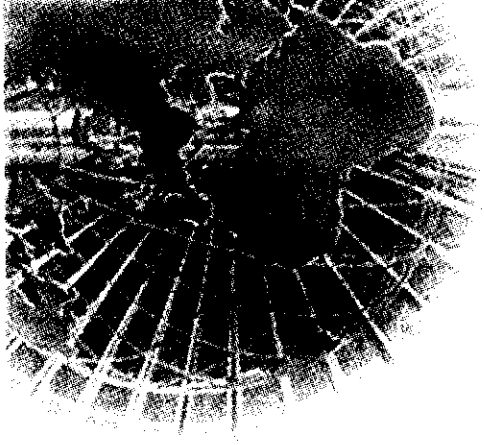


7

Working with Forms and Database

Introduction to Web Forms

In the client/server model of data processing, a user needs to enter relevant data at the client end, which is further submitted to the server for processing. The client/server data processing model is also applicable in cases of Web sites or Web applications; wherein the data to be processed is submitted to the Web server through a graphical user interface (GUI) element, called a Web form. A Web form contains various elements, such as text boxes, radio button, or check boxes, which allow the user to enter the required information. A Web form generally contains a submit button. When you enter data in the Web form and click the Submit button, the content of the form is submitted to the server for further processing.

Working with the <form> Tag and Form Elements

The <form> HTML tag is used to create an HTML form for user input. A form can contain various form elements, such as text boxes, check boxes, radio buttons, and submit buttons. A form can also contain select menus, text area, field set, legend, and label elements. To create a Web form, you first use the <form> tag and then create the form elements. Listing 7.1 shows how to use the <form> tag to create a form. You can find listing 7.1 in \Code\PHP\Chapter 07\Using form tag folder on the CD.

Listing 7.1: Using the <form> Tag

```
<html>
  <head><title>My First Form</title></head>
 <body>
  <form  name=" "  action=" "  method="get/post"  enctype=" ">
    Input elements.
  </form>
 </body
</html>
```

In Listing 7.1, the <form> tag contains four attributes, which are:

- name—Specifies the name of the form
- action—Specifies the address where you want to send the data, such as an e-mail address, a Web server address, or any other form
- method—Specifies the HTTP method used to pass the form's content to the Web server
- enctype—Specifies how the form content should be encrypted

After creating the form, let's now learn to create the following form elements:

- Text box
- Radio button
- Check box
- Drop-down box

Creating a Text Box

You use the <input> tag to create the form elements. To create a text box, you need to specify the type of the <input> tag as text. Listing 7.2 shows how to create a text box. You can find listing 7.2 in \Code\PHP\Chapter 07\Creating Text Box folder on the CD.

Listing 7.2: Creating a Text Box

```
<html>
  <head><title>My First Form</title></head>
 <body>
  <form  name="form1">
    First name:
  <input type="text" name="firstname" />
  < br />
Last name:
<input type="text" name="lastname" />
  </form>
</body> </html>
```

The output of Listing 7.2 is shown in Figure 7.1:

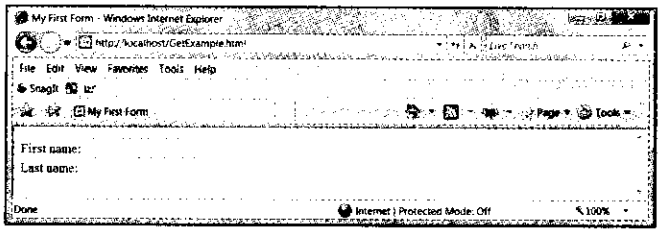


Figure 7.1: Showing two Text Boxes

Let's now learn how to create a radio button.

Creating a Radio Button

Radio buttons are used when we want the user to select one of the listed options. Listing 7.3 shows how to create a radio button. You can find listing 7.3 in \Code\PHP\Chapter 07\Creating Radio button folder on the CD.

Listing 7.3: Creating a Radio Button

```
<html>
  <head><title>My First Form</title></head>
  <body>
        <form  name="form1">
               <input type="radio" name="sex" value="male" />
               Male < br />
               <input type="radio" name="sex" value="female" /> Female
        </form>
  </body
</html>
```

The output of Listing 7.3 is shown in Figure 7.2:

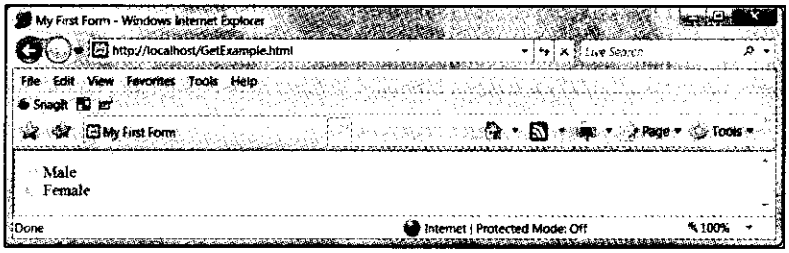


Figure 7.2: Displaying Radio Buttons

Let's learn how to create check boxes next.

Creating a Check Box

Unlike a radio button, a check box allows you to select multiple options at a time. Listing 7.4 shows how to create a check box. You can find listing 7.4 in \Code\PHP\Chapter 07\Creating Check Box folder on the CD.

Listing 7.4: Creating a Check Box

```
<html>
  <head><title>My First Form</title></head>
  <body>
  <form  name="form1">
        I have a jeans:
        <input type="checkbox" name="clothes" value="jeans" /> < br />
        I have a T-Shirt:
        <input type="checkbox" name="clothes" value="T-Shirt" /> < br />
        I have a Shirt:
```

```
        <input type="checkbox" name="clothes" value="Shirt" />
    </form>
    </body
</html>
```

The output of Listing 7.4 is shown in Figure 7.3:

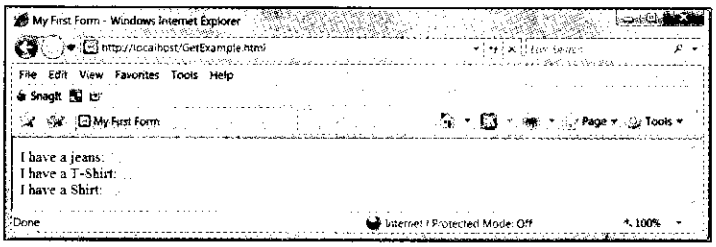


Figure 7.3: Displaying Check boxes

Creating a Drop-Down Box

A drop-down box allows a user to select only one out of a given list of options. It is used in cases where similar kind of data can be grouped under a common heading. The drop-down box occupies much less space as compared to radio buttons for displaying the same amount of information. For example, names of all the 12 months of a year can be grouped only under a common heading, Months in a drop-down box. Listing 7.5 show how to create a drop-down box. You can find listing 7.5 in \Code\PHP\Chapter 07\Drop-down list folder on the CD.

Listing 7.5: Creating a Drop-Down Box

```
<html>
  <head><title>My First Form</title></head>
  <body>
    <form name="Form1">
      <select name="cars">
        <option value="volvo">Volvo</option>
        <option value="saab">Saab</option>
        <option value="fiat">Fiat</option>
        <option value="audi">Audi</option>
      </select>
    </form>
</body>
</html>
```

The output of Listing 7.5 is shown in Figure 7.4:

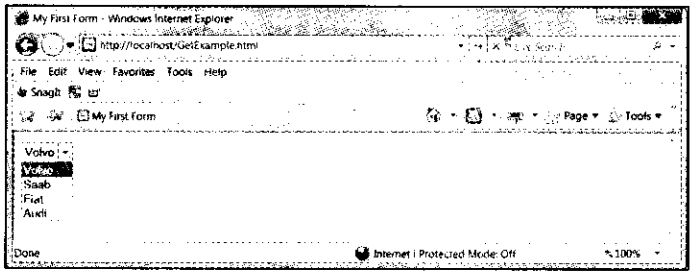


Figure 7.4: Displaying a Drop-Down Box

Processing a Web Form

As you have learned earlier, the information entered in a Web form is sent to a Web server for processing. The processing of a Web form may include the following activities:

- Submitting the form data

- Retrieving the form data
- Validating the form data

Let's learn about these in detail next.

Submitting the Form Data

The data in a Web form can be sent to the server by using either the get() or post() method.

Using the get() Method

After you press the Submit button on a Web form, the get() method sends the form data to the server by attaching it at the end of the Uniform Resource Locator (URL). The attached form data is called query_string. The get() method allows you to send only a limited amount of information at a time. In addition, the information sent by using the get() method is displayed in the address bar of the Web browser.

NOTE

In HTML, the method name must be written in the lower case. However, case restriction is not applicable in case of PHP.

Listing 7.6 shows how to use the get() method to send the form data to the Web server. You can find listing 7.6 in \Code\PHP\Chapter 07\Using get Method folder on the CD.

Listing 7.6: Sending Form Data Using the get() Method

```
<html>
  <head><title>Sending the Form Data with the get() Method</title></head>
  <body>
    <form name="Form1" method="get" action="Welcome.php">
      Enter Your Name:
      <input type="text" name="Name" />
      </br>
      Enter Your Age:
      <input type="text" name="Age" />
      </br>
      <input type="submit" name ="Btn" value="Submit"
    </form>
  </body>
</html>
```

In Listing 7.6, the Form1 form has two text boxes, Name and Age, and a submit button, named Btn. The value of the Btn button is given as Submit, which is displayed as the button caption. The data entered in the Name and Age text boxes is sent to the Welcome.php page using the get() method.

The output of Listing 7.6 is shown in Figure 7.5:

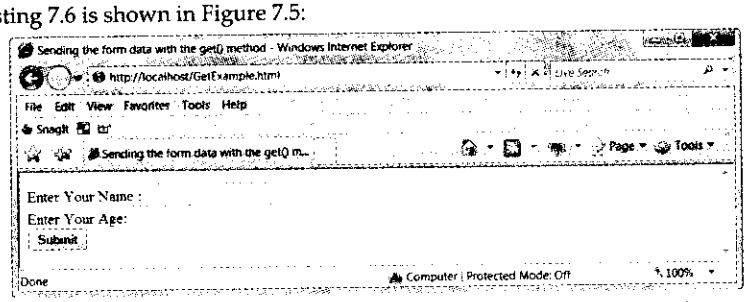


Figure 7.5: Displaying the Form1 Form using the get() method

Now, enter the values in the Name and Age textboxes and click the Submit button. A new Web page appears, displaying the information entered in the previous page, as shown in Figure 7.6:

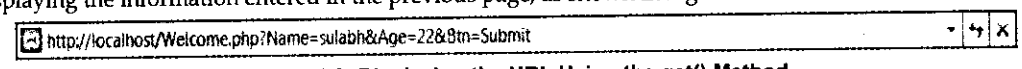


Figure 7.6: Displaying the URL Using the get() Method

Note that the data entered in the Name and Age text boxes appears in the URL of the Web page shown in Figure 7.6.

NOTE

Since the information entered is displayed in the URL, this method should not be used when sending passwords or other sensitive information.

Using the post() Method

Unlike the get() method, the post() method appends all the form data into the Hyper Text Transfer Protocol (HTTP) header message body. You can send any amount of data using the post() method. Information sent through this method is not displayed in the URL of the Web browser. Listing 7.7 shows how to use the post() method to send the form data to the server. You can find listing 7.7 in \Code\PHP\Chapter 07\Using post Method folder on the CD.

Listing 7.7: Sending Form Data Using the post() Method

```
<html>
  <head><title>Sending the Form Data with the post() Method</title></head>
  <body>
    <form name="Form1" method="post" action="welcome.php">
      Enter Your Name:
      <input type="text" name="Name" />
      </br>
      Enter Your Age:
      <input type="text" name="Age" />
      </br>
      <input type="submit" name ="Btn" value="Submit"
    </form>
  </body>
</html>
```

The output of Listing 7.7 is shown in Figure 7.7:

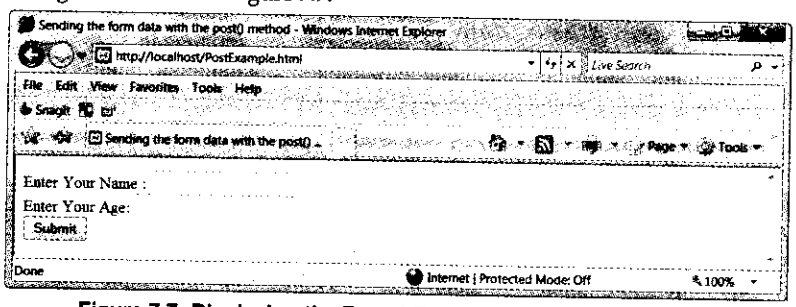


Figure 7.7: Displaying the Form1 Form Using the post() Method

Now, click the Submit button. A new Web page appears, as shown in Figure 7.8:

Figure 7.8 Displaying the URL Using the post() Method

You can see in Figure 7.8 that the information entered by the user is not displayed in the URL of the Web browser.

After learning how to submit the form data using the get() and post() methods, let's now learn how to retrieve the submitted form data.

Retrieving the Form Data

PHP provides various functions to retrieve the submitted form data. The following are some of the most commonly used functions to retrieve data:

- The $_GET[] function
- The $_POST[] function
- The $_REQUEST[] function

In addition, you can use the $_SERVER ['REQUEST_METHOD'] method to determine the method used to send the form data to the server.

Let's discuss each of them in detail.

The $_GET[] Function

The $_GET[] function is a built-in function used to retrieve values from a form sent through the get() method. Note that this function retrieves data sent only through the get() method. In the succeeding example, we create a file named Welcome.php, and use the $_GET[] function to retrieve the data from another file, GetExample.html. The retrieved data is displayed on the Welcome.php page. Listing 7.8 shows how to retrieve the data using the $_GET[] function. . You can find listing 7.8 in \Code\PHP\Chapter 07\Using GET_Function folder on the CD.

Listing 7.8: Retrieving the Form Data Using the $_GET[] Function

```
<html>
<head><title>Receiving the Form Data using $_GET[] Function</title></head>
   <body>
    Welcome <?php echo $_GET["Name"]; ?><br />
    You are <?php echo $_GET["Age"]; ?> years old
   </body>
</html>
```

Listing 7.8 shows the use of the $_GET[] function to retrieve the form data from the GetExample.html file.

The output of Listing 7.8 is shown in Figure 7.9:

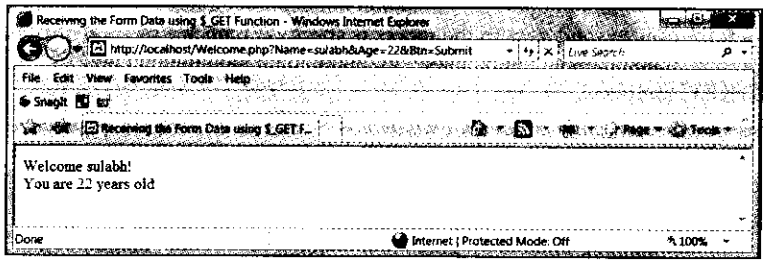


Figure 7.9: Displaying the Welcome.php Page Using the $_GET[ ] Function

The $_POST[] Function

The $_POST[] function is a built-in function used to retrieve the values from a form sent through the post() method. Let's consider the preceding example of the HTML file PostExample.html, which used the post() method to send the data to the Welcome.php file. The Welcome.php file can use the $_POST[] function to retrieve and display the data from the submitted form. Listing 7.9 shows how to use the $_POST[] function to retrieve form data. You can find listing 7.9 in \Code\PHP\Chapter 07\Using POST_Function folder on the CD.

Listing 7.9: Receiving Form Data Using the $_POST[] Function

```
<html>
<head><title>Receiving the Form Data using $_POST[] Function </title></head>
   <body>
    </br>
    Welcome <?php echo $_POST["Name"]; ?><br />
    You are <?php echo $_POST["Age"]; ?> years old
   </body>
</html>
```

The output of Listing 7.9 is shown in Figure 7.10:

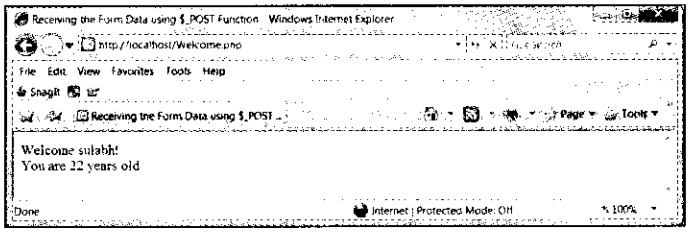


Figure 7.10: Displaying the Welcome.php Page Using the $_POST[] Function

Using the $_REQUEST[ ] Function

The $_REQUEST[ ] function can be used to collect form data sent with both the get() and post() methods. Listing 7.10 shows how to use the $_REQUEST[ ] function to retrieve form data. You can find listing 7.10 in \Code\PHP\Chapter 07\Using REQUEST_Function folder on the CD.

Listing 7.10: Receiving Form Data Using $_REQUEST[ ] Function

```
<html>
<head><title>Receiving the Form Data using $_REQUEST[] Function </title></head>
   <body>
    Welcome <?php echo $_REQUEST["Name"]; ?>!<br />
    You are <?php echo $_REQUEST["Age"]; ?> years old
   </body>
</html>
```

The output of the $_REQUEST[] function depends upon the method used to send data to the sever.

Using the SERVER ['REQUEST_METHOD'] Method

The method that a form uses to send data to the server is specified with the method attribute of the <form> tag. You can find which method has been used to send data to the server by using the $_SERVER ['REQUEST_METHOD'] method.

Listing 7.11 shows how to use the $_SERVER ['REQUEST_METHOD'] method to retrieve the method used to send data to the server. You can find listing 7.11 in \Code\PHP\Chapter 07\Using Server_REQUEST_Method folder on the CD.

Listing 7.11: Using the $_SERVER ['REQUEST_METHOD'] Method

```
<?php
If ($_SERVER['REQUEST_METHOD']=='get')
{
     echo " get() function" ;
}else
{
echo " post() function" ;
}
?>
```

Listing 7.11 uses the $_SERVER ['REQUEST_METHOD'] method to find out which method is used by the client to send the form data to the server. Figure 7.11 shows the output of using the $_SERVER ['REQUEST_METHOD'] method when the data is sent through the get() method:

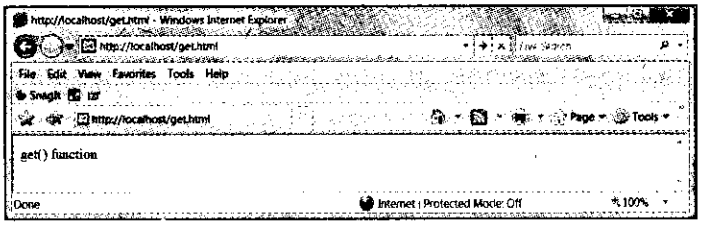


Figure 7.11. Displaying the Method Used to Send the Form Data

Now, let's discuss how to validate the information entered by a user in a form.

Validating a Form

Validating a Web form involves verifying that the users have entered the correct data in the relevant form elements of a form. For example, you can check whether or not a user has correctly filled an email address in the respective field before submitting a form. When using PHP, the data sent in a Web form is verified at the server end. Form validation is the process of checking that a form has been filled in correctly before it is processed. For example, if our form has a box for the user to type their email address, we might want our form handler to check that they've filled in their address before we deal with the rest of the form.

There are two main methods for validating forms: **server-side** (using CGI scripts, PHP, etc), and **client-side** (usually done using JavaScript). Server-side validation is more secure. Figure 7.12 shows the process of validating a Web form:

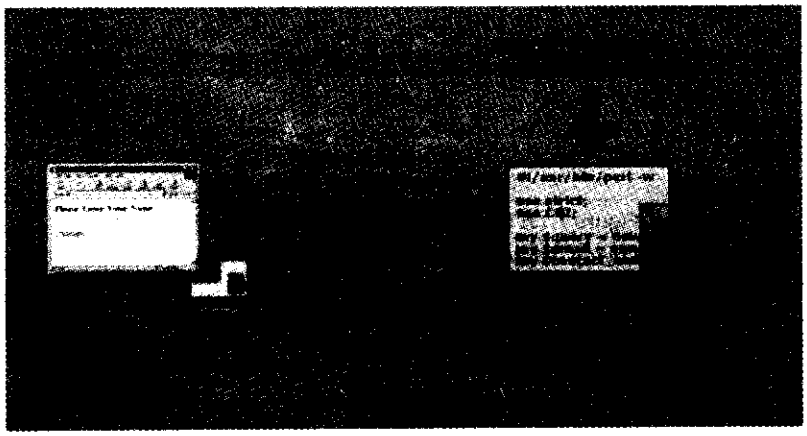


Figure 7.12: Showing Server Side Form Validation

Figure 7.12 shows that the data entered in a form is sent to the server. The data sent is then verified at the server end.

When the server encounters an incorrect data element being sent through a Web form, an error message is displayed. In addition, the server also verifies if the data entered in a form element is in the correct format. For example, a Date text box on a Web form must accept data only in the date format. Let's now explore how we display an error message and enforce data rules to ensure that data is entered in the correct format.

Server side form validation is usually performed with PHP, ASP, or CGI.

Displaying an Error Message when a Field is Empty

The most basic type of form validation is to enforce that a particular field must contain a value. In the case of a text input that is submitted with no value entered, the element in $_POST is still created, but it contains an empty value. A Web form must ensure that a user has entered relevant data in all mandatory form elements. In case a user leaves a mandatory form element blank, an error message can be displayed instructing the user to enter relevant data in the mandatory form elements. You can find listing 7.12 in \Code\PHP\Chapter 07\Web Form with Text box on the CD.

Listing 7.12 shows the content of a Web form containing two text boxes

Listing 7.12: Web Form Containing Two Text Boxes

```
<html>
  <head><title>Form validation using PHP</title></head>
  <body>
    <form method="post" action="form_validate.php">
    Name :
<input type="text" name="userid"></br>
```

```
    Password:
  <input type="password" name="password"></br>
  <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

In Listing 7.12, two text boxes, Name and Password, are created. The data entered by the user is then sent to the form_validate.php file, which verifies for null entries. The output of the preceding listing is shown in Figure 7.13 **Form.php**, we keep two input boxes and we will not allow any user to left them empty. So our form validation script will check for null entry and do the processing accordingly.

Output:

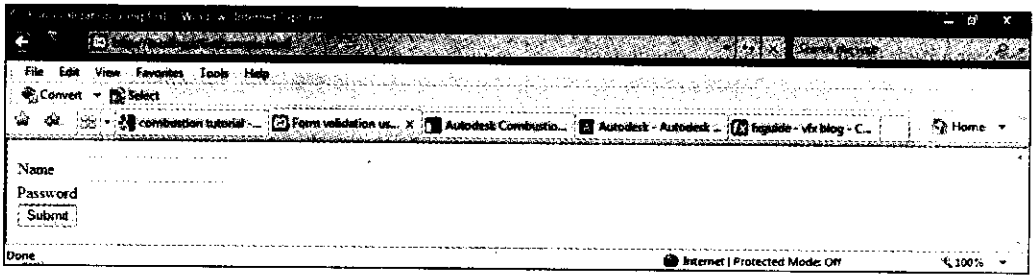


Figure 7.13: Showing the Output of Listing 7.12

Let's posts the data to The data being entered in the preceding form is posted to another Web page and names it as named form_validate.php. We will check and validate the variables in The form_validate.php (Listing 7.13) file validates the entered data for null entries, as shown in Listing 7.13. You can find listing 7.13 in \Code\PHP\Chapter 07\Validating Form Data folder on the CD.

Listing 7.13: Validating Form Data

```
<?php
  $flag="OK";                          // This is the flag and we set it to OK
  $msg="";                           // Initializing the message to hold the error messages
  if(!$_POST['userid'])                    //this check Check for empty Name field
  {
        $msg = ("Please enter user id.")</BR>";
        $flag="NOTOK";                        //setting the flag to error flag.
  }
  if( !$_POST['password'])
  {
        $msg=" Please enter the password ";
        $flag="NOTOK";   //setting the flag to error flag.
  }
  if($flag =="NOTOK")
  {
        // if flag is set to NOTOK, then there is display the back button for the user to
        go navigate back and correct the entries echo "<h1>". $msg."</h1>"."<input
        type='button' value='back' onClick='history.go(-1)'>";
  }
  else
  {
  echo "all entries are correct and let us proceed with the database checking etc ..";
}
```

Now, suppose a user left the Name field empty and clicks the Submit button– (Figure 7.13), the Listing 7.13 validates the entered data and displays an error message, as shown output is shown in the Figure 7.14:

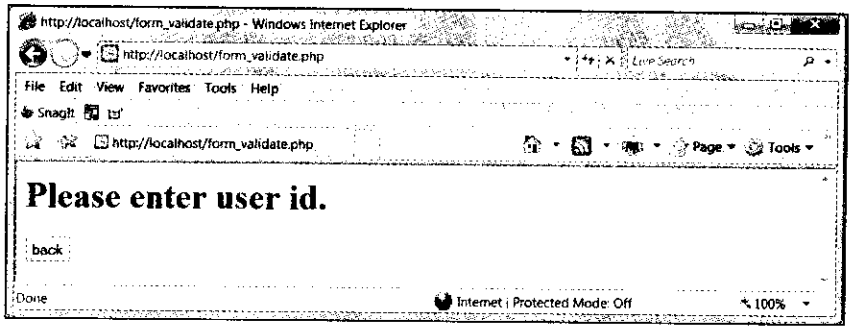


Figure 7.14: Displaying the Error Message

Note that if a user submits a Web form without entering relevant data in a form element, the associated $_POST element is still created; however, it contains an empty value.

Enforcing Data Rules

A Web form may contain specific form elements that accept data in a specific format; for example the Date field. You can verify whether or not the data entered in these specific fields is in the right format by enforcing data rules. Consider a case wherein a Web form contains a text box named Telephone. The Web form would need to enforce data rules to ensure that the data entered in the Telephone text box is in the numerical format. In addition, the Web form must also ensure that the number of digits entered as the telephone number is correct.

We will often want to ensure not only that data is entered into required fields but also that the quality of the data is good enough before proceeding. For instance, we might want to check that an email address or a phone number has the right format or not. In the previous (Listing 7.13) example we have checked only the number of characters entered by the user. Now ILet's now validate a text box so that only characters can be entered in it, as us move one more step and go for form validation checking the type of entry. Say in "userid" field other than characters are not allowed. For this purpose, we use "ereg()" regular expression function of PHP. Now, the modified version of file form_validate.php is shown in Listing 7.14.

You can find listing 7.14 in \Code\PHP\Chapter 07\Enforcing Data Rules folder on the CD.

Listing 7.14: Enforcing Data Rules

```
<?php

$flag="OK";
        $msg="";
        if($_POST['userid'])
        {
                if (ereg('[^A-Z a-z]', $_POST['userid']))
                {
                        $msg="Please use only lower or upper case letters";
                        $flag="NOTOK";
                }
        }
        else
        {
                $msg="Please enter userid";
                $flag="NOTOK";
        }
        if(!$_POST['password'])
        {
                $msg=" Please enter the password ";
                $flag="NOTOK";   //setting the flag to error flag.
        }
        if($flag =="NOTOK")
```

```
        {
                // if flag is set to NOTOK then there is back button for the user to go
                back and correct the entries
                echo "<h1>". $msg."</h1>"."<input type='button' value='back'
                onClick='history.go(-1)'>";
        }
        else
        {
                echo "all entries are correct and let us proceed with the database
                checking etc …";
        }
        ?><?php
        $flag="OK";
        // This is the flag and we set it to OK $msg="";
        // Initializing the message to hold the error message
        if(!$_POST['userid']) //this check for empty Name field
        {
                $msg = ("Please enter user id.")</BR>";
                If (ereg('[^A-Z a-z]', $userid))
                {
                        $msg="Please Use use Only only lower or upper case letters";
                         $flag="NOTOK";       //setting the flag to error flag.
                }
        }
        if( !$_POST['password'])
        {
                $msg=" Please enter the password ";
                $flag="NOTOK";   //setting the flag to error flag.
        }
        if($flag =="NOTOK")
        {
                // if flag is set to NOTOK then there is back button for the user to go
                 back and correct the entries echo "<h1>". $msg."</h1>"."<input
                type='button' value='back' onClick='history.go(-1)'>";
        }
else
{
echo "all entries are correct and let us proceed with the database checking etc …";
}

?>
```

In the preceding listing, the ereg() function is used to validate a field.

Now, if the user inputs enters any characters other than **a-z or A-Z** in "the userid"Name text box, there will be an error an error message is displayed instructing the user to enter only lower or upper case letters. "Please Use Only lower or upper case letters" on the form_validate.php page.

Let's now explore the concept of databases.

Introducing Databases

A database is an organized collection of information that can be easily accessed, managed, and updated. A database consists of logically related data or records stored in computer systems. A database is used in various applications, such as banking, airlines, universities, credit card transactions, telecommunications, sales, finance, and Human Resource (HR).

A database relies on software to store data in an organized manner, and this software is known as Database Management System (DBMS). A DBMS is a collection of programs that allows you to create, maintain, and use databases. DBMS has various advantages over the traditional file systems to manage data, which are as follows:

- Reducing data redundancy and inconsistency.

- Allowing easy access to data.
- Isolating data by providing a common format (table) for data storage.
- Solving the data integrity problems by providing constraints.
- Providing atomicity, specifically for secure online transactions. For example, while making an online transaction, atomicity ensures that either all the tasks of a particular transaction are performed or none is performed. This ensures that if a transaction is incomplete, the data would not be affected by it. In such cases, the data would be updated only if the transaction is complete.
- Providing enhanced security features.

Data in a database can be organized using various database models, such as relational, network, hierarchical, and object oriented models. We explore the relational database model next.

Exploring Relational Database Model, Records, and Primary Keys

Relational database is the primary database model used for storing and manipulating data. A relational database stores data in a tabular format, and might contain one or more tables. Each table in a relational database is assigned a unique name. The tables in a relational database are logically related to each other, which helps in manipulating data across the entire database. The relational database model was introduced by E.F Codd in 1970.

A table in a relational database stores data in the form of rows and columns. Rows in a table are also known as tuples, and columns are also known as attributes. Table 7.1 shows an example of a table named Employee_Information:

Table 7.1: Employee_Information

Emp_ID	Emp_Name	Emp_Address	Emp_PhoneNumber
1001	David	149,Stevenson Rd.Etobicoke	001475689235
1002	William	126,Missisagua Ontario	001564789237
1003	Jeniffer	15,Markham Ave.Toronto	001785349358
1004	Joana	56, Finch Ave. Toronto	001785463298

In the preceding table, there are four rows and four columns. The columns or attributes are named as Emp_Id, Emp_Name, Emp_Address, and Emp_PhoneNumber. Each of these attributes allows only certain specific values to be entered in them. These set of values are known as domain. For Example, the Emp_PhoneNumber attribute can have only numerals and no characters.

The rows in a table can be uniquely identified with the help of a unique attribute, which is known as the primary key. Therefore, the values of any of the attribute must be such that it can uniquely identify a tuple. In the Employee_Information table, the Emp_Id attribute acts as the primary key as it uniquely identifies each tuple in a table. It is important to note that no two tuples in a table can have the same set of values.

Understanding Relationships and Foreign Keys

In the relational database model, two or more tables can be linked to each other by one or more common attributes, called foreign keys. The foreign keys make it possible to create relationships between different tables and combine data from multiple tables to create more comprehensive result sets. While creating a relationship between two or more tables, a primary key of one table can be referred in another table as the foreign key.

Figure 7.15 shows the relationship of three tables:

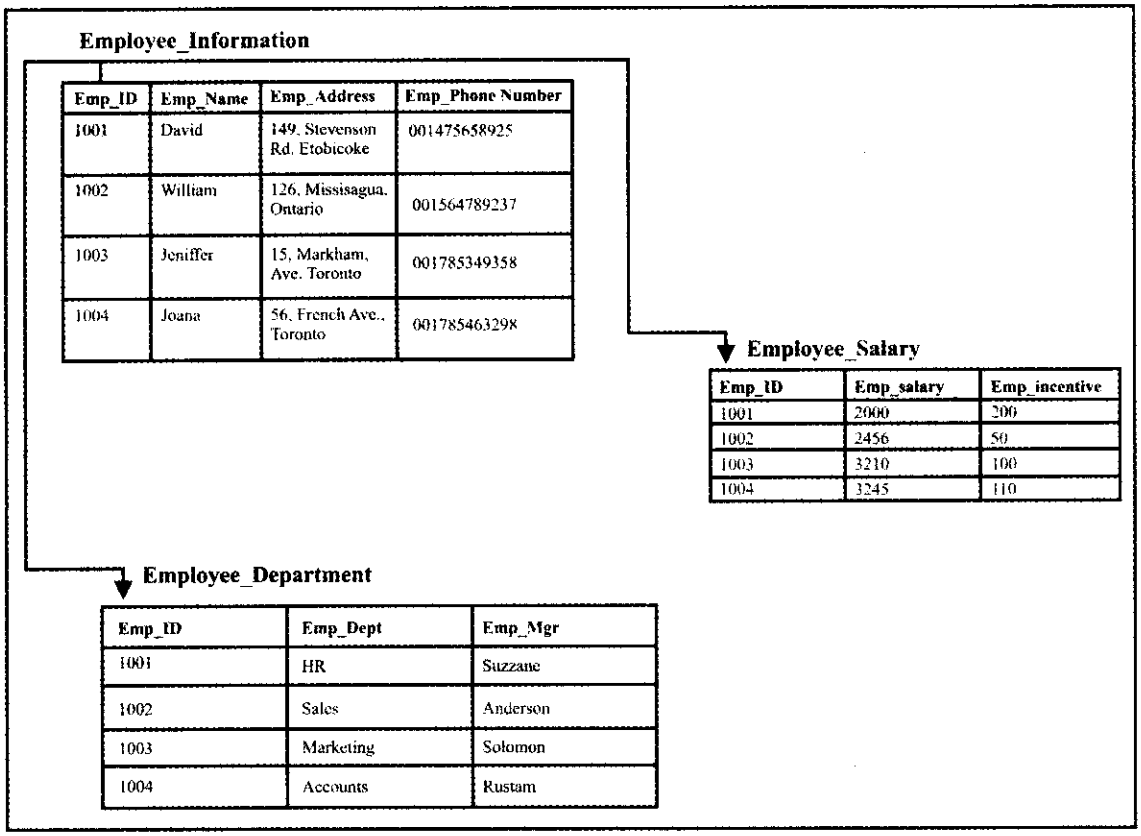


Figure 7.15: Showing the Concept of Foreign Key

The preceding figure shows three tables, Employee_Information, Employee_Department, and Employee_Salary, linked through foreign keys. The Emp_ID attribute acts as the primary key in the Employee_Information table; whereas it acts as the foreign key in the Employee_Department and Employee_Salary tables.

Storing relevant information in independent tables and linking these tables with foreign keys ensures that a particular piece of information appears only once, and redundancy of data is reduced.

Let's now explore Structured Query Language (SQL) statements used to manipulate data in a database.

Working with SQL Statements

SQL is the most widely used query language to manipulate data in a database. SQL statements can be categorized as follows:

- **Data Definition Language (DDL)** – Defines the structure of a database and its tables. These statements are used to create, delete, and modify databases and tables.
- **Data Manipulation Language (DML)** – Allows you to alter and extract data from a database table. These statements are used to add, delete, and retrieve records from a database table, and join tables together using common fields.
- **Data Control Language (DCL)** – Specifies access levels and security privileges for a database. These statements allow you to grant or deny user privileges, assign roles, change passwords, view permissions, and create sets of rules to protect access to data.

Table 7.2 shows syntax of SQL statements:

Table 7.2: Syntax of Commonly Used SQL Statements

SQL Statement	Type of SQL Statement	Action
CREATE DATABASE *database-name*	DDL	Creates a new database
CREATE TABLE *table-name (field1, field2, ...)*	DDL	Creates a new table
INSERT INTO *table-name (field1, field2, ...)* VALUES (*value1, value2, ...*)	DML	Inserts new tuple in an existing table
UPDATE *table-name* SET *field1=value1, field2=value2, ...* [WHERE condition]	DML	Updates the values in a an existing tuple
DELETE FROM *table-name* [WHERE *condition*]	DML	Deletes a particular tuple from a table
SELECT *field1, field2, ...* FROM *table-name* [WHERE condition]	DML	Retrieves the tuples that matches the condition, from the table
RENAME *table-name* TO *new-table-name*	DML	Renames a particular table
DROP TABLE *table-name*	DML	Deletes a particular table from the database
DROP DATABASE *database-name*	DML	Deletes a database

The basic structure of the SQL statement to retrieve data consists of three clauses: **select, from, and where**.

Let's discuss each of these clauses.

The **select** clause

The select clause is used to retrieve data form of a table. For example, the SQL statement to retrieve data from the Employee_Information table is:

```
select Emp_Name from Employee_Information
```

The preceding example retrieves the values of the Emp_Name attribute. If more attributes need to be retrieved, the attribute names are separated by commas.

For example:

```
select Emp_ID,Emp_Name from Employee_Information
```

The preceding example retrieves the values of the Emp_Name and Emp_ID attributes of the Employee_Information table.

If all the attributes need to be retrieved, an asterisk (*) sign is used.

For Example:

```
select * from Employee_Information
```

The preceding example retrieves the values of all the attributes of the Employee_Information table.

The **where** clause

The **where** clause is used with the select clause to retrieve only those records that match a specified condition. The condition to retrieve the records is specified in the where clause by using the assignment operator (=), logical operators, such as **and, or,** and **not,** or comparison operators, such as <,>,<=,>=,= and <>.

An example to retrieve the specific records that matches the where condition is:

```
select Emp_Name from Employee_Information where Emp_ID=1001
```

The preceding example retrieves the values of only those tuples having Emp_ID, 1001.

The **from** clause

The from clause is used to specify the name of the table/tables from where records need to be retrieved.

For Example:

```
select * from Employee_Information
```

The preceding example retrieves records from the Employee_Information table.

Using PHP and MySql

MySql is a Relational Database Management System (RDBMS) used by the world's largest and fastest-growing organizations. The MySQL database is owned, developed, and supported by Sun Microsystems, one of the world's largest contributors to open source software. MySQL was originally founded and developed in Sweden by two Swedes and a Finn: David Axmark, Allan Larsson and Michael "Monty" Widenius.

PHP is used to create dynamic Web pages. The dynamic content does not always require a database; however, in most cases there is need of a database. Web based applications, such as content management systems, blogs, and emails use databases to store information.

The most widely used database in PHP is MySql. To access the MySql database server, you need to check its configuration.

Checking Configuration

While accessing a database you can avoid problems, such as connectivity errors, by checking the configuration of the MySql database server.

You can run the following code to check the configuration of MySql in PHP:

```
<?php
phpinfo();
?>
```

When you run the preceding code on the Web browser, the output appears, as shown in Figure 7.16:

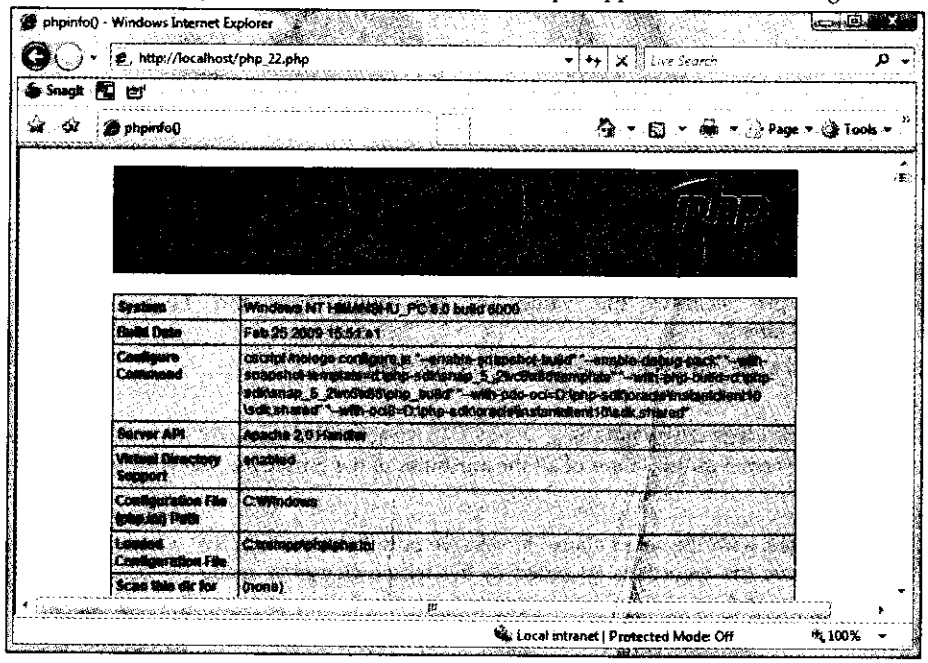


Figure 7.16: Showing the Configuration of the System

When you scroll down the Web page, the configuration of the MySql appears, as shown in Figure 7.17:

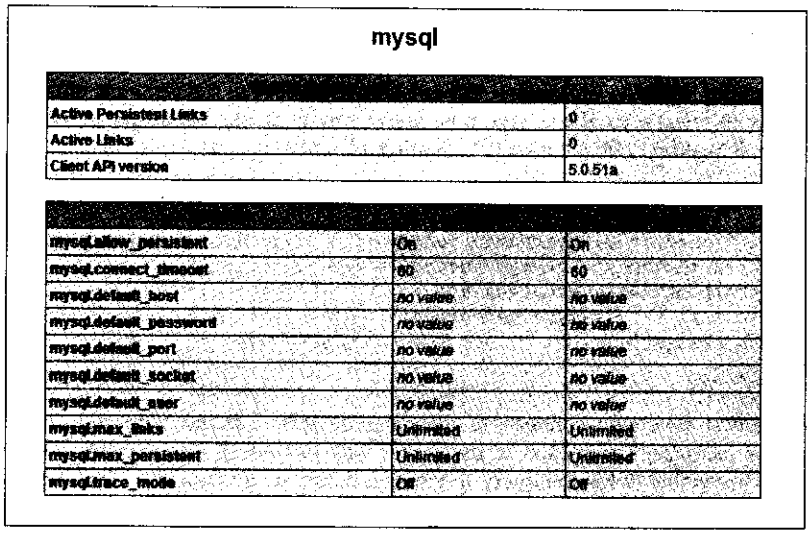


Figure 7.17: Showing the MySql Configuration in PHP

If you do not find the MySql section in the list of configurations, then you cannot access MySql using the current configuration.

Connecting to Database

In PHP, you can use database by establishing a connection to the MySql database server. PHP requires the following information to connect to a database:

- hostname
- database username
- password

Listing 7.15 shows how to establish connection with the MySql database server in PHP. You can find listing 7.15 in \Code\PHP\Chapter 07\Connecting with MySQL folder on the CD

Listing 7.15: Creating a Connection with MySql

```
<?php
mysql_connect("hostname", "username","password") or die(mysql_error());
echo "Connection to the server was successful!";
?>
```

Listing 7.15 displays the text, Connection to server was successful!, if the connection with MySql is established successfully else it displays the corresponding error. The Mysql_error() function is used to display the MySql errors.

NOTE

Suppose, you need to provide database connectivity to 150 Web pages. You can accomplish this task by writing hard code on each Web page; however, this is not appreciated because if you want to change any of the information, such as hostname, username, and password, then you have to change it on every Web page. You can perform this task by creating a single file, which contains the code to connect to the database and you can call this file on whichever page the connectivity is required. You can call this file by using the following code:

```
<?php
include 'filename';
?>
```

Where, filename refers to the name of the file that contains code to connect to the database.

Let's now learn how to select a database.

Selecting a Database

The data is stored in a database from where it can be retrieved. A database server can have many databases, therefore while working with the database it is necessary to select the database on which you want to work. Before learning how to select a database, you must know how to create a database in the database server. You can find listing 7.16 in \Code\PHP\Chapter 07\Creating a Database folder on the CD.

Listing 7.16: Creating a Database

```
<?php
mysql_connect("hostname", "username","password") or die(mysql_error());
echo "Connection to the server was successful!<br/>";
$query  = "CREATE DATABASE mydatabase";
mysql_query($query) or die(mysql_error());
echo "Database created";
mysql_close();
?>
```

In Listing 7.16, the first statement connects to the database server. If the connection is successful, the echo statement prints the message; Connection to the server was successful! else it prints the corresponding error message. The $query variable holds the query to create a database, mydatabase. The mysql_query()function is used to execute the query in MySql database server. The $query variable is passed to the mysql_query()function to execute the query. After the query is executed, the connection to the database must be closed. The mysql_close() function is used to close the connection.

NOTE

You can also pass the query directly in the mysql_query() function as:

```
mysql_query("CREATE DATABASE mydatabase")
```

After creating the database, let's learn how to select a database. MySql database server provides a function mysql_select_db() to select a database. Listing 7.17 shows how to select a database:

Listing 7.17: Selecting a Database

```
<?php
mysql_connect("hostname", "username","password") or die(mysql_error());
echo "Connection to the server was successful!<br/>";
$dbase_name= "mydatabase";
mysql_select_db($dbase_name) or die(mysql_error());
echo "Database is selected";
//code for queries runs here
mysql_close();
?>
```

In Listing 7.17, the mysql_connect() function establishes connection with the database server. Next, the $dbase_name variable is passed to the mysql_select_db() function to select the database. If the database selection is successful, the echo statement prints the message, Database is selected else it prints the corresponding error message. The code to execute a query is placed after selecting the database. Finally, the mysql_close() function is used to close the database connection.

NOTE

The database name can also be directly passed in the mysql_select_db() function as:

```
mysql_select_db("mydatabase")
```

Now, Let's learn how to add and alter tables in MySql.

Adding Table in a Database

As already discussed, the data is stored in the form of tables in a database; therefore, it is necessary to learn how to add a table in a database. Following syntax is used to create a table:

Syntax:

```
CREATE TABLE table_name
```

```
(
attribute_name1 data_type,
attribute _name2 data_type,
attribute _name3 data_type,
....
)
```

Following example shows how to insert data in a table:

```
CREATE TABLE Employee_Information
(
Emp_ID integer NOT NULL,
Emp_Name varchar(40) NOT NULL,
Emp_Address varchar(80) NOT NULL,
Emp_PhoneNumber varchar(15) NOT NULL,
PRIMARY KEY(Emp_ID)
)
```

In the preceding example, the table "Employee_Information" is created. The attributes of the table are Emp_ID, Emp_Name, Emp_Address, Emp_PhoneNumber. The datatype for all the attributes is varchar, the number in the braces represent the maximum number of characters that the attribute can hold. Some other common datatypes are shown in Table 6.3. The NOT NULL clause shows that values must be provided for the corresponding attribute while inserting data in the table. If NOT NULL is not used and value for that attribute is not provided while inserting data, the SQL automatically inserts a NULL value for that attribute.

Listing 7.18 shows how to create a table in a database:

Listing 7.18: Creating a Table

```
<?php
$hostname="hostname";
$user="username";
$password="password";
$database="database";
mysql_connect($hostname,$user,$password);
mysql_select_db($database) or die( "Unable to select database");
$query=" CREATE TABLE Employee_Information
(
Emp_ID integer NOT NULL,
Emp_Name varchar(40) NOT NULL,
Emp_Address varchar(80) NOT NULL,
Emp_PhoneNumber varchar(15) NOT NULL,
PRIMARY KEY(Emp_ID)
) ";
mysql_query($query) or die(mysql_error());
echo "Table successfully created";
mysql_close();
?>
```

In Listing 7.18, first the connection with the database is established using the mysql_connect() function. Next, the corresponding database is selected using the mysql_select_db() function. The query to create a table is stored in the $query variable and executed using the mysql_query() function. If the query is executed successfully, the echo statement prints a message, Table successfully created else it prints the corresponding error message.

Table 7.3 lists various datatypes used in MySql:

Table 7.3: List of Datatypes in MySql

Data Types	Description
CHAR( )	Specifies a fixed range from 0 to 255 characters long
VARCHAR( )	Specifies a variable range from 0 to 255 characters long
TINYTEXT	Specifies a string with a maximum length of 255 characters

Table 7.3: List of Datatypes in MySql

Data Types	Description
TEXT	Specifies a string with a maximum length of 65535 characters
BLOB	Specifies a string with a maximum length of 65535 characters
MEDIUMTEXT	Specifies a string with a maximum length of 16777215 characters
MEDIUMBLOB	Specifies a string with a maximum length of 16777215 characters
LONGTEXT	Specifies a string with a maximum length of 4294967295 characters
LONGBLOB	Specifies a binary string with a maximum length of 4294967295 characters
TINYINT( )	-128 to 127 normal, 0 to 255 UNSIGNED
SMALLINT( )	-32768 to 32767 normal, 0 to 65535 UNSIGNED
MEDIUMINT( )	-8388608 to 8388607 normal, 0 to 16777215 UNSIGNED
INT( )	-2147483648 to 2147483647 normal, 0 to 4294967295 UNSIGNED
BIGINT( )	-9223372036854775808 to 9223372036854775807 normal, 0 to 18446744073709551615 UNSIGNED.
FLOAT	Specifies a small number with a floating decimal point
DOUBLE(,)	Specifies a large number with a floating decimal point
DECIMAL(,)	Specifies a DOUBLE stored as a string, allowing for a fixed decimal point.
DATE	Specifies the date in YYYY-MM-DD format
DATETIME	Specifies the date in YYYY-MM-DD HH:MM:SS format
TIMESTAMP	Specifies the date in YYYMMDDHHMMSS format
TIME	Specifies the date in HH:MM:SS format

Let's learn how to alter a table.

Altering a Table in a Database

The structure of an existing table can be altered using SQL queries. You can drop, add, modify and change the name of a column using the alter table query. Following syntax is used to delete a column of a table:

Dropping a Column(used to delete an entire column):

```
alter table Table_Name drop column Column_Name
```

Following example shows how to drop a column of a table:

```
alter table Employee_Information drop column Emp_PhoneNumber
```

Following syntax is used to add new column in a table:

Adding a Column(used to add new column in table):

```
alter table Table_Name add column Column_Name datatype
```

Following example shows how to insert a new column in a table:

```
alter table Employee_Information add column Emp_PhoneNumber varchar(10)
```

Following syntax is used to change the column of a table:

Changing a Column(used to change the column name):

```
alter table Table_Name change Old_Column_Name New_Column_Name datatype_of_new_column
```

Changing a Column(used to change the column name):

```
alter table Table_Name change Old_Column_Name New_Column_Name datatype_of_new_column
```

Following example shows how to change a column of a table:

```
alter table Employee_Information change Emp_PhoneNumber Emp_Phone_Number varchar(10)
```

Following syntax is used to modify the column of a table:

Modifying a column(used to modify the definition of a column):

```
alter table Table_Name modify Column_Name New_Datatype
```

Following example shows how to modify a column of a table:

```
alter table Employee_Information modify Emp_Phone_Number varchar(20)
```

Consider the table Employee_Information, where the first query executes, the attribute (column) Emp_PhoneNumber is dropped from the table Employee_Information. In the second query, an attribute (column) Emp_PhoneNumber is added in the table Employee_Information. In the third query, the name of the column Emp_PhoneNumber is changed to Emp_Phone_Number. In the fourth query, the column Emp_Phone_Number is modified by expanding its datatype to varchar(20).

Listing 7.19 shows how to alter the structure of a table:

Listing 7.19: Altering the Structure of a Table

```
<?php
mysql_connect("hostname","username","password") or die(mysql_error());
mysql_select_db("database") or die(mysql_error());
$query="alter table Employee_Information change Emp_PhoneNumber Emp_Phone_Number
  varchar(10)";
mysql_query($query) or die(mysql_error());
echo "Column name changed";
mysql_close();
?>
```

In Listing 7.19, first statement establishes a connection with the database server using the mysql_connect() function. Next, the corresponding database is selected using the mysql_select_db() function. The query to alter the table is stored in the $query variable and executed by using mysql_query() function. If the query is executed successfully, the echo statement prints the message, Column name changed else it prints the corresponding error message.

Inserting Data in a Table

Once a table is created in the database, you can insert data in it. Let's learn how to insert data in the table. Following syntax is used to insert data in a table:

Syntax:

```
INSERT INTO table_name ( field1, field2,...fieldN )
                       VALUES
                       ( value1, value2,...valueN )
```

Following example shows how to insert data in a table:

```
INSERT INTO Employee_Information (Emp_ID,Emp_Name,Emp_Address,Emp_PhoneNumber)
                          VALUES(1001,'David','149 Stevenson Rd Etobicoke',0013245678)
```

While providing the values for the attributes, the values for the attributes with text type data types such as VARCHAR(), CHAR() or TINYTEXT should be placed within single quotes. In the previous example as the data type for all the attributes is VARCHAR() hence their values are placed in single quotes. Listing 7.20 shows how to insert data in a table. You can find listing 7.20 in \Code\PHP\Chapter 07\Inserting Data in Table folder on the CD.

Listing 7.20: Inserting Data in a Table

```
<?php
mysql_connect("localhost","root") or die(mysql_error());
mysql_select_db("database");
$query= "INSERT INTO Employee_Information (Emp_ID,Emp_Name,Emp_Address,Emp_PhoneNumber)
VALUES('1001','David','149 Stevenson Rd Etobicoke','0013245678')";
mysql_query($query) or die(mysql_error());
echo "Data successfully inserted";
mysql_close();
?>
```

In Listing 7.20, first statement establishes a connection with the database server. Next, the corresponding database is selected using the mysql_select_db() function. The query to insert data is stored in the $query variable and executed by using the mysql_query() function. If the query is executed successfully, the echo statement prints the message, Data successfully inserted else it prints the corresponding error message.

Modifying Data in a Table

The data stored in a table often needs to be modified to keep the information updated. Following syntax is used to update data in a table:

Syntax:

```
UPDATE table_name
SET column1=value1, column2=value2,...
WHERE condition
```

- **UPDATE**—Specifies that the query is used to update data
- **SET**—Specifies new values to be inserted into the table
- **WHERE**—Specifies the condition that selects the rows to be affected

Following example shows how to update data in a table:

```
UPDATE Employee_Information SET Emp_Name='Jonathan' where Emp_ID=1001
```

In the example, the attribute Emp_Name is updated to 'Jonathan'. for Emp_ID=1001. Listing 7.21 shows how to update data in a table. You can find listing 7.21 in \Code\PHP\Chapter 07\Updating Data in a Table folder on the CD.

Listing 7.21: Updating Data in a Table

```
<?php
mysql_connect("hostname","username", "password") or die(mysql_error());
mysql_select_db("database");
$query= " UPDATE Employee_Information SET Emp_Name='Jonathan' where Emp_ID=1001";
mysql_query($query) or die(mysql_error());
echo "Data successfully inserted";
mysql_close();
?>
```

In Listing 7.21, the first statement establishes a connection with the database server. In the next statement the database is selected using the mysql_select_db() function. The query to update data is stored in the $query variable and executed by using the mysql_query() function. If the query is executed successfully, the echo statement prints the message, Data successfully inserted else it prints the corresponding error message.

Retrieving Data

The content displayed on dynamic web pages is retrieved from a database Following syntax is used to retrieve data:

```
select attribute1,attribute2.....attributeN from table_name where condition
```

In the preceding query, attribute1, attribute2.....attributeN represents the attributes in the table whose values need to be retrieved. The where clause is used to specify certain condition in the select statement, although it is not mandatory.

Following example shows how to retrieve data:

```
select Emp_Name from Employee_Information
select Emp_Name from Employee_Information where Emp_ID=1001
```

The first select statement retrieves the values in the Emp_Name attribute. The second select statement retrieves only those values of the Emp_Name attribute that have Emp_ID=1001. Various types of logical and comparison operators can also be used in the where clause. Listing 7.22 shows how to retrieve data from a table. You can find listing 7.2 in \Code\PHP\Chapter 07\ Retrieving Data from a Table folder on the CD.

Listing 7.22: Retrieving Data from a Table

```
<?php
mysql_connect("hostname", "username","password") or die(mysql_error());
```

```
mysql_select_db("database") or die(mysql_error());
$data = mysql_query("SELECT * FROM Employee_Information")
or die(mysql_error());
print "<table border cellpadding=3>";
while($info = mysql_fetch_array( $data ))
{
print "<tr>";
print "<th>Emp ID:</th> <td>".$info['Emp_ID'] . "</td> ";
print "<th>Name:</th> <td>".$info['Emp_Name'] . " </td></tr>";
print "<th>Address:</th><td>".$info['Emp_Address']."</td><tr>";
print "<th>Phone Number:</th><td>".$info['Emp_Phone_Number']."</td><tr>";
}
print "</table>";
mysql_close();
?>
```

In Listing 7.22, the first statement establishes connection with the database server. In the next statement, the database is selected using the mysql_select_db() function. Next, query to retrieve data is passed in the mysql_query() function and the retrieved data is stored in the $data variable. The data stored in the $data variable is converted in to an array using the mysql_fetch_array() function. The array generated by the mysql_fetch_array() function is stored in the $info variable. The array stored in the $info variable is displayed on the Web page using the while loop. The output is displayed in the HTML format on the Web page (Figure 6.18).

Figure 7.18 shows the data retrieved form a table on the Web page:

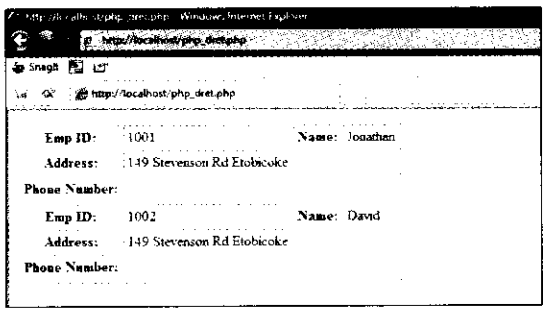


Figure 7.18: Data Retrieved from a table

The mysql_fetch_array() function retrieves a row in an associative array or a numeric array or both. It returns an array that corresponds to the fetched row and returns false, if there are no rows. The mysql_fetch_array() function is an extended version of the mysql_fetch_row() function, which returns a row as a numeric array. It also stores the data in associative indices, using the field names as keys. A disadvantage of the mysql_fetch_array() function over the mysql_fetch_row() function is that it is slower than the mysql_fetch_row() function. Another function, mysql_fetch_assoc() is used to return data only as an associative array.

Let's summarize the key points learned in this chapter.

Summary

In this chapter, you have learned about HTML forms, different HTML tags and form elements used in it. You have learned to process the form in PHP, submitting the form data to PHP using the GET and POST methods. You have learned to validate a form. You have also learned about the relational database, its records, primary and foreign keys, and SQL statements. In the end of the chapter you have learned to use PHP and Mysql.

In the next chapter, we will discuss about cookies, sessions and PHP security.

Quick Revise

Q1. What does HTML stands for?

a. Hyper Text Markup Language

b. Home Tool Markup Language
c. Hyperlinks and Text Markup Language
d. Hyper Text Modern Language

Ans: a

Q2. What is the correct HTML for making a checkbox?
a. <checkbox>
b. <check>
c. <input type="checkbox">
d. <input type ="chek">

Ans: c

Q3. Which function is used to retrieve data submitted using get() method?
a. $_POST[]
b. $_GET[]
c. $_REQUEST[]
d. $GET()

Ans: b

Q4. When using the POST method, variables are displayed in the URL:
a. False
b. True

Ans: a

Q5. Which SQL statement is used to extract data from a database?
a. GET
b. EXTRACT
c. SELECT
d. OPEN

Ans: c

Q6. Which SQL statement is used to insert data into the database?
a. INSERT INTO
b. ADD RECORD
c. ADD NEW
d. INSERT NEW

Ans: a

Q7. What is the correct way to connect to a MySQL_database?
a. Connect_mysql("localhost");
b. dbopen("localhost");
c. mysql_open("localhost");
d. mysql_connect("hostname","username","password");

Ans: d

Q8. In PHP5, MySQL support is enabled by default:
a. False
b. True

Ans: a

Q9 True/False

a. The $_REQUEST[] function can be used to collect form data sent with both the get() and post() methods. **True**

b. The $_POST[] function is a built-in function used to retrieve the values from a form sent through the get() method. **False**

c. SQL stands for Standard Query Language. **False**

d. The Select statement in SQL comes under Data Manipulation Language(DML). **True**

e. The phpinfo() function is used to connect to the MySql database server. **False**

f. Hostname, database name and password of the MySql database server are required to connect to the MySql database server. **True**

g. The mysql_fetch_row() function retrieves data in the form of associative array. **False**

Q10. Explain the <form> tag?

Ans: The <form> HTML tag is used to create an HTML form for user input. A form can contain various form elements, such as text fields, check boxes, radio-buttons, and submit buttons. A form can also contain select menus, text area, field set, legend, and label elements.

To create a Web form, you first use the <form> tag and then create the form elements.

Example:

```
<html>
  <head><title>My First Form</title></head>
   <body>
        <form  name=" "  action=" "  method="get/post"  enctype=" ">
                Input elements.
        </form>
  </body
</html>
```

The <form> tag contains four attributes, which are:

1. **name**—Specifies the name of the form
2. **action**—Specifies the address where you want to send the data, such as an e-mail address, a Web server address, or any other form
3. **method**—Specifies the HTTP method used to pass the form's content to the Web server
4. **enctype**—Specifies how the form content should be encrypted

Q11. Write the HTML code to create the radio buttons and a drop down box?

Ans: Creating a Radio Button:

```
<html>
  <head><title>My First Form</title></head>
        <body>
        <form  name="form1">
         <input type="radio" name="sex" value="male" /> Male
   < br />
   <input type="radio" name="sex" value="female" /> Female
        </form>
  </body
</html>
```

Creating a Drop-Down Box:

```
<html>
  <head><title>My First Form</title></head>
  <body>
        <form name="Form1">
        <select name="Course">
        <option value="MCA">MCA</option>
        <option value="MBA">MBA</option>
```

```
            <option value="BCA">BCA</option>
             <option value="BBA">BBA</option>
            </select>
            </form>
    </body>
    </html>
```

Q12: Which of the following statements is true?

a. **SQL stands for Standard Query Language**

b. **The Select statement in SQL comes under Data Manipulation Language(DML)**

c. **The phpinfo() function is used to connect to the MySql database server.**

d. **Hostname, database name and password of the MySql database server are required to connect to the MySql database server.**

e. **The mysql_fetch_row() function retrieves data in the form of associative array.]**

Ans: b, d

Q13. What are the various categories of SQL statements?

Ans: SQL statements can be categorized as follows:

- **Data Definition Language (DDL)**—Defines the structure of a database and its tables. These statements are used to create, delete, and modify databases and tables.
- **Data Manipulation Language (DML)**—Allows you to alter and extract data from a database table. These statements are used to add, delete, and retrieve records from a database table, and join tables together using common fields.
- **Data Control Language (DCL)**—Specifies access levels and security privileges for a database. These statements allow you to grant or deny user privileges, assign roles, change passwords, view permissions, and create sets of rules to protect access to data.

Q14. What are the advantages of Database Management System (DBMS) over the traditional file system of storing data?

Ans: DBMS has various advantages over the traditional file systems to manage data, which are as follows:

- Reducing data redundancy and inconsistency.
- Allowing easy access to data.
- Isolating data by providing a common format (table) for data storage.
- Solving the data integrity problems by providing constraints.
- Providing atomicity, specifically for secure online transactions. For example, while making an online transaction, atomicity ensures that either all the tasks of a particular transaction are performed or none is performed. This ensures that if a transaction is incomplete, the data would not be affected by it. In such cases, the data would be updated only if the transaction is complete.
- Providing enhanced security features.

Q15. Explain the various queries to alter the structure of a table

Ans: The queries to alter the structure of a table is as follows:

Dropping a Column(used to delete an entire column):

```
alter table Table_Name drop column Column_Name
```

Adding a Column(used to add new column in table):

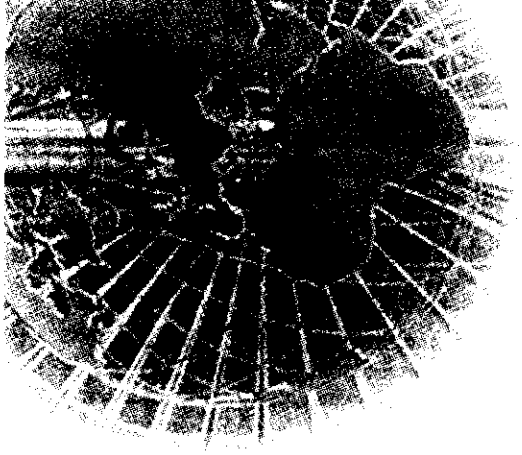
```
alter table Table_Name add column Column_Name datatype
```

Changing a Column(used to change the column name):

```
alter table Table_Name change Old_Column_Name New_Column_Name datatype_of_new_column
```

Modifying a column(used to modify the definition of a column):

```
alter table Table_Name modify Column_Name New_Datatype
```



8

Exploring Cookies, Sessions, and PHP Security

A cookie is a small file containing information that the server embeds on the user's computer. Each time the same computer requests a Web page from a browser, it sends the cookie with the requested Web page. In PHP, you can create and retrieve cookie values. A session is used to store information that is used across multiple Web pages. Sessions stores information on a server whereas cookie stores information on a Web site. However, cookies and sessions are vital keys for any hacker to steal confidential information of the user. In PHP, the process of encryption and decryption is used to safeguard the data. Encryption and decryption of the data is done with the help of special keys. In absence of encryption and decryption keys, user cannot retrieve the data.

In this chapter, you learn about cookies in which you learn to create, read, and remove cookies. You also learn about sessions, in which you learn to start a session, add data to a session, read session data, remove data from a session, and end a session. In addition, you learn how to prevent a session from hijacking. Next, you learn about the differences between cookies and sessions. In the end, you learn how to protect your data from unauthorized users and configure PHP security.

Let's start by learning about cookies.

Working with Cookies

Cookies are used to store data in the remote browser and help to track or identify data returned to users on the Web browser.A cookie is a small bit of information stored on a computer of a user by request from a Web page. This information is constantly passed between the Web browser and Web server, the browser sends the current cookie as a request to the server and the server sends updated data back to the user as response. The size of a cookie depends on a browser but in general should not exceed 1K (1,024 bytes). One common use of cookies is that it stores your username and password on your computer so that you do not need to enter your credentials each time you visit a website.

Exploring Cookie Attributes

A cookie has several attributes which are used as parameters with the setcookie () function. The attributes contains vital information about the cookie, such as its name, domain name from where it belongs to, and address of valid path within a domain. Some commonly used attributes of a cookie are as follows:

- **Name-Value**—Represents the variable name and corresponding value to be stored in the cookie.
- **Expiration date**—Determines the time when to delete a cookie. It is expressed as a Unix timestamp.
- **Valid domain**—Represents a domain name (partial or complete) to which the cookie is sent. For example, if the value for the valid domain attribute is www.gmail.com, the client sends the cookie information to the Web browser every time the client visits the www. gmail.com.
- **Valid path**—Identifies sites within various paths in the same domain. Setting this to the server root (/) allows the entire domain to access the information stored in the cookie.
- **Security Flag**—Restricts a browser from sending cookie information over unsecured connections. It allows the cookie to be sent over any type of HTTP connection. The default value of the security Flag attribute is 0. It may be set to 1 which permits the cookie to be sent over a secure HTTP connection.

After exploring attributes of cookies, let's now learn about cookie header.

Using Cookie Header

Cookies are transmitted between the user's browser and a remote Web site by using the HTTP header. For example, to set a cookie, a Web site must send the Set-Cookie header, containing the necessary attributes, to the user's browser. The following example illustrates the header sent to create two cookies for a domain:

```
Set-Cookie: username=yash; path=/;domain=.thiswebsite.com;
expires = Friday,23-mar-09 12:32:45 IST
set-Cookie: location=INDIA; path=/; domain=.thiswebsite.com;
expires = Friday, 23-mar-09 12:32:45 IST
```

Similarly, if a particular cookie is valid for a Web site, the user's browser automatically includes the cookie information in a Cookie header when requesting the URL of the Web site. In the preceding example, the browser automatically includes the following header in the request:

```
Cookie: username=yash; location=india
```

Let's now learn how to create cookies.

Creating Cookies

You can create cookies in PHP using the setcookie() function. The setcookie() function takes numerous arguments, such as name value, expiration date, and domain name. Listing 8.1 shows how to create cookies. You can find listing 8.1 in \Code\PHP\Chapter 08\Creating Cookie folder on the CD.

Listing 8.1: Creating a Cookie

```
<?php
setcookie('usrname', 'yash', time() + 3600);
echo 'Cookies has been set;
?>
```

The preceding listing creates a cookie on the computer system of a user who loads the page containing the name value pair userName= yash. The cookie expires in 3600 seconds after it has been created. You can also create multiple cookies by using the setcookie() function.

Reading Cookies

When a user visits a PHP page that can read a cookie, which is present in the user's computer at the time they call for the page, PHP automatically reads the cookie into a variable named the same as the cookie, but prefixed with a $ sign. Therefore, to read a cookie, we simply reference its variable name, as shown in Listing 8.2. You can find listing 8.2 in \Code\PHP\Chapter 08\Reading Cookie folder on the CD.

Listing 8.2: Reading a Cookie

```
<?php
echo 'Reading cookie<br>';
echo 'userName = ' . $_COOKIE['userName'];
?>
```

The output of the preceding listing is as follows:

```
Reading cookie
userName = yash
```

Removing Cookies

Cookies are removed by default when the user closes the Web browser. You can override this default setting by setting a time for the cookie to expire. Listing 8.3 shows how to remove a cookie.

Listing 8.3: Removing a Cookie

```
<?php
setcookie("cookiename", "This text will be in the cookie",  time( ) + 3600);
?>
```

In the preceding listing, the expiration time of the cookie is specified by the time( ) parameter followed by the number of seconds. The number of seconds must be prefixed with the plus sign. In this case, the cookie expires in one hour after its creation.

There may be occasions when you need to remove a cookie before the user closes the Web browser and before the specified expiration time of the cookie. In such a case, we use the setcookie( ) function with the appropriate name of the cookie, along with the time( ) parameter and the number of seconds. However, in such cases, we use a negative sign in place of the addition symbol before the number of seconds specified as the cookie expiration time. Listing 8.4 shows how to remove a cookie. You can find listing 8.4 in \Code\PHP\Chapter 08\Removing Cookie folder on the CD.

Listing 8.4: Removing a Cookie Before the User Closes the Web Browser

```
<?php
setcookie ("cookiename", "", time( ) - 1);
?>
```

In Listing 8.4, the cookie, cookiename, expires before the user closes the Web browser.

Storing Arrays in Cookies

You can store an array in a cookie. However, you need to store each element of the array separately within the cookie, as shown in Listing 8.5. You can find listing 8.5 in \Code\PHP\Chapter 08\Storing Array in Cookie folder on the CD.

Listing 8.5: Storing Array in a Cookie

```
<?php
$address =  $_SERVER['REMOTE_ADDR'];
$browser = $_SERVER['HTTP_USER_AGENT'];
$os = $_SERVER['OS'];
setcookie ( "cookie[0]", "$address", time()+200);
setcookie ( "cookie[1]", "$browser", time()+200);
setcookie ( "cookie[2]", "$os", time()+200);
?>
```

Although the array elements are stored separately, when you read the array it will be returned as an array which can be access as follows:

```
$list= $_COOKIE["cookie"];
```

Listing 8.6 shows how to read an array element as an array. You can find listing 8.6 in \Code\PHP\Chapter 08\Reading Array Elements folder on the CD.

Listing 8.6: Reading the Array Element as an Array

```
<?php
$address =  $_SERVER['REMOTE_ADDR'];
$browser = $_SERVER['HTTP_USER_AGENT'];
$os = $_SERVER['OS'];
setcookie ( "cookie[0]", "$address", time()+200);
setcookie ( "cookie[1]", "$browser", time()+200);
setcookie ( "cookie[2]", "$os", time()+200);
?>
<html>
<body>
<?php
$list = $_COOKIE["cookie"];
echo "IP Address: $list[0]";
echo "<br>";
echo "Client Browser: $list[1]";
echo "<br>";
echo "Operating System: $list[2]";
?>
</body>
</html>
```

The output of the preceding listing is as follows:

- **IP Address**—127.0.0.1
- **Client Browser**— Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .NET CLR 3.5.21022; .NET CLR 3.0.30618)
- **Operating System**—Windows VISTA™ Ultimate

After learning about cookies, let's now learn about sessions.

Working with Sessions

A session stores different information for each user accessing a Web site. A session includes server-side and client-side cookies; where the client-side cookie contains a reference to the requested data on the server. Therefore, when a user visits a Web site, the Web browser sends the reference code to the server, which loads the requested data. A session creates a file in a temporary directory on the server, where registered session variables and their values are stored. This data is available to all the Web pages of the Web site when the user is accessing the specified Web site. Session data can be made persistent by storing it to a persistent cookie, or in a MySQL database, if desired.

Sessions are safer to use than cookies because a user can block the cookies from being written, but cannot block the sessions. Moreover, the session information is lost when the user exits the Web site.

Starting a Session

A PHP session is easily started by making a call to the session_start() function. This function first checks whether the specified session has already been started. The function starts a new session if it does not find any open session. While creating a Web page, it is recommended to include the call to the session_start() function at the beginning of the Web page. Session variables are stored in an associative array called $_SESSION[ ]. These variables can be accessed during the entire lifetime of a session. Listing 8.7 shows how to create a session. You can find listing 8.7 in \Code\PHP\Chapter 08\Creating Session folder on the CD.

Listing 8.7: Creating a Session

```
<?php
session_start();
if( isset( $_SESSION['counter'] ) )
{
$_SESSION['counter'] += 1;
}
else
{
$_SESSION['counter'] = 1;
}
$msg = "You have visited this page ".  $_SESSION['counter']."in this  session.";
?>
<html>
<head>
<title>Setting up a PHP session</title>
</head>
<body>
<?php
echo ( $msg );
?>
</body>
</html>
```

The preceding listing starts a session, and then registers a variable called counter, which is incremented each time the Web page is visited during the session. The isset() function checks whether or not the session variable is already set.

The output of the preceding listing is as follows:

```
You have visited this page 6 times in this session.
```

Adding Session Data

Session data is stored in the $_SESSION session global array, which means that each session variable, along with its value, is one element in that array. You can add variables to this array similar to adding variables to any array. Session variables are retained in the session super global array, irrespective of the Web page that a user is currently accessing.

Before you can add any variable to a session, you must make a call to the session_start() function.

The syntax to set a session variable is:

```
$_SESSION['FIRSTNAME']="Rahul";
$_SESSION['LASTNAME']="Sharma";
```

NOTE

You cannot store resources such as database connections in sessions, because these resources are unique to each PHP script, and are usually cleaned when that script is terminated.

After learning the process of adding session data, let's now learn about reading session data.

Reading Session Data

After you have saved the session data, it becomes available in the $_SESSION super global array along with the key of the variable name you gave it. You can read the session data as shown in Listing 8.8:

Listing 8.8: Setting and Reading Session Data

```
<?php
session_start();
$_SESSION['firstname']="Rahul";
$_SESSION['lastname']="Sharma";
$name = "My name is  ".  $_SESSION['firstname']." ". $_SESSION['lastname']. ".";
?>
<html>
<body>
<?php
 echo ( $name );
?>
</body>
</html>
```

The output of the preceding listing is as follows:

```
My name is Rahul Sharma.
```

Removing Session Data

You can remove a specific value from a session by using the `unset()` function. Note that you must use the unset() function only for specific elements of the $_SESSION array, not the $_SESSION array itself; because you will not be able to manipulate the session data after the $_SESSION array is removed. Listing 8.9 shows how to remove a specific value from a session using the `unset()` function. You can find listing 8.9 in \Code\PHP\Chapter 08\Using unset Function folder on the CD.

Listing 8.9: Using the `unset()` Function

```
<?php
session_start();
$_SESSION['firstname']="Rahul";
$_SESSION['lastname']="Sharma";
unset($_SESSION['lastname']); //Removing Session Data.
$name = "My name is  ".  $_SESSION['firstname']." ". $_SESSION['lastname'].".";
?>
<html>
<body>
<?php
 echo ( $name );
?>
</body>
</html>
```

The output of the preceding listing is as follows:

```
My name is Rahul.
```

Note that the preceding listing has removed the value Sharma, as used in Listing 8.8.

Ending a Session

If you want to explicitly end a user's session and delete the session data without closing the Web browser, you need to clear the $_SESSION array, and then call the `session_destroy()` function. The `session_destroy()` function removes all session data stored on the hard disk. Listing 8.10 shows how to end a session:

Listing 8.10: Ending a Session

```
html>
<body>
<?php
session_start();
```

```
if (isset($_SESSION['username']))
{
echo "User : ".$_SESSION['username'];
session_destroy();
}
else {
echo "Set the username";      $_SESSION['username'] = "Rahul";
}
?>
</body>
</html>
```

The output of the preceding listing is as follows:

```
Set the username
```

Let's now learn how to prevent session hijacking.

Preventing Session Hijacking

One of the fundamental principles of Web application security is not to trust data from any unknown or unauthorized client. Therefore, the client must identify itself by sending a unique identifier. The issue of identifying a client as an authorized client creates a fundamental conflict among developers wanting to build secure, stateful applications. In fact, the session mechanism in any Web application is the most vulnerable feature of that application, and session security is one of the most complex topics of Web application security on any platform.

There are numerous types of session-based attacks. Many of these fit into a category called impersonation (session hijacking), where a malicious user attempts to access another user's session by posing as that user. At the very least, these types of attacks require that the malicious user obtain a valid session identifier, because this is the minimum amount of information that must be used for identification. There are at least three ways that a valid session identifier can be obtained by an attacker:

- **Session Prediction**—Refers to guessing a valid session identifier. This guess can range from a wild guess to an educated one, depending upon the sophistication of the attack being used. With PHP's native session mechanism, valid session identifiers are extremely difficult to predict, so this is unlikely to be the weakest point in your implementation.
- **Session Capturing**—Specifies a valid session identifier, which is much more common and there are numerous types of attacks using this approach. When a cookie stores the session identifier, the Web browser vulnerability might be exploited to obtain the session identifier. When a URL variable is used, the session identifier is exposed, and there are many more potential methods to capture a session. For this reason, cookies are generally considered to be more secure than URL variables.
- **Session Fixation**—Specifies a method that tricks a user to select a session identifier already chosen by the attacker. Session fixation attacks attempt to exploit the vulnerability of a system which allows one user to set session identifier for another user. Most session fixation attacks are Web based, and rely on session identifiers, which accepts URLs (query string) or POST data. Listing 8.11 shows how to check session hijacking. You can find listing 8.11 in \Code\PHP\Chapter 08\Checking Session Hijacking folder on the CD.

Listing 8.11: Checking Session Hijacking

```
<?php
session_start();
$user_check=md5($_SERVER['HTTP_USER_AGENT'].$_SERVER['REMOTEADDR']);
if(empty($_SESSION['user_data'])) {
session_regenerate_id();
echo "New session, saving user_check.";
$_SESSION['user_data'] = $user_check;
}
if(strcmp($_SESSION['user_data'],$user_check)!==0)
{
session_regenerate_id();
echo "Warning, you must reenter your session.";
```

```
$_SESSION=array();
$_SESSION['user_data']=$user_check;
}
else
{
echo "Connection Verified!";
}
?>
```

When a browser first requests the page, a session starts. In the session, the encoded IP address and browser type is stored. When the user returns to this page to the server, server compares the value stored in the session with IP address and the browser type received from the user. If the both value do not match then there are chances of breach in the security and attack from hacker. Therefore, a new ID is created and previously saved data is removed from the session and the hacker cannot retrieve any information stored in the session. This does not cause a problem for valid users because they do not change the browser and the IP address in the middle of a session with your Web site.

When you execute the script for the first time, the browser displays the output as shown in Figure 8.1:

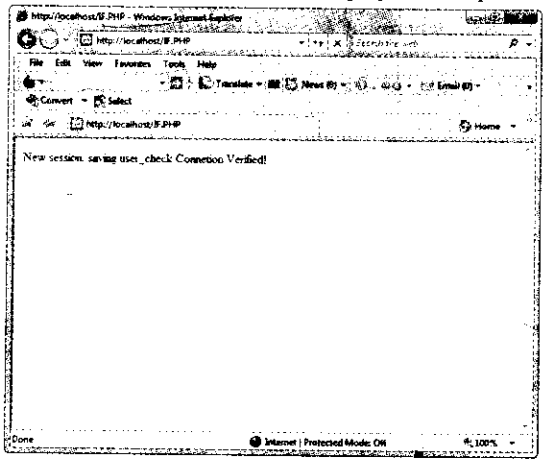


Figure 8.1: Displaying Information about a New Session

When you refreshing the web page, the connection with the server is verified and user gets a confirmation message that connection is authenticated, as shown in Figure 8.2:

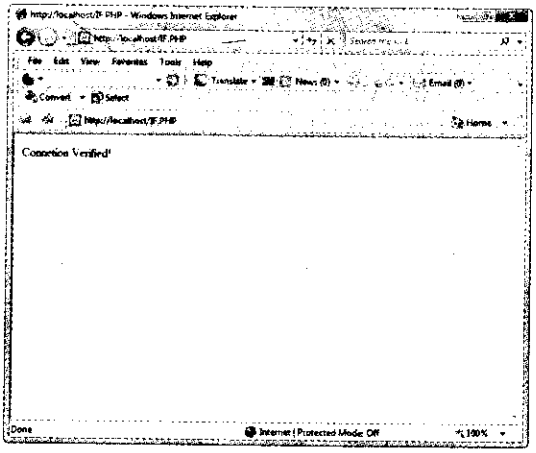


Figure 8.2: Displaying Connection Verified

In case the IP Address or type of IP Address is changed then a warning message appears. In comparison to Listing 8.11, the IP Address is changed from REMOTEADDR to REMOTE_ADDR, as shown in Listing 8.12. You can find listing 8.12 in \Code\PHP\Chapter 08\Changing Browser Type folder on the CD.

Listing 8.12: Changing Browser Type

```
<?php
session_start();
$user_check=md5($_SERVER['HTTP_USER_AGENT'].$_SERVER['REMOTE_ADDR']);
if(empty($_SESSION['user_data']))
{
session_regenerate_id();
echo "New session, saving user_check.";
$_SESSION['user_data'] = $user_check;
}
if(strcmp($_SESSION['user_data'],$user_check)!==0)
{
session_regenerate_id();
echo "Warning, you must reenter your session.";
$_SESSION=array();
$_SESSION['user_data']=$user_check;
}
else {
echo "Connection verified!";
}
?>
```

Due to changes carried out in Listing 8.12 (change IP Address), a warning message appears on the screen, as shown in Figure 8.3:

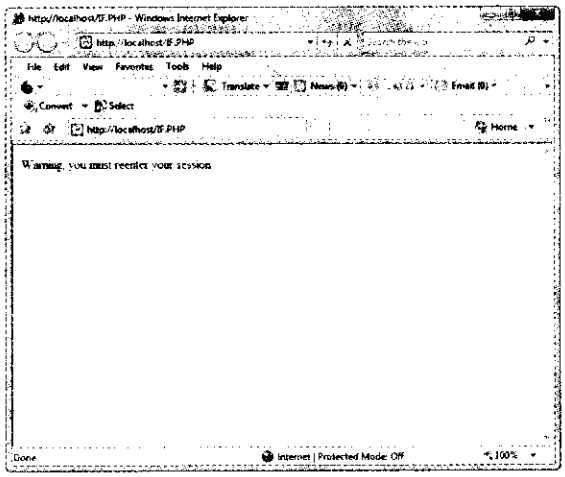


Figure 8.3: Displaying a Warning Message

Let's now discuss some common practices to secure a session.

Do not Trust Unknown or Unauthorized Users

Common users are not aware from security related issues and as a result they easily become victim of the Cross Site Request Forgery (CSRF) attack. If you have any doubt on a user's action then you must ask for credentials.

Shared Hosting by Using session_save_path()

The session_save_path() function allows you to set a new path to store data in a session. You may use *.htaccess* file to allow limited access to the users to access the session data.

Do not Pass Session Identifier in URLs

The session identifier is key for any hacker to gain unauthorized access to your information and session data. Therefore, it is quite important to secure the session identifier present in the URL. PHP offers the $_GET method which hides the session identifier in the URL and does not allow hacker to misuse it.

After discussing some common practices to secure a session, let's now discuss the difference between cookies and sessions.

Differentiating Cookies and Sessions

Both cookies and sessions accomplish much the same task of storing data across Web pages on your Web site. However, there are few differences between the working of cookie and session. Depending upon your requirement, you can choose any one or combination of both.

Cookies can be set to a longer lifespan, which means data can be stored in cookies for months. Once the data is stored in a cookie, it works smoothly when you have a cluster of web servers (a group of independent web servers). Sessions are stored on the server, which means clients do not have access to the information you store in the session. At the time of starting a session, user needs to provide a unique ID for authentication and after successful authentication data is loaded on the Web page from the Web server. Sessions are important when you do not want to share classified information with other users. Also, session can store any amount of data whereas cookie can store only limited amount of data.

Let's now learn how to protect data from unauthorized users.

Protecting Data

The Internet is the most unsafe computing environment in existence. There are many users waiting on the Internet to gain unauthorized access to your information and misuse it. Therefore, practicing encryption, both for data you store locally and you exchange with your clients, is not only recommended but it is a staple requirement to safeguard the data while working on the Internet. The easiest way to protect your data is to hash it, but to get true protection you need full encryption.

Hashing is the process of transforming data into a fixed-length value that is a checksum of the data. The hash sum of a file is often used with downloads. For example, if any unauthorized user attempts to download a file then hash sum is compared with the site report and ultimately downloading of the file stops.

The Secure Hash Algorithm 1 (SHA1), which generates hash values from any information you pass to it, is a great algorithm which you can use to encrypt passwords or other sensitive information on your server. If you store any information that is sensitive and someone hacks into your system, the information will be freely available to them - not so if you hash it.

Data Encryption

Encryption is the process of collecting data (referred as plaintext) and encrypting the data into an unrecognizable format known as cipher text. On the other hand, decrypting is the process of converting cipher text into plain text with the help of decryption key. Without decryption key it is very difficult to decrypt the data.

Types of Encryption

You can encrypt data by using different methods, which are as follows:

- **Symmetric Encryption**—Represents a secret key that can be a number, word, or string of random letters, which is applied to the data to change the content. This might be as simple as shifting each letter by a number of places in the alphabet. As long as both sender and receiver know the secret key, they can encrypt and decrypt the data using this key.
- **Asymmetric Encryption**—Represents a key pair in which there are two keys: public key and private key. A public key is available to any user who might want to send you a message. A private key is available only to authorized users. Any message that is encrypted by using the public key can only be decrypted by using the

matching private key. Any message that is encrypted by using the private key can only be decrypted by using the matching public key.

Symmetric Encryption

Symmetric encryption uses a secret key that can be a number, word, or string of random letters, which is applied to the data to change the content. This might be as simple as shifting each letter by a number of places in the alphabet. As long as both sender and receiver know the secret key, they can encrypt and decrypt the data using this key.

In Symmetric encryption, the ROT13 algorithm is used to encrypt the data. The ROT13 algorithm is the most widely know encryption algorithm; however, it is not very secure as it can be decrypted by any user just using pencil and paper.

You can implement the ROT13 algorithm by using the str_rot13() function. It takes one parameter (the string to encrypt), and returns the encrypted version. Listing 8.13 shows how to encrypt the data using the str_rot13() function. You can find listing 8.13 in \Code\PHP\Chapter 08\Using ROT13 folder on the CD.

Listing 8.13: Using the str_rot13() Function

```
<?php
    $string = "Hello, world!\n"
    print str_rot13($string);
?>
```

The output of the preceding listing is as follows:

```
Uryyb, jbeyq!
```

Note that in Listing 8.13, every letter is shifted thirteen places to the right side in the alphabet, double letters, such as ll in Hello are replaced by the same alphabet.

Advanced Symmetric Encryption

As discussed, the ROT13 algorithm is not secured algorithm to encrypt the data. Therefore, you can use the mcrypt extension, which is the most secured and reliable extension to encrypt the data. You need to call various functions to use this extension, which are listed in Table 8.1:

Table 8.1: Commonly Used Symmetric Encryption Functions and their Parameters

Function Name	Parameters
resource mcrypt_module_open	(string algorithm, string algorithm_directory, string mode, string modedirectory)
string mcrypt_create_iv	(int size, int source)
int mcrypt_get_iv_size	(resource td)
int mcrypt_get_key_size	(resource td)
int mcrypt_generic_init	(resource td, string key, string iv)
string mcrypt_generic	(resource td, string data)
bool mcrypt_generic_deinit	(resource td)
bool mcrypt_module_close	(resource td)

Listing 8.14 shows the use of different symmetric functions to encrypt the data:

Listing 8.14: Using Symmetric Encrypted Functions

```
<?php
    srand((double)microtime()*1000000 );
    $td = mcrypt_module_open(MCRYPT_RIJNDAEL_256, '', MCRYPT_MODE_CFB, '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    $ks = mcrypt_enc_get_key_size($td);
```

```
    $key = substr(sha1('Your Secret Key Here'), 0, $ks);
    mcrypt_generic_init($td, $key, $iv);
    $ciphertext = mcrypt_generic($td, 'This is very important data');
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    print $iv . "\n";
    print trim($ciphertext) . "\n";
?>
```

In Listing 8.14, the first function called is mcrypt_module_open(), which opens an encryption algorithm. It takes four parameters, but we have given only two parameters, which are the name of the algorithm to use and the type of block cipher.

The next function called is mcrypt_create_iv(), which creates an initialization vector for encryption.

Next, the mcrypt_enc_get_iv_size() function is called, which retrieves the key size of the specified cipher. Then, the mcrypt_generic_init() function is used to initialize the buffers required to encrypt data. After this, the mcrypt_generic() function is called to encrypt data. Next, the mcrypt_generic_deinit() function is used to clear the buffers. In the end, the mcrypt_module_close() function is called to close the mcrypt module. The output of the preceding listing is as follows:

```
"Å!e□¡K.‹□#¢³ó e§□À€0–á{Y□□]Ý\¶ ô□æE =²Êœ),Pr□Í~"K□ ÀúZQ1r
```

Symmetric Decryption

After encrypting the data, you can decrypt it using the same functions listed in Table 6.1. Let's decrypt the same code as we have encrypted in the preceding listing, as shown in Listing 8.15:

Listing 8.15: Using Symmetric Functions to Decrypt the Data

```
<?php
    srand((double)microtime()*1000000 );
    $td = mcrypt_module_open(MCRYPT_RIJNDAEL_256, '', MCRYPT_MODE_CFB, '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    $ks = mcrypt_enc_get_key_size($td);
    $key = substr(sha1('Your Secret Key Here'), 0, $ks);
    mcrypt_generic_init($td, $key, $iv);
    $ciphertext = mcrypt_generic($td, 'This is very important data');
    mcrypt_generic_deinit($td);
    mcrypt_generic_init($td, $key, $iv);
    $plaintext = mdecrypt_generic($td, $ciphertext);
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    print $iv . "\n";
    print trim($ciphertext) . "\n";
    print trim($plaintext) . "\n";
?>
```

In Listing 8.15, the mcrypt_generic_deinit() function is used to encrypt the data and then the mcrypt_generic_init() function is used to decrypt the data. The output of the listing is as follows:

```
"Å!e□¡K.‹□#¢³ó e§□À€0–á{Y□□]Ý\¶ ô□æE =²Êœ),Pr□Í~"K□ ÀúZQ1r
This is very important data
```

Types of Encryption Algorithm

You can also use other encryption algorithms to encrypt the data, which are as follows:

- MCRYPT_3DES
- MCRYPT_BLOWFISH
- MCRYPT_DES
- MCRYPT_RC6_128
- MCRYPT_RC6_192

- MCRYPT_RC6_256
- MCRYPT_RIJNDAEL_128
- MCRYPT_RIJNDAEL_192
- MCRYPT_RIJNDAEL_256
- MCRYPT_SERPENT_128
- MCRYPT_SERPENT_192
- MCRYPT_SERPENT_256
- MCRYPT_TWOFISH_128
- MCRYPT_TWOFISH_192
- MCRYPT_TWOFISH_256

These algorithms are passed as a first parameter to the mcrypt_module_open() function. The Rijndael algorithm is one of the most secured encryption algorithms. When the National Institute of Standards and Technology (NIST) in the US wanted to develop a new governmental standard for encryption, known as Advanced Encryption Standard (AES), they selected Rijndael for its encryption quality, speed of encryption, and ease of implementation. However, there are two other algorithms in AES proposals, which are Serpent and Twofish. Serpent is the most secure of the three, but it is three times slower than Rijndael. Twofish is an algorithm that can encrypt data randomly.

Listing 8.16 shows how to use encryption algorithm. You can find listing 8.16 in \Code\PHP\Chapter 08\Encryption Algorithm folder on the CD.

Listing 8.16: Using Encryption Algorithm

```
<?php
    srand((double)microtime()*1000000 );
    $td = mcrypt_module_open(MCRYPT_DES, '',MCRYPT_MODE_CFB,  '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    $ks = mcrypt_enc_get_key_size($td);
    $key = substr(sha1('Your Secret Key Here'), 0, $ks);
    mcrypt_generic_init($td, $key, $iv);
    $ciphertext = mcrypt_generic($td, 'This is very important data');
    mcrypt_generic_deinit($td);
    mcrypt_generic_init($td, $key, $iv);
    $plaintext = mdecrypt_generic($td, $ciphertext);
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    print $iv . "\n";
    print trim($ciphertext) . "<br>";
    print trim($plaintext) . "\n";
?>
```

The output of the preceding is as follows:

```
±Þxìeð§¬ %Gd#YÛÕÕ\‹Û[á‹yául:¥ þÕMwÛZ
This is very important data
```

Types of Block Cipher Mode

PHP offers various block cipher modes to split the plaintext into blocks that are equal to the cipher block size. These blocks are encrypted individually. The different types of block cipher mode are as follows:

- MCRYPT_MODE_ECB
- MCRYPT_MODE_CBC
- MCRYPT_MODE_CFB
- MCRYPT_MODE_OFB

Let's discuss all these in detail.

MCRYPT_MODE_ECB

Electronic Codebook (ECB) splits the plaintext into blocks that is equal to the cipher block size, and encrypts them separately, with no further processing. However, the problem with this is that if any of the plaintext blocks are same, the encrypted version will also be the same, which makes hacking your encrypted data much easier. Therefore, ECB is quite insecure, and not recommended.

MCRYPT_MODE_CBC

Cipher Block Chaining (CBC) works with the XOR operator and converts each plaintext into block that is crated using XOR, and then encrypts it.

MCRYPT_MODE_CFB

Cipher Feedback (CFB) works with the XOR operator and converts each block of plaintext into the ciphertext.

MCRYPT_MODE_OFB

Output Feedback (OFB) creates a pseudo-random stream and then converts the plaintext into the ciphertext. OFB reduces the occurrence of an error in transmission of the encrypted data, because an error in one OFB block does not affect the other OFB blocks.

Listing 8.17 shows how to use the MCRYPT_MODE_OFB cipher mode:

Listing 8.17: Using the MCRYPT_MODE_OFB Cipher Mode

```
<?php
    srand((double)microtime()*1000000 );
    $td = mcrypt_module_open(MCRYPT_DES, '',MCRYPT_MODE_OFB, '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    $ks = mcrypt_enc_get_key_size($td);
    $key = substr(sha1('Your Secret Key Here'), 0, $ks);
    mcrypt_generic_init($td, $key, $iv);
    $ciphertext = mcrypt_generic($td, 'This is very important data');
    mcrypt_generic_deinit($td);
    mcrypt_generic_init($td, $key, $iv);
    $plaintext = mdecrypt_generic($td, $ciphertext);
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    print $iv . "\n";
    print trim($ciphertext) . "<br>";
    print trim($plaintext) . "\n";
?>
```

The output of the preceding listing is as follows:

```
Ä`YX!'0S „á [RöŽ`/Ï0f3æz1Š□□‼Ãå½œ
This is very important data
```

Let's now learn how to configure PHP security.

Configuring PHP Security

Yu can configure PHP security to reduce the risk of hackers gaining unauthorized access to your application. You can set various PHP configuration directives to configure PHP security. These directives can be set in the PHP configuration file, php.ini, or at run time, with the ini_set() function. The various directives available in PHP are as follows:

- disable_functions
- disable_classes
- allow_url_fopen
- open_basedir
- error_reporting
- display_errors

- log_errors
- expose_php
- max_input_time
- session.name

Let's discuss all these in detail.

The disable_functions Directive

The disable_functions directive allows developers to disable certain built-in PHP functions, such as exec() and passthru(), for security reasons.

The disable_classes Directive

The disable_classes directive allows developers to display certain PHP classes that pose a security risk.

The allow_url_fopen Directive

The allow_url_fopen directive determines whether or not a PHP script can read data of remote URLs. This data is read by the file functions, such as file_get_contents(), include(), or fopen().

The open_basedir Directive

The open_basedir directive allows developers to restrict all file operations of a particular directory and its subdirectories within a PHP script. When this directive is enabled, a PHP script is not able to view the top-level directory named in this directive. This directive is useful to restrict an application to a defined directory tree, and reduce the possibility of its accessing or manipulating sensitive system files.

The error_reporting Directive

The error_reporting directive performs the same function, it allow the developer to control the error reporting level. The PHP security suggests that in the most of the cases, this should be set to E_ALL, so that all errors (notices and warning) are reported to the developer.

The display_errors Directive

The display_errors directive checks the errors reported by a script are actually displayed in the output. The PHP manual recommends enabling this directive in development environments, but disabling it in production environments, as hackers can often use the diagnostic information displayed in an error message to locate and exploit vulnerabilities in your PHP code.

The log_errors Directive

The log_errors directive specifies that the errors occurring in a PHP script should be written to a log file for later analysis. In most cases, this directive should be enabled, especially if display_errors is turned off, so that you can have record of the errors generated by a script.

The expose_php Directive

The expose_php directive determines whether or not PHP adds information about itself to the Web server context. The PHP manual recommends disabling this directory to avoid providing potential hackers with additional information about the capabilities of the Web server.

The max_input_time Directive

The max_input_time directive determines the maximum amount of time a PHP script has taken to receive and parse input data, including data passed through the GET and POST methods. A limit on this time interval reduces the time available to a hacker attempting to interactively create and transmit a POST or GET request.

The session.name Directive

The session.name directive checks the name of the session cookies used by PHP to track user sessions. The name of this cookie is PHPSESSID, by default. You can change the name of this cookie to make it difficult for attackers to identify it and view its content.

NOTE

You must restart the Web server to activate changes made to the PHP configuration file, php.ini.

With this, we come to the end of the chapter. Let's now summarize the main topics learned in this chapter.

Summary

In this Chapter, you have learned about cookies and sessions that are used to store data. The attributes, such as expiration date, name-value, valid domain, valid path, and security flag, of cookies are also discussed in the chapter. You have also learned about creating, reading, and removing cookies. Next, you have learned about sessions. The chapter also discussed about starting, adding, reading, removing, and ending a session. Further, you have learned to prevent a session hijacking. The difference between the cookies and sessions is also explained in the chapter. You have also learned to protect data using various encryption algorithms. In the end, you have learned about configuring PHP security that helps in securing data.

In the next chapter, we will discuss about new technology i.e. Java, used for creating web applications.

Quick Revise

Q1. True/False:

a. Session is a combination of server-side cookies. **False**

b. A session creates a file in a temporary directory on the server. **True**

c. A PHP session is start by calling the session_start() function. **True**

d. The size of a cookie depends on the server. **False**

f. Cookies are created in PHP using the setcookie() function. **True**

g. The ROT13 algorithm is a secure algorithm. **False**

h. It is possible to store an array in a cookie. **True**

i. Session variables are stored in associative array called $_SESSION []. **True**

j. mcrypt_generic_init() function initializes all buffers needed for decryption. **False**

k. Session Prediction refers to guessing a valid session identifier. **True**

l. Decrypting is the process of converting cipher text into plain text with the help of decryption key. **True**

m. ROT13 algorithm is used rot13() function. **False**

n. mcrypt_generic() function encrypts data. **True**

o. Hashing is the process of transforming data into a fixed-length value that is a checksum of the data. **True**

p. Public and private key are used in symmetric encryption. **False**

q. It is necessary to restart the Web server in order to activate changes made to the PHP configuration file, php.ini. **True**

r. MCRYPT_MODE_ECB stands for Electronic Code block. **False**

s. DES stands for Data Encryption Standard. **True**

t. SHA1 generates the hash values. **True**

u. RIJNDAEL algorithm is an unsecure encryption algorithms. **False**

Q2. What is Cookie? How it is create and read.

Ans: Cookies are used to store data in the remote browser and help to track or identify data returned to users on the Web browser.A cookie is a small bit of information stored on a computer of a user by request from a Web page. This information is constantly passed between the Web browser and Web server, the browser sends the current cookie as a request to the server and the server sends updated data back to the user as response. The size of a cookie depends on a browser but in general should not exceed 1K (1,024 bytes).

- **Creating a Cookie**—Cookies are creates by using the setcookie() function. The setcookie() function takes number of arguments, such as name value, expiration date, and domain name.

Example:

```
<?php
setcookie('username', 'Rahul', time() + 3600);
echo "Cookies has been set";
?>
```

The output of the preceding example is as follows:

```
Cookies has been set
```

- **Reading a Cookie**—The PHP $_COOKIE variable is used to retrieve a cookie value.

Example:

```
<?php
setcookie('username', 'Rahul', time() + 3600);
echo 'UserName = ' . $_COOKIE['username'];
?>
```

The Output of the preceding example is as follows:

```
UserName = Rahul
```

Q3. What is session? How it is starting, adding value, and reading a session.

Ans: A session stores different information for each user accessing a Web site. A session includes server-side and client-side cookies; where the client-side cookie contains a reference to the requested data on the server. A session creates a file in a temporary directory on the server, where registered session variables and their values are stored. This data is available to all the Web pages of the Web site when the user is accessing the specified Web site.

- **Starting and Adding a Session**—Session variables are stored in an associative array called $_SESSION[] (session global array). These variables can be accessed during the entire lifetime of a session. A PHP session is easily started by makaing a call to the session_start() function. This function first checks whether the specified session has already been started.

  Session data is stored in the $_SESSION[], which means that each session variable, along with its value, is one element in that array. You can add variables to this array similar to adding variables to any array.

Example:

```
<?php
session_start();
$_SESSION['firstname']="Rahul";
$_SESSION['lastname']="Sharma";
?>
```

- **Reading a Session**—After you have saved the session data, it becomes available in the $_SESSION[] along with the key of the variable name you gave it.

Example:

```
<?php
session_start();
$_SESSION['firstname']="Rahul";
$_SESSION['lastname']="Sharma";
```

```
echo "My name is ". $_SESSION['firstname']." ". $_SESSION['lastname']. ".";// reading a
    session
?>
```

The output of the preceding example is as follows:

```
My name is Rahul Sharma.
```

Q4. Describe the difference between the Cookie and Session?

Ans: Differentiating Cookies and Sessions

Both cookies and sessions accomplish much the same task of storing data across Web pages on your Web site. However, there are few differences between the working of cookie and session. Depending upon your requirement, you can choose any one or combination of both.

Cookies can be set to a longer lifespan, which means data can be stored in cookies for months. Once the data is stored in a cookie, it works smoothly when you have a cluster of web servers (a group of independent web servers). Sessions are stored on the server, which means clients do not have access to the information you store in the session. At the time of starting a session, user needs to provide a unique ID for authentication and after successful authentication data is loaded on the Web page from the Web server. Sessions are important when you do not want to share classified information with other users. Also, session can store any amount of data whereas cookie can store only limited amount of data.

Q5. How to prevent Session Hijacking?

Ans: One of the fundamental principles of Web application security is not to trust data from any unknown or unauthorized client. Therefore, the client must identify itself by sending a unique identifier. The issue of identifying a client as an authorized client creates a fundamental conflict among developers wanting to build secure, stateful applications. In fact, the session mechanism in any Web application is the most vulnerable feature of that application, and session security is one of the most complex topics of Web application security on any platform.

There are numerous types of session-based attacks. Many of these fit into a category called impersonation (session hijacking), where a malicious user attempts to access another user's session by posing as that user. At the very least, these types of attacks require that the malicious user obtain a valid session identifier, because this is the minimum amount of information that must be used for identification.

There are at least three ways that a valid session identifier can be obtained by an attacker:

- **Session Prediction**—Refers to guessing a valid session identifier. This guess can range from a wild guess to an educated one, depending upon the sophistication of the attack being used. With PHP's native session mechanism, valid session identifiers are extremely difficult to predict, so this is unlikely to be the weakest point in your implementation.
- **Session Capturing**—Specifies a valid session identifier, which is much more common and there are numerous types of attacks using this approach. When a cookie stores the session identifier, the Web browser vulnerability might be exploited to obtain the session identifier. When a URL variable is used, the session identifier is exposed, and there are many more potential methods to capture a session. For this reason, cookies are generally considered to be more secure than URL variables.
- **Session Fixation**—Specifies a method that tricks a user to select a session identifier already chosen by the attacker. Session fixation attacks attempt to exploit the vulnerability of a system which allows one user to set session identifier for another user. Most session fixation attacks are Web based, and rely on session identifiers, which accepts URLs (query string) or POST data.

Q6. What is symmetric encryption and decryption?

Ans: There are many users waiting on the Internet to gain unauthorized access to your information and misuse it. Therefore, practicing encryption, both for data you store locally and you exchange with your clients, is not only recommended but it is a staple requirement to safeguard the data while working on the Internet. The easiest way to protect your data is to hash it, but to get true protection you need full encryption.

Encryption is the process of collecting data (referred as plaintext) and encrypting the data into an unrecognizable format known as cipher text. On the other hand, decrypting is the process of converting cipher text into plain text with the help of decryption key. Without decryption key it is very difficult to decrypt the data.

Symmetric encryption uses a secret key that can be a number, word, or string of random letters, which is applied to the data to change the content in unreadable form.

Example:

```
<?php
    srand((double)microtime()*1000000 );
    $td = mcrypt_module_open(MCRYPT_RIJNDAEL_256, '', MCRYPT_MODE_CFB, '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    $ks = mcrypt_enc_get_key_size($td);
    $key = substr(sha1('Your Secret Key Here'), 0, $ks);
    mcrypt_generic_init($td, $key, $iv);
    $ciphertext = mcrypt_generic($td,' Ram is a good boy');
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    print $iv . "\n";
    print trim($ciphertext) . "\n";
?>
```

The output of the preceding example is as follows:

```
NüÒÝµ□w-rs□A-O□â wa□°2\ª^IðbŽ□‡1 □å}6k úywÁ□n„w$H,C
```

Symmetric decryption, after encrypting the data, you can decrypt the data using the same functions and same key.

Example:

```
<?php
    srand((double)microtime()*1000000 );
    $td = mcrypt_module_open(MCRYPT_RIJNDAEL_256, '', MCRYPT_MODE_CFB, '');
    $iv = mcrypt_create_iv(mcrypt_enc_get_iv_size($td), MCRYPT_RAND);
    $ks = mcrypt_enc_get_key_size($td);
    $key = substr(sha1('Your Secret Key Here'), 0, $ks);
    mcrypt_generic_init($td, $key, $iv);
    $ciphertext = mcrypt_generic($td, 'Ram is a good boy');
    mcrypt_generic_deinit($td);
    mcrypt_generic_init($td, $key, $iv);
    $plaintext = mdecrypt_generic($td, $ciphertext);
    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);
    print $iv . "\n";
    print trim($ciphertext) . "\n";
    print trim($plaintext) . "\n";
?>
```

The output of the preceding example is as follows:

```
□□□,□-9o+ŽÀ zÃ)3c²zò.µ¼¢ƒ";á1- □çY1i,=Q□0"b8«çfo
Ram is a good boy
```

Q7. Explain how many type of block cipher modes used in PHP?

Ans: PHP offers various block cipher modes to split the plaintext into blocks that are equal to the cipher block size. These blocks are encrypted individually. The different types of block cipher mode are as follows:

- **MCRYPT_MODE_ECB**—Electronic Codebook (ECB) splits the plaintext into blocks that is equal to the cipher block size, and encrypts them separately, with no further processing. However, the problem with this is that if any of the plaintext blocks are same, the encrypted version will also be the same, which makes hacking your encrypted data much easier. Therefore, ECB is quite insecure, and not recommended.

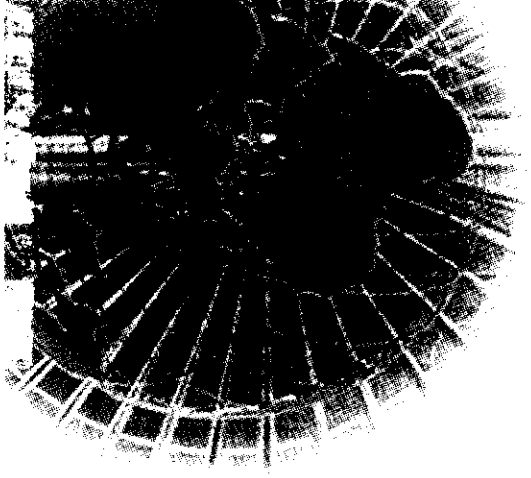
- **MCRYPT_MODE_CBC**—Cipher Block Chaining (CBC) works with the XOR operator and converts each plaintext into block that is crated using XOR, and then encrypts it.
- **MCRYPT_MODE_CFB**— Cipher Feedback (CFB) works with the XOR operator and converts each block of plaintext into the ciphertext.
- **MCRYPT_MODE_OFB**— Output Feedback (OFB) creates a pseudo-random stream and then converts the plaintext into the ciphertext. OFB reduces the occurrence of an error in transmission of the encrypted data, because an error in one OFB block does not affect the other OFB blocks.

Q8. Explain how to configure PHP security?

Ans: Configure PHP security is used to reduce the risk of hackers gaining unauthorized access to your application. You can set various PHP configuration directives to configure PHP security. These directives can be set in the PHP configuration file, php.ini, or at run time, with the ini_set() function. The various directives available in PHP are as follows:

- **The disable_functions Directive**—The disable_functions directive allows developers to disable certain built-in PHP functions, such as exec() and passthru(), for security reasons.
- **The disable_classes Directive**—The disable_classes directive allows developers to display certain PHP classes that pose a security risk.
- **The allow_url_fopen Directive**—The allow_url_fopen directive determines whether or not a PHP script can read data of remote URLs. This data is read by the file functions, such as file_get_contents(), include(), or fopen().
- **The open_basedir Directive**—The open_basedir directive allows developers to restrict all file operations of a particular directory and its subdirectories within a PHP script. When this directive is enabled, a PHP script is not able to view the top-level directory named in this directive. This directive is useful to restrict an application to a defined directory tree, and reduce the possibility of its accessing or manipulating sensitive system files.
- **The error_reporting Directive**—The error_reporting directive performs the same function, it allow the developer to control the error reporting level. The PHP security suggests that in the most of the cases, this should be set to E_ALL, so that all errors (notices and warning) are reported to the developer.
- **The display_errors Directive**—The display_errors directive checks the errors reported by a script are actually displayed in the output. The PHP manual recommends enabling this directive in development environments, but disabling it in production environments, as hackers can often use the diagnostic information displayed in an error message to locate and exploit vulnerabilities in your PHP code.
- **The log_errors Directive**—The log_errors directive specifies that the errors occurring in a PHP script should be written to a log file for later analysis. In most cases, this directive should be enabled, especially if display_errors is turned off, so that you can have record of the errors generated by a script.
- **The expose_php Directive**—The expose_php directive determines whether or not PHP adds information about itself to the Web server context. The PHP manual recommends disabling this directory to avoid providing potential hackers with additional information about the capabilities of the Web server.
- **The max_input_time Directive** —The max_input_time directive determines the maximum amount of time a PHP script has taken to receive and parse input data, including data passed through the GET and POST methods. A limit on this time interval reduces the time available to a hacker attempting to interactively create and transmit a POST or GET request.
- **The session.name Directive**—The session.name directive checks the name of the session cookies used by PHP to track user sessions. The name of this cookie is PHPSESSID, by default. You can change the name of this cookie to make it difficult for attackers to identify it and view its content.

PART 3
CREATING JAVA WEB APPLICATIONS



9

Getting Started with Web Applications in Java

You may have worked on different types of applications while performing your day to day activities on a computer. These applications can be categorized into different categories according to the technologies and platforms being used by the applications. At a broader level, the applications can be categorized into two categories, Windows applications and Web applications. Windows applications are standalone applications and function on different computers individually; whereas Web applications run on a server and are accessed by multiple clients. With the advent of new Internet technologies, the number of users interacting with different Web applications has increased. As a result, the number of Web applications that need to be developed for various purposes is increasing day by day. These Web applications can be developed by using different technologies such as ASP.Net or Java Servlets.

Before we start describing different Java technologies, which are used to develop different components involved in the functioning of a Web application, we need to discuss various aspects of a Web application in detail. In this chapter, we learn about a brief introduction of different Java technologies available and used to develop Web applications.

Introduction to Web Applications

A Web application can be defined as an application that can be accessed through the Internet using a Web browser. You must have interacted with some Web applications that process your input data and provide you with the required information. For example, the official Web sites of different banks provide information about clients and their accounts, in addition to allowing different transactions online. In other words, we are talking about a Client-Server based application that runs on a server and is accessed by multiple client computers.

In earlier times, a client-server application had its own client programs. A client program served as a user interface to the client and needed to be installed on every client computer. However, with the advent of Web applications, client-server applications have changed a lot. A Web application provides a series of Web pages to a client; and these Web pages can be accessed by all types of clients using any browser of their choice. Usually, a Web page is delivered to the client as a static document, but the sequence of Web pages, one after another, provides an interactive environment to the client. In addition, you do not need to install separate client programs on all client computers as the Web browser being used interprets and displays all the Web pages and works as a common client for a Web application. This reduces the time to develop Web applications without spending t on creating separate client programs supported by different platforms.

Let's now discuss various technologies used to develop Web applications in Java.

Exploring Java Based Web Technologies

Java Servlets and Java Server Pages (JSPs) are also referred to as Web technologies. They can be combined together to design efficient content publishing systems that support separation of presentation and business logic of Web applications.

While building the server-side Web applications, we separate the presentation and the logic. This allows the less-experienced Web designers to generate the Web pages with dynamic content easier and faster. This separation process is highly beneficial where content changes frequently, and helps present new information to Web site visitors faster.

Earlier, Web applications faced maintenance problems. Using the Model/View/Controller (MVC) paradigm for building user interfaces has solved this problem. In MVC, the back-end system is the Model, the templates for creating the look and feel of the response is the View, and the code that combines them all together is the Controller. Sun Microsystems introduced two solid technologies for implementing MVC architecture, which are Java Servlets and JSP. JSPs fit perfectly into this solution as a way of creating a dynamic response or View. Servlets contain the logic for managing requests and act as the Controller, while the existing business rules act as the Model. Since the introduction of Java Servlet and JSP technology, additional Java technologies and frameworks for building interactive Web applications have been developed. In this section, we briefly introduce some Java based Web technologies, which are:

- Servlets
- JSP

Both Servlets and JSP use Java Database Connectivity (JDBC) for handling database operations in Web applications. Therefore, before starting discussion on Servlets and JSPs, let's first understand what JDBC is.

Describing JDBC

JDBC™ is a specification from Sun Microsystems, and provides a standard abstraction (API / Protocol) for Java applications to communicate with different relational databases. It is based on X/Open Call Level Interface (CLI) and uses Structured Query Language (SQL) as its database access language. Before the advent of JDBC, if an application needed to interact with a database, the interaction was carried out by using Native Libraries provided by database vendors and Open Database Connectivity (ODBC) API.

However, there are many disadvantages of using these approaches to interact with a database. These are as follows:

- Disadvantages of Native Libraries provided by database vendors:
  - Application becomes vendor dependent
  - Application has to use Java Naming and Directive Interface (JNDI) to interact with Native Libraries, which may cause serious problems for platform independency in Web applications
  - If there is any problem in the Native Library being used, it may destroy the Java Virtual Machine (JVM); that is, bugs in Native libraries used can crash the JVM
- Disadvantages of using ODBC API
  - ODBC API can solve the problems raised by Native Libraries as it provides a common API to interact with any database that has an ODBC Service Provider's implementation written in Native API. ODBC API requires a Data Source Name (DSN) to be configured on each computer that interacts with the database.
  - In ODBC, data and SQL call statements are passed through the ODBC layer, which may slow down the data transfer speed. Therefore, ODBC API is slower than the Native Libraries approach.

To solve the preceding problems, we require a generic Java abstraction that allows Java applications to communicate with different databases. To meet this requirement, Sun Microsystems has come up with a common specification API, and this specification is called JDBC.

Exploring the JDBC Architecture

The JDBC API supports both 2-tier and 3-tier applications. First, let's discuss about the 2-tier architecture, as shown in Figure 9.1:

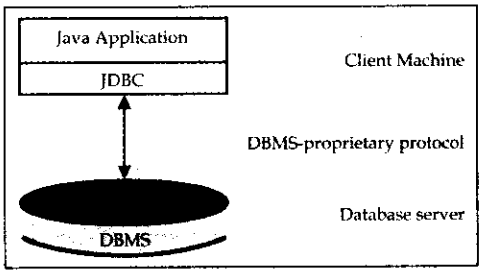


Figure 9.1: Displaying the JDBC 2-tier Architecture

As shown in Figure 9.1, in the 2-tier architecture, a Java application directly communicates with the data source using JDBC APIs. The communication requires a JDBC driver that accesses the particular database. The client computer (on which the Java application is running) delivers the commands to the database or other data source and the results are sent back to the client computer by the database server (Server computer). The database or data source may be located on the client's computer or on another computer connected through a network. This is referred as the client-server configuration.

Now we consider the 3-tier architecture, as shown in Figure 9.2:

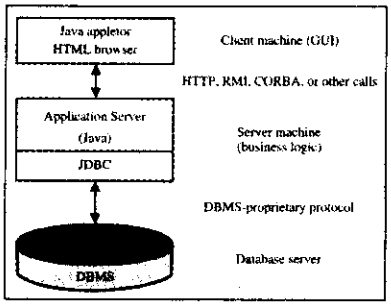


Figure 9.2: Displaying the JDBC 3-tier Architecture

As shown in Figure 9.2, in the 3-tier architecture, the client delivers the command to a middle tier, which then sends the command to the database or other data source. The data source then processes the commands and the results are sent back to the middle tier. The middle tier then sends the results back to the client. Here, the middle tier implements the JDBC driver to interact with the particular database.

Therefore, if we want to communicate with any database through JDBC, we need a JDBC driver, which intelligently communicates with the respective database. Currently, more than 220 JDBC drivers are available in the market and have been designed to communicate with different data stores. All these drivers are categorized into four types, based on their implementation (that is, the approach to communicate with the data store). These categories are popularly known as JDBC Type-1, Type-2, Type-3, and Type-4 Driver. We learn about these four JDBC driver Architectures in *Chapter 13, Java Database Programming*.

Describing Servlets

Let's now understand the evolution of Java Servlets. We can access simple, static HTML pages by using a Web browser, which sends requests to a Web server. The Web server sends back the requested Web page stored on the server by using HTTP. These Web pages are static, which means they exist in a constant state, for example, a text file that does not change.

However, with changing technologies and requirements, we now need to design the Web server to process the data given by the client and display dynamic content to the client. To add this dynamic behavior, we need to develop applications that can run on the server, are integrated with the Web server, process client requests, and generate the response content dynamically.

To meet this requirement initially, we have some proprietary solutions such as some APIs from various Web server providers, which allow us to build programs that can run with Web servers. However, this approach to build server-side applications results in the following problems:

- The application becomes vendor dependent
- Migrating the application from one server to another becomes difficult
- While developing the application, we have to decide the server on which the application has to be deployed, and consequently, we have to learn the vendor specific API to develop the application.

CGI came up as a solution for the preceding stated problems. CGI is an open standard abstraction between HTTP adapter (Web server) and an external application written by using any of the programming language, such as C or C++. This external application is known as CGI application and is responsible to process client requests and generate dynamic content.

Some of the benefits of CGI are as follows:

- **Simplicity**—Easy to understand and implement.
- **Open standards**—Allows us to deploy the CGI application on different Web servers.
- **Architecture Independent**—Does not depend on any particular Web server architecture.
- **Language Independent**—Can be written in any language.

- **Process Isolation**—Allows to run the CGI applications in a separate process, that is, outside the Web server process. Consequently, if there is any bug in the CGI applications, it will not crash the Web server process.

In the CGI approach, the server creates and destroys the process for every request, which increases the workload on the server and the time taken to process the requests. This, in turn, reduces the availability of the Web applications. This approach works well for few clients, but as the number of clients increase, the Web server using the CGI applications is not able to serve the clients. Therefore, CGI entailed the following problems:

- Low Performance
- Poor Efficiency
- Stateless support

Java Servlets came as an alternate to CGI, by providing high level, component-based, platform-independent, and server-independent standards for developing Web applications in Java. Java Servlets technology provides a simple, vendor independent mechanism for extending the functionality of Web servers.

Advantages of Servlets

The advantages of Servlets over other server-side extension abstraction, such as CGI, are as follows:

- Servlets are faster than CGI.
- They use a standard vendor-independent API that is supported by numerous Web servers.
- Servlets provide a high level API to programs.
- They provide declarative security management support that allows us to easily build and modify the security logic for server-side extensions.
- Java Servlet extends the functionality of a Web server. Servlets function as applets that run on the server. These are portable platform and Web server independent means to deliver dynamic content. A browser-based application that calls the Servlets need not support the Java programming language. This is because of the fact that the output of a Servlet may be of the HTML, XML, or any other content type.
- Servlets are written in the Java programming language. This allows Servlets to function on any platform that has a JVM and a Web server that supports Servlets. Servlets can communicate directly with existing enterprise resources by using generic APIs, such as JDBC. Consequently, application development becomes simple and easy.
- Servlets are extensible, implying that by using Servlets, developers can extend the functionalities of a Web application similar to any Java application. For example, a Controller Servlet is extended to become a secure Controller. The functionalities of the original Controller are also retained along with the addition of new security features.
- Servlets perform better than the CGI scripts. For example, once a Servlet is loaded into memory (a Servlet can be loaded into memory once and then called as many times as needed without requiring additional hardware.), it can run on a single lightweight thread; while CGI scripts must be loaded in a different process for every request. Another benefit of Servlets is that a Servlet can maintain and/or pool connections to databases or other necessary Java objects; which saves time in processing requests.
- Servlets eliminate the complexity of retrieving parameters from an HTTP request; components have direct access to parameters because parameters are presented as objects. With the CGI-based applications, parameters submitted from a form are converted to environment properties, which must then read into a program.
- Servlets provide uniform APIs to maintain the session data throughout a Web application and interact with user requests. We know that HTTP is a stateless protocol; means it does not maintain the state of user requests. To overcome the limitations of Web applications due to the stateless nature of HTTP, we can use session data to maintain the state of user requests.
- Servlets are accessible with wide range of service APIs available under the Java platform; which allows us to solve complex problems by using a simple API.

Describing JavaServer Pages (JSP)

We learnt that Java Servlets are used to develop server side extensions for presenting dynamic content to clients. However, one important thing that we observe while using Java Servlet technology for creating dynamic views is the numerous use of the out.println statements. These out.println statements make the Java Servlets' code very complex and difficult to understand. Most of the code lines in a Servlet contain static content presentation statements (in the out.println statements) and contain very less java logic, which leads to the following problems:

We cannot take the advantage of the IDEs to design html views, which increases the development time and cost of developing an application.

We are not able to utilize the html designer services to prepare the presentation view; instead we need to depend on Java programmers to prepare the views. In other words, the presentation views have to be built by respective programmers.

To solve the preceding problems, we need to include additional instructions for processing dynamic content in the HTML pages. Therefore, a page that includes some additional instructions has to be parsed and resolved into a standard server side extension (or Servlet) that are understood by the Web container.

Tag based additional instructions to process dynamic content provide the following advantages:

- Can be easily used by the Web designers
- The format of tag based instructions are understood by the IDEs used to design the HTML views.

This requirement of tag based instructions has necessitated the introduction of JSP. The JSP technology clearly differentiates the application logic and content. As a result, content providers are not required to have the knowledge of the Java technology to update or maintain the content. Instead, they can design interfaces by using the JavaBeans components and custom tags provided by Web Application developers. Similarly, Web application developers do not need to be experts in user interface design to build Web applications. In addition to the feature of separating the presentation and application content, the JSP technology provides all the benefits offered by Java Servlets.

Request Processing of a JSP Page

Let's now describe how a Web container processes a JSP page to handle an incoming user request, as shown in Figure 9.3:

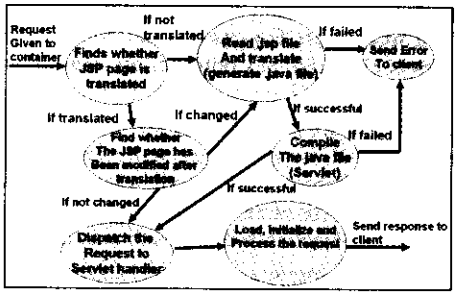


Figure 9.3: Displaying the Request Processing by JSP Page

Figure 9.3 shows how the requests are processed by a JSP page.

After understanding the request processing, let's now explain the life-cycle of a JSP page.

Describing the JSP Life-Cycle

A Web container processes and manages JSP pages through a well-defined life-cycle that defines how a JSP page is translated, compiled, loaded, and initialized. The JSP life-cycle also defines how to handle and destroy user requests.

The life-cycle of a JSP document is defined under five phases, which are:

- **Page Translation**—Allows the Servlet container to translate the JSP document into equivalent Java code, that is, Servlet.

- **Compilation**—Compiles the JSP page immediately after the page translation phase. In this phase, the Servlet container compiles the Servlet into the Java byte (class) code. After compilation the class file is generated, the container can decide to discard or retain the Java source code. In general, most of the containers discard the generated Java source code by default, unless the source code needs to be retained for debugging.
- **Loading and Initialization**—Allows the Web container to load the generated and compiled Servlet and create an object of the Servlet. If this phase is performed successfully, the Web container puts the Servlet object into the active state, that is, makes it available and ready to handle requests.
- **Request Handling**—Allows to handle the client requests, the Web container uses a JSP page's Servlet object that is initialized successfully. In this phase, the container creates the `ServletRequest` and `ServletResponse` types of objects. If the client uses HTTP to send requests, the Web container creates the `HttpServletRequest` and `HttpServletResponse` types of objects. The request object represents the request (the requested data), and can be used to retrieve client information. The response object can be used by the Servlet to generate the response. The Servlet then invokes the *`_jspService()`* method on the Servlet object and passes the created request and response objects to the `_jspService()` method.
- **Destroying**—Allows the Servlet container to destroy the JSP page's Servlet instance or end the services provided by the Servlet instance. The Servlet container performs the following operations in this phase:
  - It allows all the threads currently in the service method of the Servlet instance to complete their jobs. Meanwhile, it makes this instance unavailable for new requests.
  - After the current threads have completed their jobs, the Servlet container calls the *destroy ()* method on the Servlet instance..
  - After the destroy method is executed, the Servlet container releases all the references of the Servlet instance and makes it available for garbage collection.

Till now we discussed about the Web application technologies, now we will discuss about the architecture of the Web applications.

Introducing Web Architecture Models

Different programmers may design an application in different ways, each providing the same functionality. Designing an application depends upon how the programmers recognize the problem and apply their respective approach to solve it. Therefore, it becomes difficult for other programmers to understand the flow of a program.

Development models solve the problem of flow control and help other developers to understand the flow of a program. They provide a standard way of flow control in an application and describe the technologies used in different parts of the application. A development model facilitates the design process by separating the code according to the functions performed by different components of the Web application.

Two types of development models are used in Java for Web applications. These models are classified on the basis of different approaches used to develop Web applications. These models are:

- Model 1 Architecture
- Model 2 Architecture

Let's learn about these models in detail.

Describing the Model-1 Architecture

The Model 1 architecture, shown in Figure 9.4, was the first development model used to develop Web applications. Perhaps this was the reason behind its name being coined as Model 1. This model uses JSP to design applications; therefore, JSP is responsible for all the activities and functionalities provided by an application. An application using the Model 1 architecture contains a number of JSP pages; with each page providing different functionality and view to different users. Figure 9.4 show the Model-1 architecture:

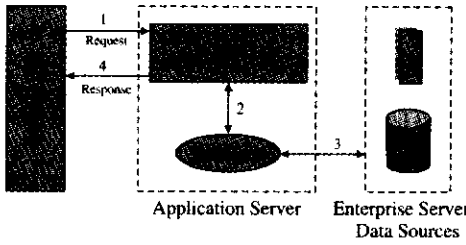


Figure 9.4: Displaying the Model-1 Architecture

In Model 1, Web applications are developed by mixing the business logic and the presentation logic. In this model, JSP pages receive the HTTP requests, which are then transferred to the data layer through JavaBeans. After the requests are serviced, a JSP page sends the HTTP response back to the client. A JSP page not only contains the display elements, but also retrieves HTTP parameters, calls the business logic; and handles the HTTP session.

Model 1 is page-centric and is suitable for small Web applications only. Web applications implementing Model 1 have a series of JSP pages; where the user navigates from one page to another. The Model 1 architecture is not suitable for large Web applications because of its limitations, discussed in the following subsection.

Limitations of Model-1 Architecture

Despite its simple structure and easy to learn features, Model 1 was not successful to design large projects because of some limitations. These limitations are as follows:

- Applications are inflexible and difficult to maintain. A single change in one page may cause changes in other pages, leading to unpredictable results.
- Developer is involved in both the page development and the business logic implementation stage. There is no provision for the division of labor between page designer and business logic developer.
- The complexity of the program increases with the increase in the size of a JSP page; therefore it becomes difficult to trace the flow of control and debug the program.
- Each page must be individually responsible for handling the Web application, verifying the input, and ensuring security.
- A series of JSP pages are included in Model 1, which increases the maintenance effort required per page.

Describing the Model-2 Architecture

The drawbacks in the Model 1 architecture lead to the introduction of a new model, named Model 2.

The Model 2 architecture was targeted at overcoming the drawbacks of Model 1 and helping the developers to design more powerful Web applications. Since it came after the advent of Model 1, it was named Model 2 to recognize it as a part of the Model 1 series. Figure 9.5 shows the Model 2 architecture:

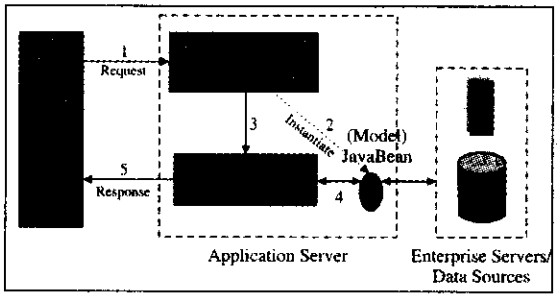


Figure 9.5: Displaying the Model-2 Architecture

Originally, JSPs were introduced as an alternative to Servlets, as JSP is more powerful than Servlet. Developers soon realized that JSPs and Servlets can be used together to develop a Web application. Servlets handle the

control flow while JSPs handle HTML page creation. In due course of time, the approach of using JSPs and Servlets together evolved as the Model 2 architecture. In Model 2, the presentation logic is separated from the business logic.

This design model is based on the Model-View-Controller (MVC) design model. As you can see from Figure 9.5, the Model 2 architecture (or MVC) contains the following components:

- **Model**—Represents enterprise data and the business rules that specify how data is accessed and updated. Model is generally implemented by using JavaBeans.
- **View**—Renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. The View component is designed by using JSP.
- **Controller**—Receives the HTTP requests. The Controller component receives requests from a client, determines the business logic to be performed, and delegates the responsibility to produce the next phase of the user interface to an appropriate View component. The Controller has complete control over each View; implying that any change in the Model is immediately reflected in all the Views. The Controller component is implemented by Servlets.

Advantages of Model 2 Architecture

So far Model 2 is the most successful development model to develop Web applications. It not only overcomes the limitations of Model 1, but also provides new features that have their own advantages. The following are the new advantages of the Model 2 architecture:

- The business logic is designed by using reusable software components. That is, these components can be used in the business logic of other applications.
- Model 2 offers great flexibility to the presentation logic. The presentation logic can be modified without affecting the business logic.
- Each software component performs a different task. This makes it easier to design an application by simply embedding these components in the application.

We learned earlier that the Model 2 architecture resembles the classical Model-View-Controller (MVC) architecture. The detailed description of the MVC architecture is presented in the next section.

Introducing the MVC Architecture

Today, a Web application may need to interact with different types of clients with different types of user interfaces. A Web application may require representing an HTML view to an ordinary customer, a WML view to a wireless customer, a JFC/Swing view to the administrator, and an XML view to some other type of client.

When a Web application needs to support only one type of client with one type of interface, it is always a good idea to combine client-specific data and business logic with the interface-specific logic. However, if this type of approach is used for Web applications that require interaction with multiple types of clients, it would require different applications to support each type of client. This leads to the repetition of non-interface specific code in the interface-specific code, which further leads to increased effort in designing, debugging, and maintenance of Web applications.

The solution of this problem is to use the MVC architecture while designing Web applications. The idea behind the MVC architecture is to separate the core business model functionality of the Web application from the presentation and control logic of the interface-specific application. With this sort of arrangement, it is possible to see multiple views of the same data model of the Web application. Under this architecture, data (Model) and user interface (View) are separated from each other so that any change in the user interfaces does not affect the data in the database, and vice-a-versa. The three components of the MVC, the Model, the View, and the Controller, must communicate with each other if the Web application needs to ensure a coherent interaction with the user. The communication between the View and the Controller is straightforward because these two components are specifically designed to work together. However, the Model communicates in a more subtle manner. Figure 9.6 shows the interaction among the various MVC components:

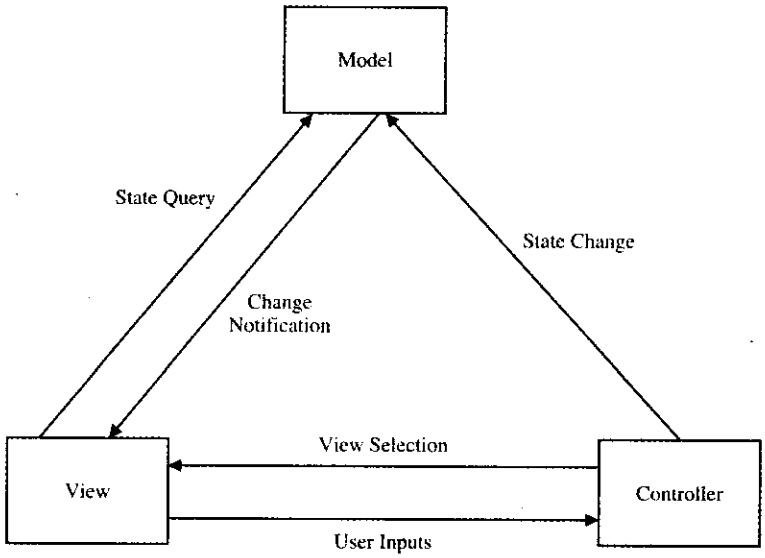


Figure 9.6: Displaying the Interaction among the MVC architecture components

In Figure 9.6, interaction among three components of MVC architecture, i.e. Model, View, and Controller, is shown.

Let's learn more about each of these components next.

Describing the Model Component

The Model displays the data on which an application is based. In a Web application, Java beans hold the data needed by the Web application. Events sent to the Web Controller serves the basis for all modifications to the data. Model represents the data and the business logic of the application and has no concern with the presentation of the application. MVC does not provide the specification for data access logic. It performs its interfacing with other components with a set of public methods.

An application may have many states that need to be stored somewhere. These states are encapsulated by the Model component of the application. All the functionalities and features of the application are provided by the Model. The application behaves according to the business logic of the application, implemented by the Model. All the Views associated with a Model are notified immediately regarding any state change in the Model, and this change is reflected in all the Views.

Describing the View Component

A View presents the data represented by the model in a way that is targeted at a specific type of client. Most Web applications support a number of different views. The same Model can have a Swing view or a Web view. The View for a Web application consists of JSP files, which have the sole responsibility to display the Model data.

The View provides Graphical User Interface (GUI) for the Model. The user interacts with the application through a View. It represents the information based upon the Model and provides the user with an ability to alter the data. A Model can have multiple Views. The information provided by the Model has a different meaning for different users and is interpreted by them in different ways. The View is responsible for representing the information to the user in a form understandable by him/her.

- The Model manages the database, whose content can be changed and updated frequently. Any change in the database must be reflected to the user as soon as possible. Therefore, the View communicates with the Model to get the state change information in case any change takes place in the database. The user, if he/she wishes to modify the content of the Model, does not communicate with the Model directly. The user

communicates with the Model through the View, which further communicates with the Controller regarding the user input. The Controller then makes the required changes in the Model and also notifies all the other associated Views.

Describing the Controller Component

The Controller is responsible for controlling all the Views associated with a Model. When a user interacts with the View and tries to update the Model, the Controller invokes various methods to update the Model. The Controller also controls the data flow and transformation between Model and View.

The behavior of a Web application is determined by the behavior of the various MVC components. The interaction among various components is managed by the Controller. The Controller, by controlling the flow among these components, determines the way the application should perform the intended tasks.

Let's understand how the Controller performs its role when a user needs to change the data stored in the Model. If the user wishes to change this data, it sends a request to the Controller, which further consults the Model to update all the Views. The process followed to change the data is as mentioned:

- The user sends a request through an interface provided by the View, which further passes this request to the Controller.
- The Controller receives the input request coming from the user through the interface provided by the View.
- The Controller processes the request according to the Controller logic, and if no access to the Model is required, it moves to step 5.
- The Model is accessed and modified, if required. The Model then needs to notify all the associated Views about the modification.
- The View presents a user interface according to the modified or original Model, as the case may be.
- The View remains idle after the current interaction and waits for the next interaction to begin.
- This process is repeated again in the same way with every new request.

Summary

In this chapter, we learned how Web applications can be made more flexible and maintainable through the use of Java-based Web technologies such as Servlets and JSP, which are used to generate dynamic content in a portable and scalable manner. Next, we learned about the Web architecture and Web models. In addition, we learned that Web applications should be developed by using modular components. These components include Servlets, JSP pages, JavaBeans components, and Tag libraries containing custom tags. Depending on the composition of your development team, time constraints, and application architecture, the use of JSP pages and Servlets will differ.

Finally, we learned about the MVC architecture, which allows the separation of business logic, data, and presentation logic. In this architecture, the Controller, which is a Servlet, stores all the business logic. The View, normally a JSP page, contains all the code for presentation. The Model is implemented by using pure Java class or bean class. The different objects in the application, such as Controller, Model, and View, are reusable and make applications highly scalable.

In the next chapter, we learn about the Java Beans in detail.

Quick Revise

Q.1. What is a Web application?

Ans: A Web application can be defined as an application that can be accessed through the Internet using a Web browser and Web protocols such as HTTP.

Q.2. What does CGI stand for?

Ans: CGI stands for Common Gateway Interface.

Q.3. What is CGI?

Ans: CGI is an open standard abstraction between HTTP adapter (Web server) and an external application written by using any of the programming language such as C or C++.

Q.4. State one benefit of CGI.

Ans: CGI is not dependent on any particular Web server architecture.

Q.5. What lead to the introduction of Java Servlets?

Ans: Java Servlets came as an alternative to CGI, by providing high level, component-based, platform-independent, and server-independent standards for developing Web applications in Java.

Q.6. Why was JSP introduced?

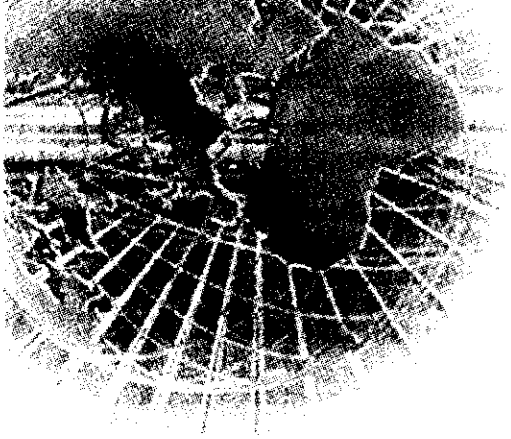
Ans: The requirement of tag based instructions has necessitated the introduction of JSP.

Q.7. What is MVC?

Ans: MVC stands for Model, View, Controller, and is an architectural pattern that divides the complex system in three important tiers/layers named Model, View, and Controller.

Q.8. What are Model-1 and Model-2 architectures?

Ans: Model-1 and Model-2 are the development models used to develop Web applications.



10

Working with JavaBeans

JavaBeans are Sun's answer to Microsoft's ActiveX controls. Beans are reusable code components written in Java that you can use in a variety of programming environments, including third-party programming environments. In fact, they work much like ActiveX controls, exposing properties, methods, and events to other code components. Using beans, you can create your own new Java "controls," such as buttons that turn colors when you click them or stock tickers that download information from the Internet.

JavaBeans is designed to be used in application builder tools, which are programming environments that allow you to configure beans. One such tool, the *beanbox*, comes with the Sun Bean Development Kit (BDK), which you can download and install from java.sun.com. We use the BDK in this chapter to construct and use beans as reusable code components, which you can be used in creating applications.

JavaBeans is very important for the Java users, as they allow you to build complex systems from software components. These components may be provided by you or supplied by one or more different vendors. JavaBeans defines an architecture that specifies how these building blocks can operate together.

If you want to understand the value of Beans in a better way, then consider the following example. In the construction of a system, hardware designers perform a very important function—bringing wide variety of components together to construct a system. The examples of simple building blocks are resistors, capacitors, and inductors. More advanced functionality is provided by the integrated circuits. All of these different parts can be reused. It is not necessary or possible to rebuild these capabilities each time a new system is needed. Moreover, in different types of circuits the same pieces can be used. This is possible only because the behavior of these components is understood and documented.

The software industry has also been seeking the benefits of reusability and interoperability of a component-based approach. A component architecture is needed that allows programs to be assembled from software building blocks, perhaps provided by different vendors to realize these benefits. It is also a must for a designer to select a component, to understand about it and how to make it available. Then, it must be easy to incorporate this functionality into existing code. Fortunately, such architecture is provided by JavaBeans.

This chapter explains various concepts related to JavaBeans, such as introspection, customizers. In addition, the API of JavaBeans is also discussed. You also learn the implementation of the bound and constrained properties in this chapter. Towards the end the chapter explores the Bean Info interface.

Introducing JavaBeans

A JavaBean is a software component that has been designed for reusability in different environments. Regarding the capability of a bean there is no restriction. It can perform different types of functions, such as obtaining an inventory value, forecasting the performance of a stock portfolio. A Bean may be designed to work autonomously on a user's workstation or to work in corporation with a set of other distributed components. An example of a Bean that can execute locally is the software that generates a pie chart from a set of data points. However, it is necessary that a Bean which provides a real-time price information from a stock or commodities exchange would need to work in corporation with other distributed software to obtain its data. The following are the key concepts of a Bean:

- Firstly, the builder tools discover a Bean's features (that is, its properties, methods, and events) by a process known as introspection. Beans support introspection in following two ways:
  - It provides support by adhering to specific rules, known as `design patterns`, at the time when naming Bean features. The `Introspector` class examines Beans for these design patterns to discover Bean features. This class relies on the `core reflection` API.
  - It provides support by explicitly providing property, method, and event information with a related `Bean Information` class, which implements the `BeanInfo` interface. Again, a `BeanInfo` class explicitly lists those Bean features that are to be exposed to application builder tools.
- Properties are the Bean's appearance and behavior characteristics, which can be changed at design time. Builder tools introspect on a Bean to discover its properties, and expose those properties for manipulation.
- Beans expose properties so that, they can be `customized` at design time. In this case, customization is supported in two ways—using property editors; by using more sophisticated Bean customizers.

- Beans use *events* in order to communicate with other Beans. A Bean that wants to receive events (i.e. a listener Bean) registers its interest with the Bean that fires the event (i.e. a source Bean). Builder tools can examine a Bean and determine which events the Bean can fire (send) and the one that it can handle (receive).
- Persistence enables Beans to save and restore their state. Once you've changed the Beans properties, you can save the state of the Bean and restore that Bean at a later time. Remember that, property changes remain intact. JavaBeans uses Java Object Serialization to support persistence.
- A Bean's methods are similar to the methods of Java methods. This method can be called from other Beans or a scripting environment. By default all public methods are exported.

Although Beans are designed to be understood by builder tools, all key APIs, including support for events, properties, and persistence, have been designed to be easily read as well as understood by programmers. The component technology is brought to the Java platform by the JavaBeans. You can create reusable, platform-independent components with the JavaBeans API. You can combine these components, using JavaBeans-compliant application builder tools, into applets, applications, or composite components. Beans are known as JavaBean components.

You require Beans Development Kit (BDK) for the development of JavaBeans. The BDK is available free on the Web. In addition to this, you will need the Java Development Kit (JDK).

After understanding the basic concepts of a Bean, let's now describe the advantages of JavaBeans.

Advantages of JavaBeans

The JavaBean technology plays a very important role. The following are the benefits that JavaBean technology provides for a component developer:

- All the benefits of Java's "write-once, run-anywhere" paradigm are present in the Bean.
- The properties, events, and methods of a Bean that are exposed to another application, such as a builder tool, can be controlled.
- To help configure a Bean, auxiliary software can be provided. This software is only needed when the design-time parameters for that component are being set. In the run-time environment, it does not need to be included.
- In the persistent storage, the configuration settings of a Bean can be saved and restored at a later time.
- You can also use Bean to register the receive events from other objects and generate events that are sent to other objects.

Introspection

Introspection is present at the core of JavaBeans. Introspection is the process by which a builder tool analyzes how a Bean works and also differentiates Beans from the typical Java classes. As Beans are coded with predefined patterns for their method signatures and class definitions, tools that recognize these patterns can determine properties and behavior of the Beans.

You can define introspection as the process of analyzing a Bean to determine its capabilities. As it allows another application, such as a design tool, to obtain information about a component, therefore, it is an essential feature of the Java Bean API. The JavaBean technology cannot operate without introspection.

There are two ways by which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an application builder tool. The simple naming conventions are used with the first method. These allow the introspection mechanisms to infer information about a Bean. In a second way, an additional class that extends the `BeanInfo` interface is provided that explicitly supplies this information.

Customizers

The role of the customizer, which is provided by a Bean developer, is to help another developer configure the Bean. A customizer can provide a step-by-step guide through the process that must be followed to use the component in a specific context. Online documentation can also be provided. To develop a customizer, a Bean developer has greater flexibility to differentiate his or her product in the marketplace.

After having a basic understandability about JavaBeans, let's explore the interfaces and classes of the java.beans package.

The JavaBeans API

Table 10.1 describes the functionality of the interfaces of the `java.beans` package:

Table 10.1: The Interfaces of the java.beans Package

Interface	Description
AppletInitializer	Works in collusion with the `java.beans.Beans.instantiate` method.
BeanInfo	Provides explicit information about the methods, properties, and events of a bean. A bean implementer who wishes to provide the explicit information about a bean may provide a BeanInfo class that implements this BeanInfo interface.
Customizer	Provides a complete custom GUI for customizing a target Java Bean.
DesignMode	Implemented by, or delegated from, instances of java.beans.beancontext.BeanContext, in order to propagate to its nested hierarchy of java.beans.beancontext.BeanContextChild instances, the current "designTime" property.
ExceptionListener	Notifies for the internal exceptions.
PropertyChangeListener	Gets fired whenever a bean changes a "bound" property.
PropertyEditor	Provides support for GUIs that want to allow users to edit a property value of a given type.
VetoableChangeListener	Gets fired whenever a bean changes a "constrained" property.
Visibility	Allows the Bean to run on servers where a GUI is not available.

Table 10.2 describes the functionality of the classes of the `java.beans` package:

Table 10.2: The Classes of the java.beans Package

Classes	Description
`BeanDescriptor`	Provides global information about a "bean", including its Java class, its `displayName`.
`Beans`	Provides some general purpose beans control methods.
`DefaultPersistenceDelegate`	Provides the concrete implementation of the abstract `PersistenceDelegate` class. It serves as a delegate, which is used by default for the classes for which no information is available.
`Encoder`	Used to create files, or streams that encode the state of a collection of JavaBeans in terms of their public APIs.
`EventHandler`	Provides support for dynamically generating event listeners whose methods execute a simple statement involving an incoming event object and a target object.
`EventSetDescriptor`	Describes a group of events that a given JavaBean fires.
`Expression`	Represents a primitive expression in which a single method is applied to a target and a set of arguments, to return a result.
`FeatureDescriptor`	Serves as the common `baseclass` for the `PropertyDescriptor`, `EventSetDescriptor`, and `MethodDescriptor` classes.
`IndexedPropertyChangeEvent`	Delivers the "`IndexedPropertyChange`" event whenever a component that conforms to the JavaBeans specification (a "bean") changes a bound indexed property.
`IndexedPropertyDescriptor`	Describes a property that acts like an array and has an indexed read and/or indexed write method to access specific elements of the array.

Table 10.2: The Classes of the java.beans Package

Classes	Description
`Introspector`	Provides a standard way for tools to learn about the properties, events, and methods supported by a target Java Bean.
`MethodDescriptor`	Describes a particular method that a Java Bean supports for external access from other components.
`ParameterDescriptor`	Allows bean implementers to provide additional information on each of their parameters, beyond the low level type information provided by the `java.lang.reflect.Method` class.
`PersistenceDelegate`	Takes the responsibility for expressing the state of an instance of a given class in terms of the methods in the class's public API.
`PropertyChangeEvent`	Delivers the "`PropertyChange`" event whenever a bean changes a "bound" or "constrained" property.
`PropertyChangeListenerProxy`	Extends the `EventListenerProxy` specifically for adding a named `PropertyChangeListener`.
`PropertyChangeSupport`	Used by beans that support bound properties.
`PropertyDescriptor`	Describes one property that a Java Bean exports via a pair of accessor methods.
`PropertyEditorManager`	Used to locate a property editor for any given type name.
`PropertyEditorSupport`	Help build property editors.
`SimpleBeanInfo`	Serves as a support class to make it easier for people to provide `BeanInfo` classes.
`Statement`	Represents a primitive statement in which a single method is applied to a target and a set of arguments.
`VetoableChangeListenerProxy`	Extends the `EventListenerProxy` specifically for associating a `VetoableChangeListener` with a "constrained" property.
`VetoableChangeSupport`	Used by beans that support constrained properties.
`XMLDecoder`	Used to read XML documents created using the `XMLEncoder` and is used just like the `ObjectInputStream`.
`XMLEncoder`	Serves as a complementary alternative to the `ObjectOutputStream` and can used to generate a textual representation of a Java Bean in the same way that the `ObjectOutputStream` can be used to create binary representation of Serializable objects.

In this chapter, you will create a number of JavaBeans—from simple ones to ones that support properties and methods and let users embed other controls inside them.

NOTE

The way you create beans varies a little among operating systems because the BDK differs a little among operating systems. In this chapter, we use the BDK for Windows.

Designing Programs Using JavaBeans

After you've downloaded and installed the Bean Development Kit (BDK), you can work with beans in the beanbox tool that comes with the BDK. Assuming that you've installed the BDK (the default path varies by system), you can open the beanbox; for example, in Windows, you use the run.bat file in the beanbox directory. When you run this batch file, the beanbox opens, as shown in Figure 10.1:

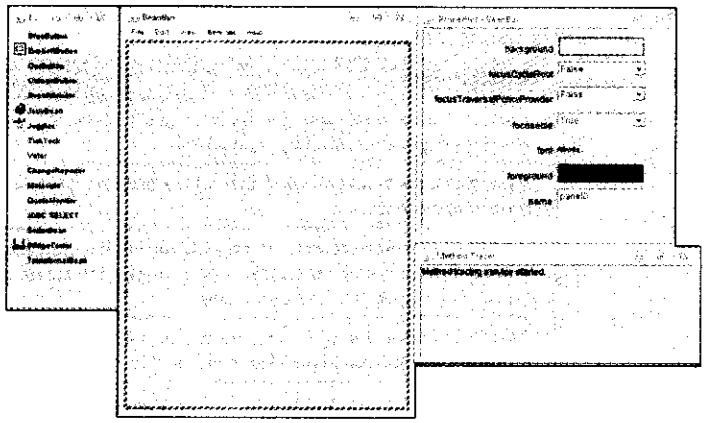


Figure 10.1: Displaying the BeanBox and other Related Windows

In Figure 10.1, you can see the available beans in the toolbox on the left. When you add beans to an application, they'll appear in the beanbox window next to the toolbox. You can also use the Properties window to set the properties that have been designed into a bean, and you can use the Method Tracer window to handle method execution. This section demonstrates how to work with beans.

Click the Juggler bean in the toolbox, which changes your cursor to a cross. Next, "draw" a Juggler bean in the beanbox by dragging the mouse (this bean displays the Java mascot image "juggling" coffee beans). Do the same for the bean named `OurButton`. Figure 10.2 shows the Juggler and button beans:

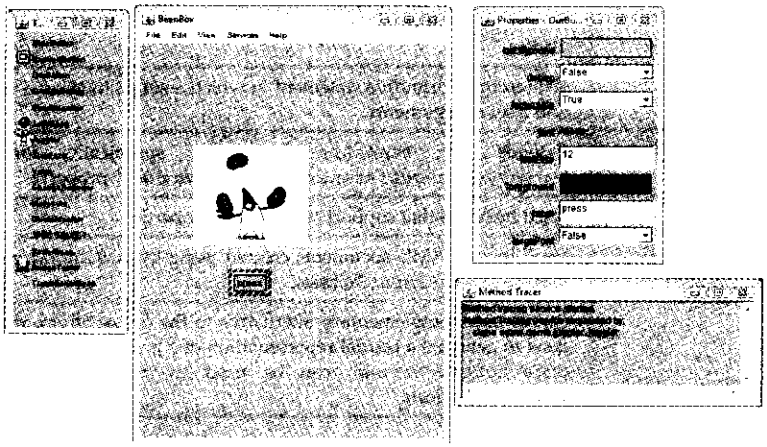


Figure 10.2: Creating Beans

You can connect the beans in a beanbox, thus creating a single application from several beans. For example, you can connect the button to the Juggler bean so when you click the button, the juggler will stop juggling. First, click the button bean in the beanbox. The Properties window will display the properties you can set for this bean, including its label—if you want to enter a new label for this button (such as "Click Me"), you can do it in the Properties window. To make the button do something when you click it, select the Edit→Events→action→ActionPerformed menu item from the BeanBox window. When you do, a red line appears between the mouse location and the button. Stretch that line to the Juggler bean now and click the Juggler bean, as shown in Figure 10.3:

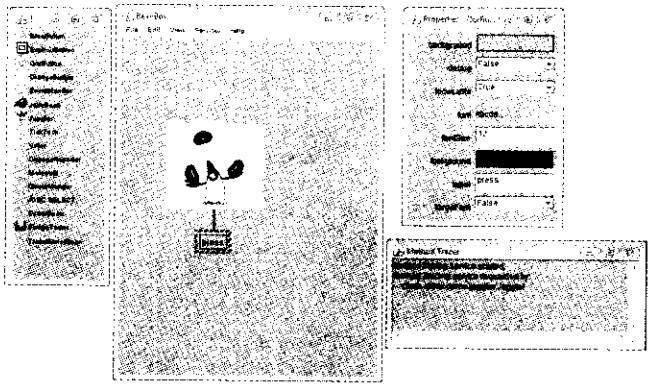


Figure 10.3: Connecting one Bean to Another

When you click the Juggler bean, the Event Target dialog box appears, as shown in Figure 10.4:

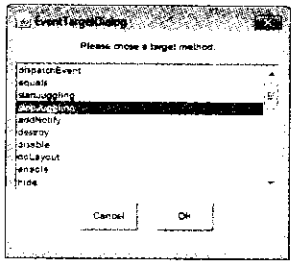


Figure 10.4: The EventTarget Dialog Box

Figure 10.4 shows the available methods you can call in the Juggler bean when the button is clicked. For this example, choose the `stopJuggling` method to make the juggler stop juggling; then, click on OK to close the Event Target dialog box.

Now, when you click the button in the beanbox, the juggler will stop juggling. After creating a simple application, let's discuss how to create you own JavaBean.

Creating a Java Bean

This section helps you to create a simple Java Bean to demonstrate how beans work and this bean is used in the rest of the chapter. This bean will just draw itself in red, and when you click it, it displays a count of the number of times it has been clicked.

We place this bean in the BDK's demo directory, so let's create a directory named bean in the beanbox\demo\sunw\demo folder and store the class files for this bean in that directory. Listing 10.1 provides the code for the bean.java file (you can find the bean.java file in the code\Java EE\Chapter 10\bean folder on CD):

Listing 10.1: The bean.java File

```
package sunw.demo.bean;
import java.awt.*;
import java.awt.event.*;
public class bean extends Canvas
{
  int count;
  public bean()
  {
        addMouseListener(new MouseAdapter()
        {
```

```
                public void mousePressed(MouseEvent me)
                {
                        clicked();
                }
        });
        count = 0;
        setSize(200, 100);
    }
    public void clicked()
    {
        count++;
        repaint();
    }
    public void paint(Graphics g)
    {
        Dimension dimension = getSize();
        int height = dimension.height;
        int width = dimension.width;
        g.setColor(new Color(255, 0, 0));
        g.fillRect(0, 0, --width, --height);
        g.setColor(new Color(0, 0, 0));
        g.drawString("Click count = " + count, 50, 50);
    }
}
```

In Listing 10.1, the `Canvas` class is used to draw the bean. Then, the mouse listener is added to the canvas to record mouse clicks, and the size of the canvas is set. You must remember that we have not created a bean yet; instead a component is created. Compile the bean.java file; the bean.class file is generated with the fully qualified path. After creating the component, let's now create the manifest file for the bean class.

Creating a Bean Manifest File

To use a bean, you have to store the class file(s) and manifest file in a JAR file. You use the manifest file to indicate which classes are beans. To show you how this works, let's create a manifest file, bean.mft, for the bean class created in the previous section. Listing 10.2 shows the code for the bean.mft file (you can find the bean.mft file in the code\Java EE\Chapter 10\bean folder on CD):

```
Name: sunw/demo/bean/bean.class
Java-Bean: True
```

Save the bean.mft file in the demo directory. To indicate that a class in a JAR file is a Java Bean, you have to set its `Java-Bean` attribute to True. The bean.class file is in the sunw.demo.bean package, which means it'll be stored in the JAR file as sunw/demo/bean/bean.class (like Unix, JAR files use forward slashes as directory separators). To indicate that this class file is a bean, the complete path has been specified in the bean.mft file.

After creating the manifest file, let's now generate the JAR file.

Creating a Bean JAR File

In this subsection, let's create the JAR file for the bean. Execute the following command to create the bean.jar file in the jars folder located within the beans directory:

```
C:\...\demo>jar cfm ..\jars\bean.jar bean.mft sunw\demo\bean\*.class
```

This creates the new JAR file for this bean, bean.jar, and stores it in the demo\jars directory, which is where the beanbox will look for it. Now, after creating the bean, let's discuss how to use the bean.

Using the New Bean

We've developed a new JavaBean, bean in the previous sections and installed it in the demo\jars directory. When you open the beanbox, you'll see this bean (which we've just called "bean") listed in the toolbox, as shown in Figure 10.5:

Figure 10.5: Displaying the New Bean

You can draw a bean of this new type in the beanbox. This new bean is already active in the beanbox, as you can see by clicking it, because it shows the number of times it's been clicked. When you draw the bean from the toolbox, the bean appears in the BeanBox window, as shown in Figure 10.6:

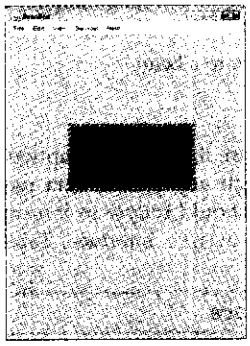


Figure 10.6: Displaying the bean in the BeanBox Window

Click the bean, shown in Figure 10.6, the value of the `count` variable increases automatically. Now, let's discuss how to add controls to beans.

Adding Controls to Beans

You can add Java controls such as buttons to your beans by ensuring that you base your bean on a class that has a container, such as the `Panel` class. Here is an example in which we add a button to a bean and have the bean display the number of times it has been clicked. Let's start by basing this bean (button) on the `Panel` class and adding it to the sunw.demo.button package. In other words, let's store the class files of the button bean in the demo\sunw\button directory. Listing 10.2 creates the panel, sizes it, and adds a button to it (you can find the button.java file in the code\Java EE\Chapter 10\bean folder on CD):

Listing 10.2: The button.java File

```
package sunw.demo.button;
import java.awt.*;
import java.awt.event.*;
public class button extends Panel implements ActionListener
{
int count;
Button button1;
public button()
{
count = 0;
setSize(200, 100);
```

```
button1 = new Button("Click me");
button1.addActionListener(this);
add(button1);
}
public void actionPerformed(ActionEvent e)
{
count++;
repaint();
}
public void paint(Graphics g)
{
Dimension dimension = getSize();
int h = dimension.height;
int w = dimension.width;
g.setColor(new Color(255, 0, 0));
g.fillRect(0, 0, w-1, h-1);
g.setColor(new Color(0, 0, 0));
g.drawString("Click count = " + count, 50, 50);
}
}
```

Listing 10.2 shows the code that when the button is clicked, the value of the count variable increases.

Let's now discuss how to provide properties to a bean.

Giving a Bean Properties

The properties of a bean allow you configure it, setting its caption, size, color, and any other aspect of the bean for which properties are defined. Although any public data member of the bean class can be treated as a property, there's a formal procedure you should follow to inform the Java framework about the properties of your beans, that is, implementing the `BeanInfo` interface. Table 10.3 describes the fields of the `BeanInfo` interface:

Table 10.3: Fields of the BeanInfo Interface

Field	Description
static int ICON_COLOR_16x16	Indicates a 16x16 color icon.
static int ICON_COLOR_32x32	Indicates a 32x32 color icon.
static int ICON_MONO_16x16	Indicates a 16x16 monochrome icon.
static int ICON_MONO_32x32	Indicates a 32x32 monochrome icon

Table 10.4 describes the methods of the BeanInfo interface:

Table 10.4: Methods of the BeanInfo Interface

Method	Description
BeanInfo[] getAdditionalBeanInfo()	Allows a `BeanInfo` object to return an arbitrary collection of other `BeanInfo` objects.
BeanDescriptor getBeanDescriptor()	Gets the bean's bean descriptor.
int getDefaultEventIndex()	Gets the default event. A bean may have a default event (the event that will most commonly be used).
int getDefaultPropertyIndex()	Gets the default property index. A bean may have a default property (the property that will most commonly be initially chosen for update).
EventSetDescriptor[] getEventSetDescriptors()	Gets the bean's event set descriptors.

Table 10.4: Methods of the BeanInfo Interface

Method	Description
Image getIcon(int iconKind)	Gets an image object that can be used to represent the bean in toolboxes, toolbars, and so on.
MethodDescriptor[] getMethodDescriptors()	Gets the bean's method descriptors.
PropertyDescriptor[] getPropertyDescriptors()	Gets the bean's property descriptors.

In fact, most beans don't implement the `BeanInfo` interface directly. Instead, they extend the `SimpleBeanInfo` class, which implements `BeanInfo`. The following is the inheritance diagram for `SimpleBeanInfo`:

```
java.lang.Object
|____java.beans.SimpleBeanInfo
```

You will find the constructor of the `SimpleBeanInfo` class in Table 10.5:

Table 10.5: The Constructor of the SimpleBeanInfo Class

Constructor	Description
SimpleBeanInfo()	Constructs a SimpleBeanInfo object.

Table 10.6 describes the methods of the SimpleBeanInfo class:

Table 10.6: Methods of the SimpleBeanInfo Class

Method	Description
BeanInfo[] getAdditionalBeanInfo()	Implemented to indicate that there are no other relevant BeanInfo objects.
BeanDescriptor getBeanDescriptor()	Returns overall information about the bean, such as its displayName, and its customizer. Returns null if the information should be obtained by automatic analysis.
int getDefaultEventIndex()	Returns index of default event in the EventSetDescriptor array returned by getEventSetDescriptors.
int getDefaultPropertyIndex()	Returns index of default property in the PropertyDescriptor array returned by getPropertyDescriptors.
EventSetDescriptor[] getEventSetDescriptors()	Returns an array of EventSetDescriptors describing the kinds of events fired by this bean.
Image getIcon(int iconKind)	Returns an image object representing the requested icon. May return null if no suitable icon is available.
MethodDescriptor[] getMethodDescriptors()	Returns an array of MethodDescriptors describing the externally visible methods supported by this bean.
PropertyDescriptor[] getPropertyDescriptors()	Returns an array of PropertyDescriptors describing the editable properties supported by this bean
Image loadImage(String resourceName)	Returns an image object. It returns null if the load failed. This utility method used to help in loading icon images.

To actually describe a property, you use the `PropertyDescriptor` class, which in turn is derived from the `FeatureDescriptor` class. The following is the inheritance diagram for the `FeatureDescriptor` class:

```
java.lang.Object
|____java.beans.FeatureDescriptor
```

Table 10.7 describes the constructor of the `FeatureDescriptor` class:

Table 10.7: The Constructor of the FeatureDescriptor class.

Constructor	Description
FeatureDescriptor()	Constructs a feature descriptor.

Table 10.8 describes the methods of the FeatureDescriptor class:

Table 10.8: Methods of the FeatureDescriptor Class

Method	Description
Enumeration attributeNames()	Returns an enumeration of the locale-independent names of this feature.
String getDisplayName()	Returns the localized display name of this feature.
String getName()	Returns the programmatic name of this feature.
String getShortDescription()	Returns the short description of this feature.
Object getValue(String attributeName)	Returns a named attribute with this feature.
boolean isExpert()	Returns True for features that are intended for expert users.
boolean isHidden()	Returns True for features that are intended only for tool use.
boolean isPreferred()	Returns True for features that are particularly important for presenting to people.
void setDisplayName(String displayName)	Sets the localized display name of this feature.
void setExpert(boolean expert)	Sets the expert flag for features that are intended for expert users.
void setHidden(boolean hidden)	Sets the hidden flag for features intended only for tool use.
void setName(String name)	Sets the name of this feature.
void setPreferred(boolean preferred)	Sets the preferred flag, used to identify features that are particularly important for presenting to people.
void setShortDescription(String text)	You can associate a short descriptive string with a feature.
void setValue(String attributeName, Object value)	You can associate a named attribute with this feature.

The following is the inheritance diagram for the `PropertyDescriptor` class:

```
java.lang.Object
    |____java.beans.FeatureDescriptor
        |____java.beans.PropertyDescriptor
```

Table 10.9 describes the constructors of the `PropertyDescriptor` class:

Table 10.9: The Constructors of the PropertyDescriptor Class

Constructor	Description
PropertyDescriptor(String propertyName, Class beanClass)	Constructs a property descriptor.
PropertyDescriptor(String propertyName, Class beanClass, String getterName, String setterName)	Takes the name of a simple property as well as method names for reading and writing the property.
PropertyDescriptor(String propertyName, Method getter, Method setter)	Takes the name of a simple property as well as `Method` objects for reading and writing the property.

Table 10.10 describes the methods of the PropertyDescriptor class:

Table 10.10: Methods of the PropertyDescriptor Class

Method	Description
Class getPropertyEditorClass()	Returns any explicit `PropertyEditor` class that has been registered for this property.
Class getPropertyType()	Returns the `Class` object for the property.
Method getReadMethod()	Returns the method that should be used to read the property value.
Method getWriteMethod()	Returns the method that should be used to write the property value.
boolean isBound()	Returns true if this is a bound property. Updates to bound properties cause a `PropertyChange` event to be fired when the property is changed.
boolean isConstrained()	Returns true if this is a constrained property. Attempted updates to constrained properties cause a `VetoableChange` event to be fired when the property is changed.
void setBound(boolean bound)	Sets the bound property of this object. Updates to bound properties cause a `PropertyChange` event to be fired when the property is changed.
void setConstrained(boolean constrained)	Sets the constrained property of this object. Attempted updates to constrained properties cause a `VetoableChange` event to be fired when the property is changed.
void setPropertyEditorClass(Class propertyEditorClass)	Returns the class for the desired PropertyEditor. Normally, property editors will be found using the property editor manager.
void setReadMethod(Method getter)	Sets the method that is used to read the property value.
void setWriteMethod(Method setter)	Sets the method that is used to write the property value.

Let's look at an example that implements a property in a JavaBean. In this case, We add a property named `filled` to the click-counting bean developed in this chapter. This property is a boolean property that, when True, makes sure the bean will be filled in with color. Let's consider this new bean as `Bean2`.

To keep track of the new `filled` property, we need add a private boolean variable of the property (filled) name to the `Bean2` class. Listing 10.3 shows the code for the Bean2.java file (you can find the Bean2.java file in the code\Java EE\Chapter 10\bean folder on CD):

Listing 10.3: The Bean2.java File

```
package sunw.demo.bean2;
import java.awt.*;
import java.awt.event.*;
public class Bean2 extends Canvas {
private boolean filled;
int count;
public Bean2() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
clicked();
}
});
count = 0;
filled = false;
setSize(200, 100);
}
public void clicked() {
count++;
repaint();
}
public boolean getfilled() {
return filled;
}
```

```
public void setfilled(boolean flag) {
this.filled = flag;
repaint();
}
public void paint(Graphics g) {
Dimension dimension = getSize();
int height = dimension.height;
int width = dimension.width;
if(filled) {
g.setColor(new Color(255, 0, 0));
g.fillRect(0, 0, --width, --height);
}
g.setColor(new Color(0, 0, 0));
g.drawString("Click count = " + count, 50, 50);
}
}
```

When you implement a property, Java will look for two methods: `getPropertyName()` and `setPropertyName()`, where `PropertyName` is the name of the property. The `get` method returns the current value of the property, which can be any supported type, and the `set` method takes an argument of that type, which you're supposed to set the property to. Listing 10.3 shows the implementation of the `getfilled` and `setfilled` methods.

Now, let's create a new class, `Bean2BeanInfo`, which returns information about this new bean property. This class will be in the same package as the bean itself, but it's based on the `SimpleBeanInfo` class. Listing 10.4 provides the code for the `Bean2BeanInfo.java` file (you can find the Bean2BeanInfo.java file in the code\Java EE\Chapter 10\bean folder on CD):

Listing 10.4: The Bean2BeanInfo.java File

```
package sunw.demo.bean2;
import java.beans.*;
public class Bean2BeanInfo extends SimpleBeanInfo {
public PropertyDescriptor[] getPropertyDescriptors() {
try {
PropertyDescriptor filled = new
PropertyDescriptor("filled", Bean2.class);
PropertyDescriptor propertydescriptor[] = {filled};
return propertydescriptor;
}
catch(Exception e) {}
return null;
}
}
```

Listing 10.4 implements the `getPropertyDescriptors()` method, which returns an array of `PropertyDescriptor` objects. Each `PropertyDescriptor` object holds the name of a property and points to the class that supports that property. After compiling this new class (Bean2BeanInfo.java), save the Bean2BeanInfo.class in the directory `demo\sunw\demo\bean2`, along with the Bean2.class file. Let's create a new manifest file that includes the Bean2 and `Bean2BeanInfo` classes. Listing 10.5 shows the code for the Bean2.mft file (you can find the Bean2.mft file in the code\Java EE\Chapter 10\bean folder on CD):

Listing 10.5: The Bean2.mft File

```
Name: sunw/demo/bean2/Bean2BeanInfo.class
Java-Bean: True
Name: sunw/demo/bean2/Bean2.class
Java-Bean: True
```

Save this new manifest file in the demo directory. Finally, create the new Bean2.jar file by executing the following command:

```
jar cfm ..\jars\bean2.jar Bean2.mft sunw\demo\bean2\*.class
```

Now when you run the beanbox and add a new `Bean2` bean to the beanbox, the new `filled` property will appear in the Properties window. Setting `filled` to True causes the bean to be filled with color.

After understanding how to set the property of a bean by using the PropertyDescriptor class, let's now discuss how to implement simple and indexed properties for a bean.

Design Patterns for Properties

The subset of a Bean's state is a property. The behavior and appearance of that component is determined by the values assigned to the properties. A property is set by using the `setter()` method. A property is obtained by using the `getter()` method. There are two types of properties. These properties are simple and indexed. Let's discuss both of them in detail.

Simple Properties

The characteristic of the simple property is that it has a single value. It can be identified by the following design patterns, where `N` is the name of the property and `T` is its type:

```
public T getN();
public void setN(T arg);
```

To access its values, a read/write property has both of these methods. A read only property has only a `get()` method and a write-only property has only a `set()` method. The following are three read/write simple properties along with their `getter()` and `setter()` methods.

```
private double depth, height, width;
public double getDepth() {
 return depth;
}
public void setDepth(double dbl) {
 depth = dbl;
}
public double getHeight() {
 return height;
}
public void setHeight(double hdbl) {
 height = hdbl;
}
public double getWidth() {
 return width;
}
public void setWidth(double wdbl) {
 width = wdbl;
}
```

Adding a Color Property to SimpleBean

Listing 10.6 provides the code to add a Color property to a bean (you can find the SimpleBeanapp.java file in the code\Java EE\Chapter 10\bean folder):

Listing 10.6: The SimpleBeanapp.java File

```
import java.awt.*;
import java.io.Serializable;
public class SimpleBeanapp extends Canvas implements Serializable {
    private Color clr = Color.green;
    public Color getColor() {
        return clr;
    }
    public void setColor(Color newclr) {
        clr = newclr;
        repaint();
    }
```

```
    public void paint(Graphics g) {
        g.setColor(clr);
        g.fillRect(20, 5, 20, 30);
    }

    public SimpleBeanapp(){
        setSize(60,40);
        setBackground(Color.red);
    }
}
```

Compile the Bean, load it in the ToolBox, and create an instance in the `BeanBox`. The following results are explained below:

1. `SimpleBean` will be displayed with a green centered rectangle.
2. As a result, the Properties sheet will contain a new Color property. At the same time, the introspection mechanism will also search for a Color property editor. A Color property editor is one of the default editors supplied with the `BeanBox`, and is assigned as `SimpleBean`'s Color property editor. To run this editor, you have to click on the Color property in the Properties sheet.

After understanding the simple properties, let's now discuss the indexed properties of a bean.

Indexed Properties

On the other hand, indexed properties consist of multiple values. By the following design pattern it can be identified. In this design pattern, `N` is the name of the property and `T` is its type, as shown in the following code snippet:

```
public T getN(int index);
public void setN(int index, T value);
public T[] getN();
public void setN(T values[]);
```

In the following code snippet an indexed property called data is set along with the `getter()` and `setter()` methods:

```
private double data[];
public double getData (int ind) {
 return data[ind];
}
public void setData(int ind, double dblvalues)  {
 data[ind] =  dblvalue;
}
public double [] getData()  {
 return data;
}
public void setData(double[] dblvalue)  {
data = new double[dblvalue.length];
System.arraycopy(dblvalues, 0, data, 0, 0, dblvalues.length);
}
```

After understanding how to set simple as well as indexed properties, let's now discuss how to use the BeanInfo interface.

Using the BeanInfo Interface

The `BeanInfo` interface enables you to explicitly control the information available to the user of a Bean. The `BeanInfo` interface defines several methods. The `BeanInfo` interface declares methods that return arrays of the above descriptors. To discuss it further, lets introduce the following code snippet:

```
PropertyDescriptor[] getPropertyDescriptors()
EventSetDescriptor[] getEventSetDescriptors()
MethodDescriptor[] getMethodDescriptors()
```

In the preceding code snippet, the role of the return array of objects is to provide information about the properties, events, and methods of a Bean. In this regard, the `PropertyDescriptor`, `EventSetDescriptor`, and `MethodDescriptor` classes are well defined within the `java.beans` package, and the methods of a Bean describe the indicated elements. By implementing these methods, a developer can designate exactly what is presented to a user, bypassing introspection based on design patterns.

While creating a class that implements the `BeanInfo` interface, you have to remember that you must call that class `bnameBeanInfo`, where `bname` is the name of the Bean. For example, if the Bean is called as `MyBean`, then the information class must be called `MyBeanBeanInfo`.

Now, the question is how to simplify the use of `BeanInfo`? In this package, to simplify the `BeanInfo`, JavaBeans supplies the `SimpleBeanInfo` class. It provides default implementations of the `BeanInfo` interface, including the three methods just described. However, you are free to extend this class and override one or more of the methods to explicitly control what aspects of a Bean are exposed. But, if you don't override a method, then the role of design-pattern introspection will come and it will be used. For example, if you don't override `getPropertyDescriptors()`, then design patterns are used to discover a Bean's properties.

After understanding the implementation of the BeanInfo interface, let's now discuss how to create the bound properties for a bean.

Creating Bound Properties

You can also create *bound* properties in JavaBeans. Bound properties generate an event when their values change. This event is of type `PropertyChangeEvent` and is sent to all registered event listeners of this type. To make a property a bound property, use the `setBound` method, as shown in the following code snippet:

```
package sunw.demo.bean2;
import java.beans.*;
public class Bean2BeanInfo extends SimpleBeanInfo {
public PropertyDescriptor[] getPropertyDescriptors() {
try {
PropertyDescriptor filled = new
PropertyDescriptor("filled", Bean2.class);
filled.setBound(true);
PropertyDescriptor propertydescriptor[] = {filled};
return propertydescriptor;
}
catch(Exception e) {}
return null;
...
}
}
```

Let's now discuss how to declare a method in a JavaBean, which can be called by other beans.

Giving a Bean Methods

You can formally describe methods of JavaBeans to the Java framework by using the `MethodDescriptor` class. The following is the inheritance diagram of this class:

```
java.lang.Object
|____java.beans.FeatureDescriptor
|____java.beans.MethodDescriptor
```

Table 10.11 describes the constructors of the `MethodDescriptor` class:

Table 10.11: Constructors of the MethodDescriptor Class

Constructor	Description
MethodDescriptor(Method method)	Constructs a method descriptor.
MethodDescriptor(Method method, ParameterDescriptor[] parameterDescriptors)	Constructs a method descriptor from a method, providing descriptive information for each of the method's parameters.

Table 10.12 describes the methods of the MethodDescriptor class:

Table 10.12: Methods of the MethodDescriptor Class

Method	Description
Method getMethod()	Returns the method that this method descriptor encapsulates.
ParameterDescriptor[]getParameterDescriptors()	Returns the parameter descriptor for each of the parameters of this method descriptor's methods.

Many beans, however, don't use the `MethodDescriptor` objects, because any public bean method is accessible from other beans. Let's consider an example in which we create a new bean based on the ones has been developed in this chapter. This bean, `Bean3`, will count the number of times it's been clicked and will also support a method named `increment` that, when invoked, will increment the click count.

Listing 10.7 provides the code for `Bean3`, including the public `increment` method (You can find the Bean3.java file in the code\Java EE\Chapter 10\bean folder on CD):

Listing 10.7: The Bean3 java File

```
package sunw.demo.bean3;
import java.awt.*;
import java.awt.event.*;
public class Bean3 extends Canvas {
int count;
public Bean3() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
clicked();
}
});
count = 0;
setSize(200, 100);
}
public void clicked() {
count++;
repaint();
}
public void increment() {
count++;
repaint();
}
public void paint(Graphics g) {
Dimension dimension = getSize();
int height = dimension.height;
int width = dimension.width;
g.setColor(new Color(255, 0, 0));
g.fillRect(0, 0, --width, --height);
g.setColor(new Color(0, 0, 0));
g.drawString("Click count = " + count, 50, 50);
}
}
```

After creating `Bean3`, you need to create the Bean3.mft file and generate the JAR file for bean3. You can find the Bean3.mft file in the code\Java EE\Chapter 10\bean folder on CD. Then, add the bean to the beanbox, we can connect other beans to the `increment` method. We have connected a button to that method, and each time the button is clicked, the click count in the `Bean3` bean is incremented and displayed.

Let's now discuss how to provide icons to a bean.

Giving a Bean an Icon

You may have noticed that some beans have icons in the beanbox. You can add your own icons to your beans by adding the `getIcon` method to the `BeanInfo` class. The following code snippet demonstrates how to implement this method and handle all possibilities – monochrome or color icons of either 16×16 or 32×32 pixels:

```
public java.awt.Image getIcon(int iconKind) {
if (iconKind == BeanInfo.ICON_MONO_16x16 ||
iconKind == BeanInfo.ICON_COLOR_16x16 ) {
java.awt.Image image = loadImage("Icon16.gif");
return image;
}
if (iconKind == BeanInfo.ICON_MONO_32x32 ||
iconKind == BeanInfo.ICON_COLOR_32x32 ) {
java.awt.Image image = loadImage("Icon32.gif");
return image;
}
return null;
}
```

Let's now discuss how to create a BeanInfo class.

Creating a BeanInfo Class

This section of the chapter uses the `ExplicitButtonBeanInfo` demo class to illustrate creating a `BeanInfo` class. There are different steps for creating a `BeanInfo` class. Let us discuss the general steps to make a `BeanInfo` class:

1. As a first step, you have to name your `BeanInfo` class. You must append the string "BeanInfo" to the target class name. If the target class name is `ExplicitButton`, then its associated Bean information class must be named `ExplicitButtonBeanInfo`
2. The second step to make a `BeanInfo` class is `Subclass SimpleBeanInfo`. This is a convenience class that implements `BeanInfo` methods to return null, or an equivalent value.
3. The third step being is to extend the SimpleBeanInfo class, as shown in the following code snippet:
   ```
   public class ExplicitButtonBeanInfo extends SimpleBeanInfo {}
   ```

Using `SimpleBeanInfo` saves you from implementing all the `BeanInfo` methods; you only have to override the methods that you need.

Override the appropriate methods to return the properties, methods, or events that you want to expose. `ExplicitButtonBeanInfo` overrides the `getPropertyDescriptors()` method to return four properties, as shown in the following code snippet:

```
public PropertyDescriptor[] getPropertyDescriptors() {
    try {
        PropertyDescriptor background =
            new PropertyDescriptor("background", beanClass);
        PropertyDescriptor foreground =
            new PropertyDescriptor("foreground", beanClass);
        PropertyDescriptor font =
            new PropertyDescriptor("font", beanClass);
        PropertyDescriptor label =
            new PropertyDescriptor("label", beanClass);
        background.setBound(true);
        foreground.setBound(true);
        font.setBound(true);
        label.setBound(true);
        PropertyDescriptor rv[] =
            {background, foreground, font, label};
        return rv;
    }
catch (IntrospectionException e) {
        throw new Error(e.toString());
```

```
        }
    }
```

In the preceding code snippet, there are two very important things to be noted. These are:

- If you leave a descriptor out, then as a result, that property, event or method will *not* be exposed. In other words, you can selectively expose properties, events, or methods by leaving out those, which you don't want to be exposed.
- If a getter (for example, `getMethodDescriptor()`) method returns null for a feature, then you have to use a low-level reflection for that feature. This means that you can explicitly specify properties, and let low-level reflection discover the methods. If you don't override the default `SimpleBeanInfo` method, which returns null, then a low-level reflection will be used for that feature.

4. As a next step, optionally associate an icon with the target Bean, as shown in the following code snippet:

```
public java.awt.Image getIcon(int icontype) {
    if (icontype == BeanInfo.ICON_MONO_16x16 ||
        icontype == BeanInfo.ICON_COLOR_16x16 ) {
        java.awt.Image img = loadImage("tom.gif");
        return img;
    }
    if (icontype == BeanInfo.ICON_MONO_32x32 ||
        icontype == BeanInfo.ICON_COLOR_32x32 ) {
        java.awt.Image img = loadImage("ExplicitButtonIcon32.gif");
        return img;
    }
    return null;
}
```

The `BeanBox`, then, displays this icon next to the Bean name in the ToolBox. You can expect the builder tools to do the same. Specify the target `Bean` class, and, if the Bean has a customizer, specify it also by using the following code snippet:

```
public BeanDescriptor getBeanDescriptor() {
    return new BeanDescriptor(beanClass, customizerClass);
}
...

resulting,private final static Class beanClass = ExplicitButton.class;
private final static Class customizerClass = OurButtonCustomizer.class;
```

In the above steps, keep the `BeanInfo` class in the same directory as its target class. Firstly, the `BeanBox` searches for a target Bean's `BeanInfo` class in the target Bean's package path. If, as a result of this search, no `BeanInfo` is found then, as a next step, the Bean information package search path (maintained by the `Introspector`) is searched. The default Bean information search path is `sun.beans.infos`. Again, if no `BeanInfo` class is found, then low-level reflection is used to discover a Bean's features.

Let's now discuss how to use the bound and constrained properties.

Bound and Constrained Properties

A Bean which contains a bound property generates an event when the property is changed. Here, the event is of type `PropertyChangeEvent`. And, it is sent to objects that previously registered an interest in receiving such notifications. A class that handles this event must implement the `PropertyChange Listener` interface.

The JavaBeans model supports two variations on standard properties—first is bound properties and the second is called constrained properties. The role of bound properties is to support the registration and notification of "interested parties", whenever the value of the property changes. And the role of constrained properties is to take this notification model one step further by allowing the notified party to exercise a veto in order to prevent the property change. Unlike event handling, most of the functionality required to support bound and constrained properties is handled with the JavaBeans framework.

Bound properties are useful for Beans that want to allow instances of the same Bean class or some other Bean class to monitor a property value and subsequently change their values accordingly (i.e. to match the "trend setting" Bean). Lets cite an example. Consider a GUI Bean that wants to allow other GUI Beans to monitor a change in its background color to update their backgrounds accordingly.

However, note that, implementation of a `bound` property is quite easy because of the user-friendly underlying framework provided by the JavaBeans architecture. `Bean` class supports a bound property. In order to support a `bound` property, the `Bean` class must instantiate an object in the JavaBeans framework, which provides the bulk of this functionality, and implement registration and unregistration methods that simply invoke the appropriate methods in the JavaBeans framework. The following code snippet shows the use of the `addPropertyChangeListener()` and `removePropertyChangeListener()` methods:

```
private PropertyChangeSupport changes = new PropertyChangeSupport(this);
public void addPropertyChangeListener(PropertyChangeListener p)
{
  changes.addPropertyChangeListener(p);
}
public void removePropertyChangeListener(PropertyChangeListener p)
{
  changes.removePropertyChangeListener(p);
}
```

Each bound property must then invoke the `firePropertyChange()` method from its `set()` method, as shown in the following code snippet:

```
public void setMood(int mood)
{
  int old = this.mood;
  this.mood = mood;
  repaint();
  changes.firePropertyChange("mood",
    new Integer(old), new Integer(mood));
}
```

In the preceding code snippet, the `PropertyChangeSupport` object is used to handle the notification of all registered targets. You have to note that `PropertyChangeSupport` provides a general-purpose functionality following a prescribed protocol. Specifically, the method invocation for the `firePropertyChange()` method must provide the property name, as well as the old and new values, which are passed along to notified targets.

Moreover, the listener (i.e. target object) must provide a `propertyChange()` method to receive the property-related notifications, as shown in the following code snippet:

```
public void propertyChange(PropertyChangeEvent e)
{
  // ...
}
```

In the preceding code snippet, constrained properties add the functionality, which might be the notified listener that can object to the property change and can execute a veto. To support constrained properties, the `Bean` class must instantiate the JavaBeans object that provides this service, namely, `VetoableChangeSupport`, and implement the corresponding registration-related methods, as shown in the following code snippet:

```
private VetoableChangeSupport veto = new VetoableChangeSupport(this);
public void addVetoableChangeListener(VetoableChangeListener vetolisten)
{
  veto.addVetoableChangeListener(vetolisten);
}
public void removeVetoableChangeListener(VetoableChangeListener vetolisten)
{
  veto.removeVetoableChangeListener(vetolisten);
}
```

You have to note that a Bean could provide one or more bound and constrained properties. In this case, it must instantiate both the `PropertyChangeSupport` and `VetoableChangeSupport` classes and provide both sets

of registration-related methods. In addition, constrained properties tend to be bound, but that is not a strict requirement. The `set()` method for bound-constrained properties is slightly more complicated. The implementation of the `set()` method is provided in the following code snippet:

```
public void setMood(int md)
 throws PropertyVetoException
{
  vetoes.fireVetoableChange("mood",
        new Integer(this.md), new Integer(md));
  int old = this.md;
  this.md = md;
  repaint();
  changes.firePropertyChange("mood",
   new Integer(old), new Integer(md));
}
```

Specifically, the `set()` method must accommodate the `PropertyVetoException` exception. In this regard, the sequence of operations is:

- Fire the vetoable change notification.
- Update the appropriate state variables.
- Fire the standard property change notification, if bound.

A veto-interested target object must implement the `vetoableChange()` method, as shown in the following code snippet:

```
public void vetoableChange(PropertyChangeEvent e)
   throws PropertyVetoException
{
  ...
}
```

It exercises a veto by the following:

1. Including a throws clause for `PropertyVetoException` and
2. Raising the exception (throw new `PropertyVetoException()`), as appropriate.

The JavaBeans framework receives the propagated exception. And in turn, JavaBeans framework propagates it to `setMood()` via the `fireVetoableChange()` invocation. Note that `setMood()` is organized such that in the event of a veto, (1) it does not handle the exception (instead, returning and propagating the exception), hence, (2) it does not complete the update to state variables or the property notification for bound-property listeners.

Now, it is very much clear that the JavaBeans framework supports a lot of property-related functionality and requires minimal work from the Bean implementation. Let's now discuss the concept of persistence.

Persistence

To save the current position of a Bean and also its properties and instance variables, the role of Persistence comes into play. Therefore, Persistence is the ability to save the current state of a Bean, which also includes the values of Bean's properties and instance variables to nonvolatile storage and thus to retrieve them at a later stage. The object serialization capabilities provided by the Java class libraries are used to provide persistence for Beans.

Among the different methods available, the easiest way to serialize a Bean is to implement the `java.io.Serializable` interface, which is nothing but a marker interface. Implementing the `java.io.Serializable` interface makes serialization automatic. Your Bean need not take any other action in this regard. Moreover, remember that, automatic serialization can also be inherited. Therefore, if any superclass of a Bean implements the `java.io.Serializable` interface, then automatic serialization is obtained. But, the only restriction is that any class that implements the `java.io.Serializable` interface must supply a parameter less constructor.

When the user is using automatic serialization, they can selectively prevent a field from being saved through the use of the transient keyword. As a result, data members of a Bean specified as transient will not be serialized.

If a Bean does not implement the `java.io.Serializable` interface, then you must provide serialization yourself. This can be done, such as, by implementing the `java.io.Externalizable` interface. Otherwise, containers cannot save the configuration of your component.

Let's now discuss the use of JavaBeans with JSP pages.

Using Beans with JSP pages

The role of JavaServer Pages technology is also to provide direct supports using JavaBeans components with JSP language elements. In this case, you can easily create and initialize beans and get and set the values of their properties.

You will understand the implementation of JavaBeans with JSP pages after having the knowledge of creating JSP pages. You will understand how to work with JavaBeans and JSP pages in the *Chapter 12, Working with JSP.*

With this we have come to the end of the chapter. Now, let's quickly summarize what you have learned in this chapter.

Summary

The chapter has described various concepts related to JavaBeans, such as customizers, introspection. You have learned how to create a bean, a manisfest file for the bean, and deploy the bean in a JAR file. We have described how to add properties to a bean. Toward the end, the chapter has discussed the role of JavaBeans in the JSP pages to store the data.

Quick Revise

Q.1 What do you mean by JavaBeans?

Ans: A JavaBean is a software component that has been designed for reusability in different environments. Regarding the capability of a bean there is no restriction.

Q2. What do you mean by introspection?

Ans: Introspection is the process by which a builder tool analyzes how a Bean works and also differentiates Beans from the typical Java classes.

Q3. What is the role of customizers?

Ans: The role of the customizer, which is provided by a Bean developer, is to help another developer configure the Bean.

Q4. State the constructor of the SimpleBeanInfo class.

Ans: SimpleBeanInfo()

Q5. List the interfaces available in the java.beans package.

Ans: The interfaces within the java.beans package are:

- AppletInitializer
- BeanInfo
- Customizer
- DesignMode
- ExceptionListener
- PropertyChangeListener
- PropertyEditor
- VetoableChangeListener
- Visibility

Q6. List the classes within the java.beans package.

Ans: The classes of the java.beans package are:

- BeanDescriptor
- Beans

- DefaultPersistenceDelegate
- Encoder
- EventHandler
- EventSetDescriptor
- Expression
- FeatureDescriptor
- IndexedPropertyChangeEvent
- IndexedPropertyDescriptor
- Introspector
- MethodDescriptor
- ParameterDescriptor
- PersistenceDelegate
- PropertyChangeEvent
- PropertyChangeListenerProxy
- PropertyChangeSupport
- PropertyDescriptor
- PropertyEditorManager
- PropertyEditorSupport
- SimpleBeanInfo
- Statement
- VetoableChangeListenerProxy
- VetoableChangeSupport
- XMLDecoder
- XMLEncoder

Q7. What do you mean by the concept of persistence?

Ans: Persistence is the ability to save the current state of a Bean, which also includes the values of Bean's properties and instance variables to nonvolatile storage and thus to retrieve them at a later stage.

Q8. List the variations on standard properties that the JavaBeans model supports.

Ans: The following are the two variations on standard properties:

- Bound properties
- Constrained properties

Q9. List the constructors of the MethodDescriptor class.

Ans: The constructors of the MethodDescriptor class are as follows:

- MethodDescriptor(Method method)
- MethodDescriptor(Method method, ParameterDescriptor[] parameterDescriptors)

Q10. List the constructors of the PropertyDescriptor class.

Ans: The constructors of the PropertyDescriptor class are:

- PropertyDescriptor(String propertyName, Class beanClass)
- PropertyDescriptor(String propertyName, Class beanClass, String getterName, String setterName)
- PropertyDescriptor(String propertyName, Method getter, Method setter)