttp;
}

```

Explanation:

The above code spec creates a method that is broken down into the following steps:

- Using the **try** block:
 - An attempt is made to spawn an **ActiveXObject** object using **new XMLHttpRequest ("MSXML2.XMLHTTP")** into a memory variable named **xmlhttp**
 - If this fails then, an object of type **Microsoft.XMLHTTP** is spawned into a memory variable named **xmlhttp**
 - If both the attempts fail then, an object of **XMLHttpRequest** is spawned using **new XMLHttpRequest()** into a memory variable named **xmlhttp**

At the end of this process, **xmlhttp** should reference a valid **XMLHttpRequest** object, no matter what browser the users run.

Code Spec:

```

function autocomplete(sender, ev)
{
    if(sender.value)
    {
        /* Processing Alphanumeric keystrokes */
        if ((ev.keyCode >= 48 && ev.keyCode <= 57) || (ev.keyCode >= 65 &&
            ev.keyCode <= 90))
        {
            /* Preparing a request to the server */
            var httpreq = createRequestObject();
            /* Assigning the value received (keyed in by the user) in
            name=value pair format to the parms memory variable. */
            var parms = "val=" + sender.value;
            /* Creating a copy of the value received (keyed in by the
            user) */
            var original_text = sender.value;

```

4. TEXT SUGGEST

```

/* Requesting the server */
httpreq.open('GET', 'doSql.php?' + parms, true);

/* The callback functionality */
httpreq.onreadystatechange = function()
{
    if (httpreq.readyState == 4)
    {
        if (httpreq.status == 200)
        {
            /* Extracting the operation (S or M) */
            var operation = (httpreq.responseText).substring(0,1);

            /* Extracting suggestion received from doSql.php */
            var suggestion = (httpreq.responseText).substring(1);

            /* If multiple suggestions are received */
            if(operation == "M")
            {
                /* Populating the DIV with the response received */
                /* */
                document.getElementById('booksSuggested').innerHTML =
                    suggestion;
            }
            /* If single suggestion is received */
            if(operation == "S")
            {
                /* Clearing the first DIV's contents */
                document.getElementById('booksSuggested').innerHTML
                    = "";

                var txtAutoSuggest =
                    document.getElementById('txtAutoSuggest');
                /* If suggestion is received and the textbox
                still holds the original text (i.e. no changes
                made) */
                if ((suggestion) && (txtAutoSuggest.value ==
                    original_text))
                {
                    /* Extracting length of the text keyed in */
                    var initial_len = txtAutoSuggest.value.length;
                    /* Extracting length of suggestion received */
                    /* */
                    var suggestion_len = suggestion.length;
                    /* Calculating the length difference */
                    var difference_len = suggestion_len - initial_len;
                    /* Clearing the text keyed in */
                    txtAutoSuggest.value = "";
                }
            }
        }
    }
}

```


If it holds (which means that the user has keyed in a character in the text box), it further checks if the character keyed in is an alpha numeric character.

```
if ((ev.keyCode >= 48 && ev.keyCode <= 57) || (ev.keyCode >= 65 &&
ev.keyCode <= 90))
{
```

Only if the character keyed in is an alpha numeric character then the following is done:

An object of type XMLHttpRequest is spawned by invoking a user defined method named createRequestObject().

```
var httpreq = createRequestObject();
```

The value received (i.e. alpha numeric character keyed in by the user) is assigned in form of name=value pair to the parms memory variable. This name=value pair will be dispatched along with the URL whilst opening a request

```
var parms = "val=" + sender.value;
```

A copy of the value received (i.e. alpha numeric character keyed in by the user) is created. This is done to maintain the text keyed in by the user, which will allow segregating the original text from the suggested text as shown in diagram 4.4

```
var original_text = sender.value;
```

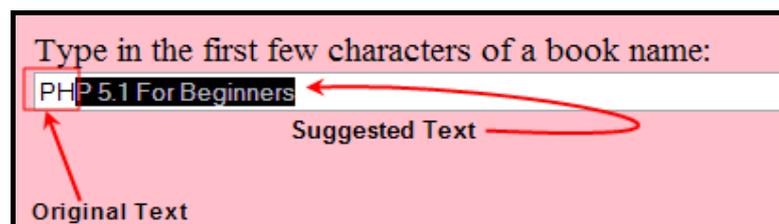


Diagram 4.4: Original / Suggested Text

Finally a request is made using the open() method of the XMLHttpRequest object

```
httpreq.open('GET', 'doSql.php?' + parms, true);
```

Here,

```
doSql.php? + parms
```

is the URL. Along with the URL, the character keyed in is passed.

Before making a request, a listener bound to a JavaScript function is declared, that will handle web server responses

This function will:

1. Check the state of the request

2. If the state (readyState property) holds the value 4

It is a green signal from the web server that the **response** is received **completely**. This allows proceeding further

3. Check the **status code** of the HTTP server response

```
if(httpreq.readyState == 4)
{
    if(httpreq.status == 200)
    {
        ...
    }
}
```

Now that the state of the request (4) and the HTTP status code of the response (200) are known, it's time to work with the data received from the web server (i.e. its response).

The data received from the server-side PHP script is in form of operation.suggestion pair. This means if the suggestion received is a single entry, operation holds 'S' and if the suggestion received are multiple entries, operation holds 'M'. The suggestion at any time will hold the text received whether it is a single suggestion or multiple suggestions.

Since the data received from the server-side PHP script is in form of operation.suggestion pair both need to be separated. This is done using substring. Data received is retrieved using the **httpreq.responseText** property of the **XMLHttpRequest** object spawned earlier.

```
var operation = (httpreq.responseText).substring(0,1);
var suggestion = (httpreq.responseText).substring(1);
```

Once this is done, based on the value held in the operation memory variable following is done:

If operation holds 'M' the first DIV is populated with the suggestion received using the innerHTML property of DIV.

```
if(operation == "M")
{
    document.getElementById('booksSuggested').innerHTML = suggestion;
}
```

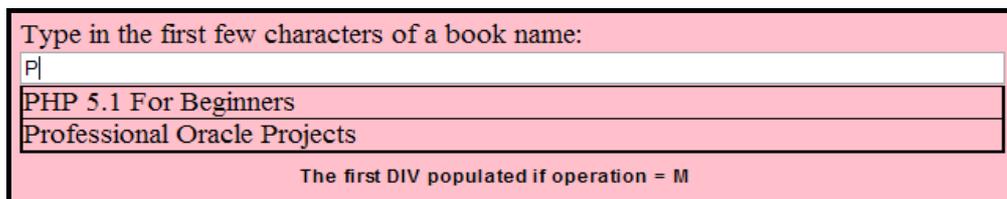


Diagram 4.5: Multiple Suggestions

4. TEXT SUGGEST

If operation holds 'S':

```
if(operation == "S")
{
```

- The content of the first DIV is cleared

```
document.getElementById('booksSuggested').innerHTML = "";
```

- If a suggestion is received and the textbox still holds the original text (i.e. no changes are made) then:

```
if ((suggestion) && (txtAutoSuggest.value == original_text))
{
```

- The length of text keyed in and suggestion received is extracted

```
var initial_len = txtAutoSuggest.value.length;
var suggestion_len = suggestion.length;
```

- Based on their length the difference is calculated

```
var difference_len = suggestion_len - initial_len;
```

- A Selection range is created that will allow selecting only the text that is suggested excluding the original text the user keyed in

```
var sel = document.selection.createRange();
```

- The suggestion received is assigned to the selection range text

```
sel.text = suggestion;
```

- The cursor is moved just after the original keyed in text

```
sel.move("character", -(difference_len));
```

- The text that follows the original keyed in text is searched for using the findText method of the selection range

```
sel.findText(suggestion.substring(original_text.length));
```

- Finally the text just after the original keyed in text is selected (highlighted)

```
sel.select();
```

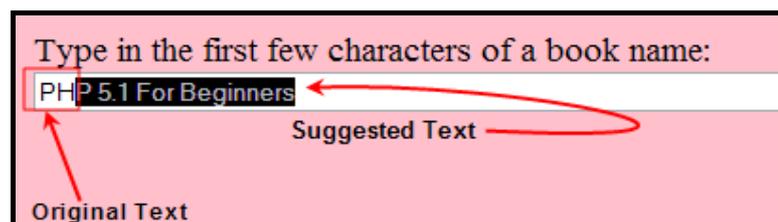


Diagram 4.6: Original / Suggested Text

Once all this is done the text suggested (book name) is now in the text box. This value is passed to a user-defined function named `displayBookInfo()`. This function will populate the book details of the suggested book's name.

```
displayBookInfo(sender.value);
```

If no suggestion was received or if the user changes the text in the text box then the contents of the second DIV is cleared.

```
else
{
    document.getElementById('bookDetailsPanel').innerHTML = "";
}
```

The request is sent to the web server (server-side PHP script) for processing.

```
httpreq.send(null);
```

Code Spec:

```
function setText(bkName)
{
    /* Assigning the bookname received to the textbox */
    document.getElementById('txtAutoSuggest').value = bkName;
    /* Calling a user-defined function to display book details of the
    bookname received */
    displayBookInfo(bkName);
}
```

Explanation:

The `setText()` function is invoked when a book name is clicked from among the multiple suggestions made and populated in the first DIV. This function when invoked populates the textbox with the book name selected from the first DIV. This function also populates the second DIV with that book's details by calling a user-defined function `displayBookInfo()`.

Code Spec:

```
function displayBookInfo(bkName)
{
    if(bkName)
    {
        /* Setting up an XMLHttpRequest */
        var request = createRequestObject();

        /* Opening a request */
        request.open("GET", "doSql.php?operation=ShowDetails&bkName=" +
                    bkName);

        /* Defining an event handler. */
        request.onreadystatechange = function()
        {
```

4. TEXT SUGGEST

```

        if(request.readyState == 4)
        {
            if(request.status == 200)
            {
                /* Holding the response received in a memory variable
                */
                var response = request.responseText;
                /* Populating the DIV with the response received */
                document.getElementById('bookDetailsPanel').innerHTML =
                    response;
            }
        }
    }
    /* Sending the asynchronous request. */
    request.send(null);
}
else
{
    /* Clearing the contents of the bookDetailsPanel DIV */
    document.getElementById('bookDetailsPanel').innerHTML = "";
}
}
}

```

Explanation:

The displayBookInfo() function is invoked to fetch and then display the book details retrieved in a DIV using HTML TABLE tags.

A check is made if the book name is received as a parameter value. As based on the value held in this value the book details will be fetched and displayed.

The createRequestObject() function is invoked to actually spawn an XMLHttpRequest object and return the same into the request variable.

A request is made using the open() method of the XMLHttpRequest object.

Before making a request, a listener bound to a JavaScript function is declared, that will handle web server responses.

This function will:

1. Check the state of the request
2. If the state (readyState property) holds the value 4
 - It is a green signal from the web server that the **response** is received **completely**. This allows proceeding further
3. Check the **status code** of the HTTP server response

```

if(request.readyState == 4)
{
    if(request.status == 200)
    {
        var response = request.responseText;
        document.getElementById('bookDetailsPanel').innerHTML = response;
    }
}

```

Now that the state of the request (4) and the HTTP status code of the response (200) are known, it's time to work with the data received from the web server (i.e. its response).

Data received from the server is processed using the **request.responseText** property of the **XMLHttpRequest** object.

The response received is set to the innerHTML property of the HTML element (in this case a DIV element). This actually generates a combo box with book names populated.

```
request.send(null);
```

The request is now sent to the web server (server-side PHP script) for further processing.

The **open()** method sets the vision for an upcoming operation. Two required parameters are the **HTTP method** intended for the request and the **URL** for the connection.

```
'GET'
```

Here, **GET** is used as the HTTP method since the length of the data being dispatched is not **greater than 512 bytes**.

```
doSql.php?operation=ShowDetails&bkName=" + bkName
```

The **URL** may be either absolute **or** relative URL with the relative URL being the default chosen in web applications. Here, along with the URL, the operation required (in this case ShowDetails) and the book name are passed which indicates that the book details should be fetched using this book name (bkName).

The bookDetailsPanel DIV's previous contents if any are cleared.

The doSql.php File

Code Spec:

```

<?php
$user = "root";
$password = "sct2306";
$db = "bookdb";
$link = mysql_connect(localhost,$user,$password) or die("Could Not Connect To
MySQL Server");
mysql_select_db($db) or die("Could Not Connect To Database");

```

4. TEXT SUGGEST

Explanation:

The control is passed to doSql.php file by the:

- autocomplete() - To suggest the names of the books

```
httpreq.open('GET', 'doSql.php?' + parms, true);
```

- displayBookInfo() - To populate the book details

```
request.open("GET", "doSql.php?operation=ShowDetails&bkName=" + bkName);
```

The above code spec instructs the PHP interpreter to connect to a MySQL Db engine running in memory (i.e. the RAM) of the **localhost** (i.e. localhost indicates that the MySQL Db engine is running on the same computer as the PHP interpreter). Unless the web site hosting company specifies otherwise **always use localhost.**

mysql_connect() is used to establish a connection to the MySQL Db engine running in memory on the local computer **or** on an entirely different computer (simply by passing the appropriate ip as an argument).

After having connected to the MySQL DB engine, running in the computer's memory (i.e. RAM) a MySQL database that exists on the HDD must be selected. This must be a database to which the **username** specified **has access permissions.**

The following does this:

```
@mysql_select_db($db)or die("Could Not Connect To Database");
```

This tells the PHP program to select the database stored in the variable **\$db**, programmatically set earlier.

Code Spec:

```
/* Checking if the request was to populate the book details DIV
*/
if($_GET['operation'] == 'ShowDetails')
{
/* Storing the query in a variable */
$query = "SELECT PublisherName, BookCost, BookSynopsis, BookPic
        FROM BookMaster WHERE BookName=" . $_GET['bkName']. """;

/* Executing the SQL query */
$resultSet = mysql_query($query);
$row = mysql_fetch_assoc($resultSet);

if($row)
{
/* Creating a TABLE to hold the book details. */
```

```

echo '<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"
      WIDTH="100%">
      <TR>
        <TD VALIGN="top" WIDTH="25%"><B>
          Publisher Name:</B></TD>
        <TD VALIGN="top">'. $row["PublisherName"].'</TD>
        <TD ALIGN="RIGHT" VALIGN="top" ROWSPAN="2">
          <IMG BORDER="2" HEIGHT="110" WIDTH="110"
            SRC="'. $row["BookPic"].'"</IMG></TD>
      </TR>
      <TR>
        <TD VALIGN="top" WIDTH="45%">
          <B>Book Cost:</B></TD>
        <TD VALIGN="top">'
          . $row["BookCost"].'</TD>
      </TR>
      <TR>
        <TD COLSPAN="3" VALIGN="top" WIDTH="45%">
          <B>Book Synopsis:</B><BR>'. $row["BookSynopsis"].'
        </TD>
      </TR>
    </TABLE>';
  }
}

```

Explanation:

If the control is passed to the doSql.php file by the displayBookInfo() function (which means the parameter named operation holds the value ShowDetails), then:

- An ANSI SQL SELECT query is formed to retrieve the required table data from the BookMaster table

```

$query = "SELECT PublisherName, BookCost, BookSynopsis, BookPic
        FROM BookMaster WHERE BookName=' . $_GET['bkName']. '";

```

- The query is executed using **mysql_query()**

```

$resultSet = mysql_query($query);

```

- The results returned by the query execution are assigned to the **\$row** variable which is a simple associative array

```

$row = mysql_fetch_assoc($resultSet);

```

- A HTML TABLE is structured to hold the contents of the **\$row** array i.e. table columns retrieved by the query

```

echo '<TABLE BORDER="0" CELLSPACING="0" CELLPADDING="0"
      WIDTH="100%">

```

- Every column extracted and available in the \$row is populated into the table columns, i.e.

```

<TR><TD>$.row['ColumnName']</TD></TR>

```

4. TEXT SUGGEST

- Finally, the TABLE tag is closed to complete the HTML TABLE and the entire HTML TABLE code spec is sent back from the Web server as a response to the code spec of the JavaScript displayBookInfo() function

```
</TABLE>';
```

Code Spec:

```
else
{
    $getBookName_qry = "SELECT BookName FROM BookMaster
                      WHERE BookName LIKE ".$_GET['val']."%";
    $res_getBookName = mysql_query($getBookName_qry);
    if (mysql_num_rows($res_getBookName) > 1)
    {
        $i = 0;
        $books = "<TABLE CELLSPACING=0 CELLPADDING=0 WIDTH='100%'
                STYLE='border: 2px ridge black;
                border-top-style: hidden;'>";
        while($i < mysql_num_rows($res_getBookName))
        {
            $books .= "<TR><TD STYLE='border-top: 1px solid black;'
                    onMouseDown=\"setText(\".
                    mysql_result($res_getBookName, $i,0).\"");\">\"
                    .mysql_result($res_getBookName, $i, 0).
                    \"</TD></TR>\";

            $i++;
        }
        $books .= "</TABLE>";
        echo "M".$books;
    }
    else
    {
        $book = mysql_result($res_getBookName,0,'BookName');
        echo "S".$book;
    }
}
```

Explanation:

If the control is passed to the doSql.php file by the autoComplete() function (which means the parameter named operation does not hold the value ShowDetails), then:

- An ANSI SQL SELECT query is formed to retrieve the required table data from the BookMaster table

```
$getBookName_qry = "SELECT BookName FROM BookMaster
                   WHERE BookName LIKE ".$_GET['val']."%";
```

- The query is executed using **mysql_query()**

```
$res_getBookName = mysql_query($getBookName_qry);
```

- If the query retrieves more than one record then:

```
if (mysql_num_rows($res_getBookName) > 1)
{
```

- An HTML TABLE is structured to hold the multiple book names in form of table columns retrieved by the query

```
$books = "<TABLE CELSPACING=0 CELLPADDING=0 WIDTH='100%'
        STYLE='border: 2px ridge black; border-top-style: hidden;'>";
```

- Using a While loop all the records are retrieved and the book names are populated in table columns of an appropriate table row

```
while($i < mysql_num_rows($res_getBookName))
{
    $books .= "<TR><TD STYLE='border-top: 1px solid black;'
              onMouseDown=\"setText(\".mysql_result($res_getBookName,
              $i,0).\"");\">".mysql_result($res_getBookName, $i, 0)
              .\"</TD></TR>\";
    $i++;
}
```

- Every such column's onMouseDown event is set to invoke the setText() function. This function is passed the bookname retrieved whilst inside the While loop
- Finally, the TABLE tag is closed to complete the HTML TABLE and the entire HTML TABLE code spec is sent back along with the operation 'M' (indicating multiple book names were retrieved) as a response to the code spec of the JavaScript autocomplete() function

```
$books .= "</TABLE>\";
echo "M".$books;
```

- If the query retrieves exactly one record then the bookname retrieved is sent back along with the operation 'S' (indicating a single book name was retrieved) as a response to the code spec of the JavaScript autocomplete() function

```
else
{
    $book = mysql_result($res_getBookName,0,'BookName');
    echo "S".$book;
}
```

Understanding The Functionality

This example uses the following files:

- index.html - The HTML page with one text box
- ajaxfunctions.js - The JavaScript file that performs asynchronous requests and processes responses from the PHP server-side script
- doSql.php - The server-side PHP script that performs database operations

4. TEXT SUGGEST

How Is The Suggestion Made

- When the user keys in a character of the book name, the autocomplete() function is invoked. The autocomplete() function is available in the ajaxfunctions.js

```
<INPUT SIZE="100" TYPE="text" ID="txtAutoSuggest" NAME="txtAutoSuggest"
onKeyUp="autocomplete(this, event);"/>
```

- The autocomplete() function in turn passes the control along with the data to a server-side PHP script named doSql.php
- The doSql.php file performs an appropriate data retrieval operation and returns a response to the listener available in the autocomplete() function of the ajaxfunctions.js
- If the response received is a single entry then:
 - The autocomplete() functions assigns the response received to a text box that will display the entire book's name
- If the response received is multiple entries then:
 - The autocomplete() functions assigns the response received to a DIV that will display a set of multiple book names
- The assignment is done using JavaScript's DOM. Since the DOM is updated, the Browser repaints the section that holds the DIV to reflect the changes made
- All this is done asynchronously and hence the entire web page is prevented from being refreshed

Hands On Exercise

Create an application such that when a user types in character, the application suggests terms that come up as search results at the Web server. The first suggestion must be filled into the textbox as a character is typed while a list of several suggestions must appear as a dropdown list beneath the textbox. The form should allow to:

- Suggest the author names starting with the first letter entered in the text box
- Retrieve the author's details once the suggested author name is selected such as the Image of author, degree, speciality, DOB and author details from the AuthorMaster table on demand

After the page is crafted it will appear as shown in diagram 4.7.

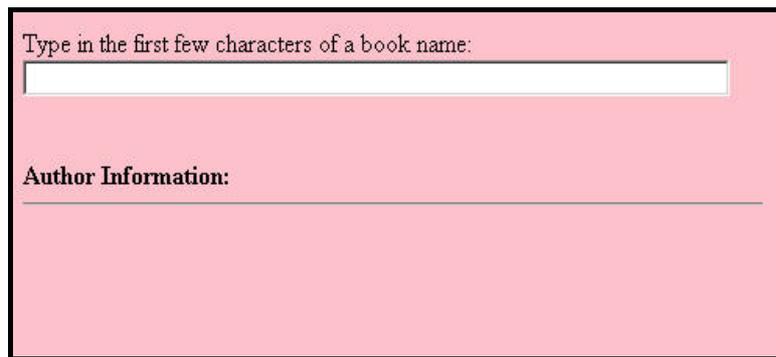


Diagram 4.7: AuthorMaster G.U.I

A suggestion appears in the text box suggestion along with the details of that particular author under **Author Information** if the letter keyed in the text box produces a single suggestion as shown in diagram 4.8.

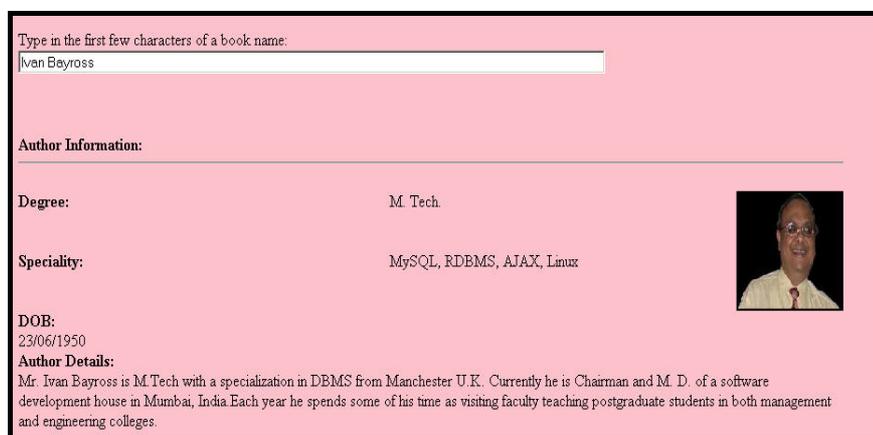


Diagram 4.8: Displaying the information of the suggestion selected

If the letter keyed in the text box produces more than one suggestion then a list of suggestions is displayed below the text. The details of author will be shown under **Author Information:** only if a particular book is selected using left mouse click.

Database Table Specifications

Database Name : bookdb
Table Name : AuthorMaster

4. TEXT SUGGEST

The table structure is as shown below:

Field Name	Data Type	Size	Description
AuthorID	Integer	6	An identity number of the author
Name	Varchar	35	The name of the author
Degree	Varchar	50	The degree of the author
Speciality	Varchar	100	The speciality of the author
DOB	Varchar	15	The date of birth of the author
AuthorPic	Blob		The directory path of the author image
AuthorDetails	Text		The short summary of the author