Integrating PHP and XML

SkillSoft Press © 2004

Learn how to use SAX, XSLT, and XPath to manipulate XML documents, as well as use of XML-RPC protocol for accessing procedures on a remote computer, and much more.

Table of Contents

Introduction

About InstantCode Books

The InstantCode series is designed to provide you - the developer - with code you can use for common tasks in the workplace. The goal of the InstantCode series is not to provide comprehensive information on specific technologies - this is typically well-covered in other books. Instead, the purpose of this series is to provide actual code listings that you can immediately put to use in building applications for your particular requirements.

How These Books are Structured

The underlying philosophy of the InstantCode series is to present code listings that you can download and apply to your own business needs. To support this, these books are divided into chapters, each covering an independent task.

Each chapter includes a brief description of the task, followed by an overview of the element of the book's subject technology that we will use to perform that task. Each section ends with a code listing: each of the individual code segments in the chapter is independently downloadable, as is the complete chapter code. You will be able to download source code files, as well as application files.

Who Should Read These Books

These books are written for software development professionals who have basic knowledge of the associated technology and want to develop customized technology solutions.

About the Book

PHP is a server-side scripting language used to create Web applications. XML is a markup language used to exchange data among Web applications. PHP can be integrated with XML to create Web applications. This book describes how to use SAX, XSLT, and XPath to manipulate XML documents. It also describes use of XML-RPC protocol for accessing procedures on a remote computer. In addition, the book covers WDDX, a technology used for information exchange between different programming languages.

This book describes how to create an online shopping cart application that allows an end user to search for a specific book in a database, place an order for the book, and purchase the book online.

About the Authors

Amrita Dubey holds a Bachelor's degree in Electronics and Communication Engineering. She has worked on various development projects, such as UNIX shell programming and designing function generator using IC XR-2206. She is proficient in 8085 microprocessor programming. She also has knowledge of CNCs, PLCs, circuit designing on PCBs, Software Quality Testing, RDBMS, SQL Server, and Database Design Studio Professional 2.21.1. As a technical writer, Amrita has co-authored books on Digital Electronics, Integrating Java with Oracle9i, PIC Microcontrollers, Robotics, and Integrating PHP and XML.

Poonam Sharma holds a Bachelor's degree in Commerce and a DNIIT diploma. In addition, she is pursuing Master's degree in Computer Applications. She is proficient in technologies such as Java, C++, VB, ASP, Servlets, JSP, HTML, UNIX, Linux, and SQL Server 2000. Poonam has worked on projects such as Online Banking using VB and SQL and Creating Chat Application in Java. As a technical writer, she has co-authored books on Working with Korn Shell, Administering Red Hat Linux 9, Basic Programming in C++, Unix Operating System, and Integrating PHP and XML.

Sreeparna Dasgupta holds a Master's degree in Computer Science and has earned 'A' Level certificate from DOEACC. She has worked and trained people on several technologies, such as Java, C, C++, Visual Basic, Macromedia Flash MX, Photoshop, Macromedia Director MX, Oracle, SQL Server, Perl, ASP, and JSP. In addition, Sreeparna has developed online Web applications using technologies such as ASP.

Vishi Gupta holds a degree in Electrical Engineering. She is proficient in C, C++, Linux and PHP. As a technical writer, she has co-authored several books and articles on varying technologies including PHP, XML, SOAP, Linux, Core Java, PHPLens, C and C++.

Lalit Kapoor holds a Bachelor's degree in Computer Engineering. He has completed DNIIT course from NIIT. As a technical writer, he has authored articles on Java, Oracle, SQL Server, and PHP.

Credits

I would like to thank Reena Roy, S. Sripriya, Sabari Roy, and Rajender Arora for helping me complete the book on time. I also thank the editors and the quality assurance team for their timely help.

Copyright

Integrating PHP and XML

Published by SkillSoft Corporation
20 Industrial Park Drive
Nashua, NH 03062
(603) 324-3000

information@skillsoft.com

Chapter 1: Introducing PHP and XML

Download CD Content

PHP is a server-side HTML embedded scripting language that you can use to create dynamic Web pages. It includes predefined functions that create, open, read, write, and close files stored on the server. You can create XML-based Web applications using PHP.

Some browsers such as Netscape Communicator 4.x series do not contain Some browsers such as Netscape Communicator 4.x series do not contain (XML) parser. Hypertext Preprocessor (PHP) overcomes this problem by supporting XML parsing. PHP provides a Document Object Model to access XML elements, an XML extension, and an eXtensible Stylesheet Language (XSL) processor to support XML parsing.

This chapter introduces PHP fundamentals. It describes data types, variables, operators, control structures, functions, and error handling in PHP. It also explains how to create classes in PHP and use PHP to work with files. This chapter also introduces XML fundamentals, such as Document Type Declaration (DTD), namespaces, and XML schemas.

Introducing PHP

PHP, developed using the C language, is specifically designed to work with databases. It provides a set of Application Programming Interfaces (APIs) to connect to different types of database servers, such as MySQL, SQL Server, and Oracle. PHP scripts are portable because they can run on various operating systems, such as Linux and Windows.

When a user accesses a Web page created using a PHP script, the Web server contacts the PHP engine to load, compile, and run the PHP script. Using PHP script, you can work with different data types, variables, constants, operators, functions, classes, objects, and files.

Data Types, Variables, and Constants

A variable is a container that stores variable data, such as number, character, and date.

PHP provides data types that specify type of values a variable can contain. For example, a data type defines that a variable, num, can contain only numbers. The data types that PHP supports are:

- Integer: Represents whole numbers. In PHP, integer variables can store values between -2, 147, 483, 648 and +2, 147, 483, 647. For example, in $a=5, $a is an integer variable that is assigned the value, 5.
- Float: Stores decimal values. PHP lets you store floating numbers in both normal and scientific notation. Scientific notation is a shorthand way of writing very large or very small numbers. A number expressed in scientific notation is expressed as a decimal number between 1 and 10, followed by e or E, multiplied by a power of 10. For example, in $a=5.2E2, $a is a float variable created using scientific notation and contains the value 520.
- Boolean: Stores the values, true or false. For example, in $a=true, $a is a Boolean variable that is assigned the value, true.
- String: Stores string information enclosed within double quotes. For example, in $a="Angela Jones", $a represents a string variable that contains the string, Angela Jones.
- Array: Stores elements in the form of key/value pairs. The key values start with zero.
- Object: Stores data and provides functions to process the data.
- Resource: Stores references to functions or other resources external to PHP.
- Null: Represents an undefined value.

Note In PHP, you can declare variables without specifying its data type.

Operators

PHP provides several operators, such as the arithmetic, comparison, logical, and assignment operators. You can perform arithmetic operations on variables and constants using the arithmetic operators of PHP.

Table 1-1 lists the arithmetic operators:

Table 1-1: Arithmetic Operators

Name	Symbol	Description
Addition	+	Adds two numbers.
Subtraction	-	Subtracts two numbers.
Multiplication	*	Calculates the product of two numbers.
Division	/	Divides two numbers.
Modulus	%	Calculates the remainder when one number is divided by the other.
Increment	++	Increments the number, to which it is applied, by 1.

Decrement	--	Decrements the number, to which it is applied, by 1.

PHP also supports the comparison operators to compare two or more values, to determine which value is greater, lesser, equal, or not equal to the other.

Table 1-2 lists the comparison operators in PHP:

Table 1-2: Comparison Operators

Name	Symbol	Description
Greater than	>	Returns true if the value on the left hand side of the operator is greater than the value on the right hand side.
Less than	<	Returns true if the value on the left hand side of the operator is less than the value on the right hand side.
Greater than equal to	>=	Returns true if the value on the left hand side of the operator is greater than or equal to the value on the right hand side.
Less than equal to	<=	Returns true if the value on the left hand side of the operator is less than or equal to the value on the right hand side.
Equal to	==	Returns true if the two values being compared are equal.
Not Equal to	!=	Returns true if the two values being compared are not equal.

PHP supports the logical operators that you can use to create Boolean expressions. Boolean expressions either return true or false.

Table 1-3 lists the operators that PHP supports:

Table 1-3: Logical Operators in PHP

Name	Symbol	Description
And	&&	Returns true if both the conditions placed on the left and right of the operator are true.
Or	||	Returns true if either of the conditions placed on the left and right of the operator is true.
Not	!	Returns true if the condition specified is not true.

You can use assignment operators in PHP to assign values to the variables. PHP also supports compound operators that perform the functions of the arithmetic and assignment operators in a single statement.

Table 1-4 lists the assignment and compound operators in PHP:

Table 1-4: Assignment and Compound Operators

Operator	Implementation	Description
=	a =5	Assign the value, 5, to the variable, a.
+=	a+=5	Adds the value, 5, to the original value of the variable, a, and then assigns the sum to the variable, a. This is equivalent to a=a+5.
-=	a-=5	Subtracts the value, 5, from the original value of the variable, a, and then assigns the difference to the variable, a. This is equivalent to a=a-5.
*=	a*=5	Multiplies the value, 5, with the original value of the variable, a, and then assigns the product to the variable, a. This is equivalent to a=a*5.
/=	a/=5	Divides the value in the variable, a, by 5 and then assigns the result to the variable, a. This is equivalent to a=a/5.
%=	a%=5	Divides the value in the variable, a, by 5 and then assigns the remainder to the variable, a. This is equivalent to a=a%5.

Control Structures

You can control the flow of execution of a PHP script using control structures. Two types of control structures are conditional statements and loops.

Conditional statements support unidirectional flow of control in processing a script. The conditional statement returns the true or false value after evaluating a condition, and then transfers the control to a section of the script, depending on the value returned after evaluating the condition. In PHP, two types of conditional statements are if else and switch case statement.

The if else conditional statement is divided into two parts, the if statement and the else statement. The if statement evaluates a condition and returns the value, true, or false. The if statement is followed by the PHP script enclosed in braces. This script runs if the value returned is true. The else statement contains the PHP script that runs if the condition evaluated is false. The syntax of the if else conditional statement is:

```
if(condition)
{statements to run if condition is true}
else
{statements to run if condition is false}
```

In the above syntax, the condition parameter is an expression that is evaluated to return true or false. To run a single PHP programming statement, you do not need the braces.

Listing 1-1 shows how to use the if condition to display the maximum of three numbers:

Listing 1-1: Maximum of Three Numbers

```
<?php
$x=10;
$y=20;
$z=30;
if($x > $y)
{
   if($x > $z)
   {
      echo("The maximum number is ");
      echo("$x<p>");
   }
   else
   {
      echo("The maximum number is ");
      echo("$z<p>");
   }
}
else
{
   if($y > $z)
   {
      echo("The maximum number is ");
      echo("$y<p>");
   }
   else
   {
      echo("The maximum number is ");
      echo("$z<p>");
   }
}
?>
```

The above listing shows the use of the if else condition statements to create a PHP script. The if condition compares the values of the two variables, and then controls the flow of execution of the script, depending on whether the condition returns true or false.

Create a new folder, PHP_XML, in the var/www/html directory of the Apache Web server. Save the above listing as MaxThree.php in the PHP_XML folder.

Figure 1-1 shows the output when PHP_XML is viewed in the Konqueror Web browser:

Figure 1-1: Viewing MaxThree.php

The switch case statement is another conditional statement that controls the flow of execution of a script. The switch statement compares a single variable with several values. To specify the values to which a variable is compared, you use the case statement with the switch statement. The syntax of the switch case conditional statement is:

```
switch($Variable)
{
   case 1:
   //statements to be run if case is true;
   break;
   case 2:
   //statements to be run if case is true;
   break;
   default:
   //statements to be run if all cases are false;
}
```

In the above syntax, the break statement is used within a case statement to identify the end of a case. The default statement is used in the switch case condition to specify the PHP statements that are run if all the cases are false.

Listing 1-2 shows how to use the switch case conditional statement to perform simple arithmetic operations:

Listing 1-2: Using the switch case Conditional Statement

```
<?php
//Initializing variables
$a = 10;
$b = 5;
echo("1. Addition <br>");
echo("2. Subtraction <br>");
echo("3. Multiplication <br>");
echo("4. Division <br>");
$ch=3;
switch ($ch)
{
   case 1:
   $d=$a+$b;
   echo("The sum of the two numbers is $d");
   break;
   case 2:
   $d=$a-$b;
   echo("The difference of the two numbers is $d");
   break;
   case 3:
   $d=$a*$b;
   echo("The product of the two numbers is $d");
   break;
   case 4:
   $d=$a/$b;
   echo("The quotient of the two numbers is $d");
   break;
   default:
   echo("Incorrect Option");
   break;
}
?>
```

The above listing shows how to use the switch case statement. The switch case statements:

- Display the sum of two numbers, if the variable, $ch, is assigned the value, 1.
- Display the difference of the two numbers, if the variable, $ch, is assigned the value, 2.
- Display the product of the two numbers, if the variable, $ch, is assigned the value, 3.
- Display the quotient of the two numbers, if the variable, $ch, is assigned the value, 4.
- Assign the variable, $ch, the value 3 to multiply two numbers.

Because value, 3, is assigned to the variable $ch, Listing 1-2 displays the product of the two numbers.

Save the above listing as Switch.php in the PHP_XML folder.

Figure 1-2 shows the output when the Switch.php file is viewed in the Konqueror Web browser:

Figure 1-2: Viewing Switch.php

Loops support the repetitive execution of a set of statements in the PHP script. The loop statements evaluate a Boolean condition on each iteration that returns true or false after evaluating a condition. A set of statements enclosed within a loop is processed if the condition evaluates to true. The different types of loop statements in PHP are:

- `while loop`
- `do while loop`
- `for loop`
- `foreach loop`

The while loop evaluates a condition as true or false. The variable that you use in the while condition needs to be initialized before the while loop. The value of the variable is changed as the while loop executes. The execution of the while loop terminates as soon as the specified condition becomes false. The syntax for using the while loop is:

```
$var=initialization
while(condition checking the value of the variable, $var)
{
   //statements to run;
   //increment/decrement var
}
```

In the above syntax, the variable, $var, is initialized outside the while loop. The while loop evaluates the condition as true or false and then runs the statements within the loop, if the condition is true. The increment or decrement operation within the loop modifies the value of the variable used in the condition till the condition becomes false.

Listing 1-3 shows how to use the while loop to display even numbers from 2 to 30:

Listing 1-3: Using the while Loop

```
<?php
$a=2;
echo("<p>Even Numbers from 2 to 30<p>");
while ($a <=30)
{
   echo("$a <br>");
   $a=$a+2;
}
?>
```

The above listing shows how to use the while loop to display even numbers. In the above listing:

- The variable, a, is initialized to value 2.
- The while condition compares whether the value of the variable is less than or equal to 30, and returns true or false.
- The increment operation adds 2 to the value of the variable, a, every time the loop runs.

Save the above listing as While.php in the PHP_XML folder.

Figure 1-3 shows the output when While.php is viewed in the Konqueror Web browser:

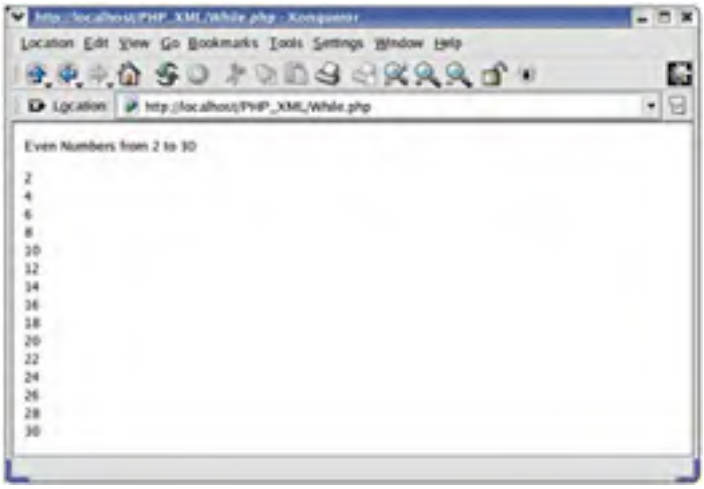


Figure 1-3: Viewing While.php

In the do-while loop, unlike in the while loop, the condition is evaluated after executing the statements once in the do-while loop. The syntax of the do-while loop is:

```
$var = initialization
do
{
   //statements to run;
   //increment/decrement var;
} while(condition);
```

In the above syntax, the variable is initialized outside the do-while loop. The do keyword indicates the beginning of the do-while loop. The while condition is specified at the end of the listing and ends with the semicolon symbol.

In a for loop statement, you can initialize a counter variable, specify the condition, and specify the increment or decrement condition in the loop statement together. The syntax of the for loop statement is:

```
for(initialization;condition;increment_decrement_operation)

{
   //Statements to run;
}
```

You can embed one for loop statement inside another.

Listing 1-4 shows how to use the for loop to display prime numbers:

Listing 1-4: Using the for Loop

```
<?php
echo("Prime Numbers between 1 and 30<p>");
//For loop from 1 to 30
for($var=1;$var<30;$var++)
{
   $ctr=0;
   for($k=2;$k<=$var/2;$k++)
   {
      $d=$var%$k;
      if ($d == 0)
      {
         $ctr=1;
         break;
      }
   }
   if($ctr==0)
   {
      echo($var);
      echo("<br>");
   }
}
?>
```

The above listing shows how to use the for loop to display prime numbers from 1 to 30. The PHP script uses nested for loops to display the value of the variable, $var. Save the above listing as For.php in the PHP_XML folder.

Figure 1-4 shows the output when For.php is viewed in the Konqueror Web browser:

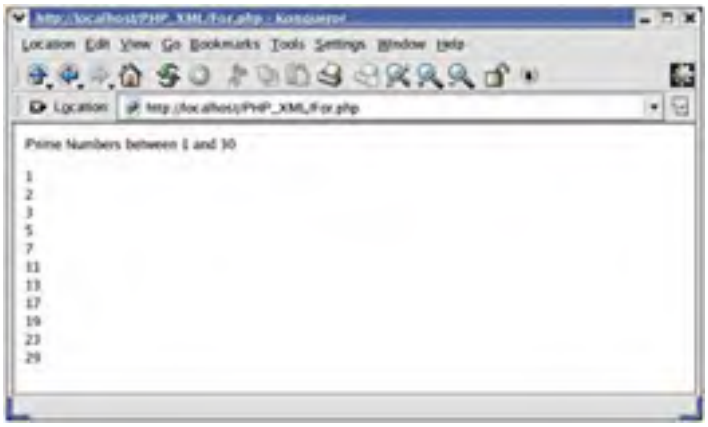


Figure 1-4: Viewing For.php

Functions

A function is a set of instructions that is grouped under a name. Instead of repeating all the instruction steps every time you use them in the script, you can call the function to run the same set of instructions again. PHP provides a set of built-in functions to use in the PHP scripts. In addition to the built-in functions, you can also define certain functions called user-defined functions. The syntax to define a function in PHP is:

```
<?php
function funcname($arg1,$arg2)
{//Statements to run;
return $val;}
?>
```

In the above syntax:

- The function keyword defines a PHP function.
- The funcname parameter defines the name of the function.
- The $arg1 and $arg2 parameters indicate the arguments that the function can receive.
- The return keyword returns a value from the function.

Listing 1-5 shows how to define and call a function in PHP:

Listing 1-5: User-Defined Function in PHP

```
<?php
echo("<center><h2>Displaying even numbers</h2></center><p><p>");
function displayeven()
{
   $ctr=0;
   echo("<font size=4>");
```

```
   for($i=2;$i<=100;$i+=2)
   {
      echo("$i    ");
      $ctr++;
      if($ctr%10==0)
      {
         echo("<p>");
      }
   }
   echo("</font>");
}
echo("<center><h2>Displaying even numbers</h2></center><p><p>");
displayeven();
?>
```

The above listing creates a function, displayeven() that displays even numbers greater than zero and less than or equal to 100.

Save the above listing as Functions.php in the PHP_XML folder.

Figure 1-5 shows the output when Function.php is viewed in the Web browser:

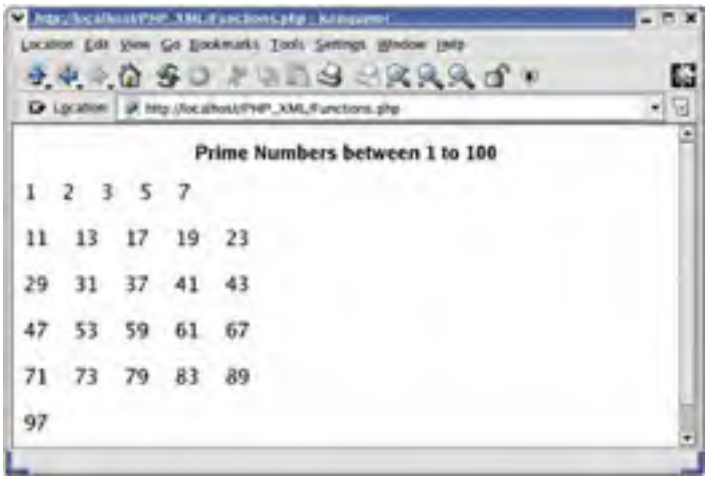


Figure 1-5: Viewing Functions.php

You can create nested functions, where you can define a function within another function. For example, in an application where you need to generate all prime numbers between two numbers entered by the end user, the outer function accepts the end user input and the inner function generates prime numbers. The inner function is a nested function, and is called only after calling the outer function. The syntax for creating a nested function is:

```
<?php
function outer_function($arg1)
{
   //Statements to run for outer_function();
   function inner_innerfunction($arg2, $arg3)
   {
      //Statements to run for inner_function()
   }
   outer_function();
   inner_function();
}
```

The above syntax shows how to create two functions, inner_function() and outer_function(). The inner_function() function is nested within the outer_function() function. The outer_function() function is called before the inner_function() function.

> **Note** In a nested function definition, calling the inner function before the outer function results in an error.

Listing 1-6 shows how to use nested functions in PHP:

Listing 1-6: Using Nested PHP Functions

```
<?php
function msg()
{
   echo("<center><h2>Displaying even
   numbers</h2></center><p><p>");
   function displayeven()
   {
      $ctr=0;
      echo("<font size=4>");
      for($i=2;$i<=100;$i+=2)
      {
         echo("$i    ");
         $ctr++;
         if($ctr%10==0)
         {
```

```
            echo("<p>");
        }
    }
    echo("</font>");
  }
}
msg();
displayeven();
?>
```

The above listing shows how to use nested functions. In the above listing:

- the displayeven() function is nested within the msg() function.
- the msg() function displays a message, Displaying even Numbers.
- the displayeven() function displays even numbers between 1 and 100.
- the displayeven() function is called only after calling the msg() function.

Save the above listing as NestedFunction.php in the PHP_XML folder.

Figure 1-6 shows the output when NestedFunction.php is viewed in the Web browser:

Figure 1-6: Viewing NestedFunction.php

Error Handling

An error represents an incorrect result, produced either because of wrong logic, such as division by zero, or improper working of other applications, such as the database server that the PHP script is using. PHP displays the error message either on the Web page, or in the error log files. The error message contains information about the nature and source of an error.

PHP has built-in support for error handling and classifies errors according to the following error types:

- E_NOTICE: Represents minor nonfatal errors that you use to identify the possible bugs within the PHP script. These errors do not stop the execution of the PHP script. E_NOTICE errors are not displayed on the Web page by default.
- E_WARNING: Represents nonfatal runtime errors that do not stop the execution of the PHP scripts. The errors generated by external applications, such as database servers, are represented by E_WARNING errors. These errors differ from E_NOTICE as the latter is associated with identifying bugs within the PHP script whereas the former is generated by external applications.
- E_ERROR: Represents fatal errors that stop the execution of the PHP script.
- E_PARSE: Represents parsing errors generated by the PHP parser, which indicate syntax error in the PHP script.

In addition to the standard errors, PHP also provides custom error types that PHP scripts can generate using the trigger_error() function. The custom error types are:

- E_USER_NOTICE: Represents the customized or user-defined version of the E_NOTICE error type.
- E_USER_WARNING: Represents the customized or user-defined version of the E_WARNING error type.
- E_USER_ERROR: Represents the customized or user-defined version of the E_ERROR error type.

PHP provides built-in error handling functions that handle errors of different types, such as E_ERROR, E_WARNING. Some error handling functions are:

- int error_reporting (int err_type): Specifies what error types should be displayed on the Web page. The parameter, err_type, takes either integers or named constants to represent the error type. Table 1-5 shows the named constants and their equivalent integer values:

Table 1-5: Named Constants and Equivalent Integer Values

Named Constant	Integer Value
E_ERROR	1
E_WARNING	2
E_PARSE	4
E_NOTICE	8
E_USER_ERROR	256
E_USER_WARNING	512
E_USER_NOTICE	1024
E_ALL	2047

- `int error_log ( string message [, int message_type [, string destination ]])`: Sends the error message to the specified location. The specified location can be the default system log file, an email address, a remote destination, or a local log file. The first parameter represents the error message that is stored in the specified destination. The second parameter represents the message type. The third parameter represents one of the four destinations where the error message needs to be stored. The second parameter can take one of the following values:
  - `0`: Stores the error messages in the default system log file.
  - `1`: Sends the error message to the destination email ID specified in the third parameter.
  - `2`: Saves the error message in a file present on a different computer in a network. The third parameter specifies the IP address and the port number of the destination computer.
  - `3`: Sends the error message to the destination file specified in the third parameter.
- `void trigger_error ( string error_msg [, int error_type])`: Triggers user-defined warnings, notices, or errors. The first parameter represents the error message and the second parameter represents the error type that needs to be triggered.

Listing 1-7 shows how to trigger errors using the trigger_error() function.

Listing 1-7: Generating Errors Using the trigger_error() Function

```
<?php
// set the error reporting level for this script
error_reporting(E_USER_ERROR);
if (assert($divisor == 0))
{
  error_log("Cannot perform division by zero", 3, "var/www/html/PHP_XML/error.log");
  trigger_error("Cannot perform division by zero", E_USER_ERROR);
}
?>
```

The above listing generates an error that stops the execution of the script if the condition passed to the assert() function is true. In the above listing:

- The assert() function checks whether the divisor is zero or not.
- The error_log() function sends the message, Cannot perform division by zero, to the log file specified by the second parameter when the divisor is zero.

Save the above listing as Trigger.php in the PHP_XML folder.

Figure 1-7 shows the output when Trigger.php is viewed in the Web browser:

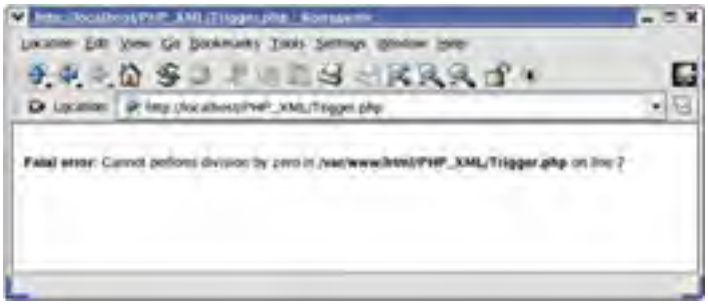


Figure 1-7: Viewing Trigger.php

You can also define your own custom error handlers. You need to define a function that acts as an error handler. The syntax to declare a custom error handling function is:

```
function func_name($errorno, $errorstr [,$errorfile, $errorline, $errorcontext])
```

The above syntax shows how to declare an error handling function. In the above code:

- The func_name represents the name of the error-handling function.
- The function must accept at least two parameters and a maximum of five parameters.
- The first two parameters represent error number and error description. The remaining parameters are optional.
- The third parameter represents the file name in which the error occurred.
- The fourth parameter represents the line number, and the fifth parameter represents the context in which the error occurred.

After defining an error handling function, you need to set it as the error handler using the set_error_handler() function. This function takes the name of the error handling function as a parameter.

Listing 1-8 shows how to use a custom error handler:

Listing 1-8: Using Custom Error Handler

```
<php
error_reporting(E_ALL);
function ErrHandler($errorno, $errorstr, $errorfile, $errorline)
{
   $display = true;
   $notify = false;
   $halt_script = false;
   $error_msg = "<br>The $errorno error is occurring at $errorline in
   $errorfile<br>";
   switch($errorno)
   {
      case E_USER_NOTICE:
      case E_NOTICE:
      $halt_script = false;
      $notify = true;
      $label = "<B>Notice</B>";
      break;
      case E_USER_WARNING:
      case E_WARNING:
      $halt_script = false;
      $notify = true;
      $label = "<b>Warning</b>";
      break;
      case E_USER_ERROR:
      case E_ERROR:
      $label = "<b>Fatal Error</b>";
      $notify=true;
      $halt_script = false;
      break;
      case E_PARSE:
      $label = "<b>Parse Error</b>";
      $notify=true;
      $halt_script = true;
      break;
      default:
      $label = "<b>Unknown Error</b>";
      break;
   }
   if($notify)
    {
      $msg = $label . $error_msg;
      echo $msg;
   }
   if($halt_script) exit -1;
}
$error_handler = set_error_handler("ErrHandler");
echo "<BR><H2>Using Custom Error Handler</h2><BR>";
trigger_error("<BR>Error caused by E_USER_NOTICE</BR>", E_USER_NOTICE);
trigger_error("<BR>Error caused by E_USER_WARNING</BR>", E_USER_WARNING);
trigger_error("<BR>Error caused by E_USER_ERROR</BR>", E_USER_ERROR);
trigger_error("<BR>Error caused by E_PARSE</BR>", E_PARSE);
?>
```

The above listing defines a custom error handler function, ErrHandler(), which handles the error generated by the trigger_error() function. In the above listing:

- The $display variable is a configuration flag that indicates whether to display the error on the Web page or not. The configuration flags are used for the proper functioning of PHP scripts.
- The $notify variable is another configuration flag that specifies whether the error should be notified to the end user or not. The default value for the $notify variable is false.
- The $halt_script configuration variable specifies whether to stop the execution of the PHP script when a fatal error occurs. The default value of the $halt_script variable is true.

- The $error_msg variable represents the error message to be displayed when an error occurs.
- The switch statement execute statements for any error type based on the value of the $error variable.
- The $notify variable displays the error message.
- The ErrHandler() function is set as the error handler using the set_error_handler() function.
- The trigger_error() function triggers the error of each of the error types.

Note The trigger_error() function can only trigger error of the E_USER types. It cannot trigger error for standard error types. Save the above listing as Trigger2.php.

Figure 1-8 shows the output when Trigger2.php is viewed in the Konqueror Web browser:

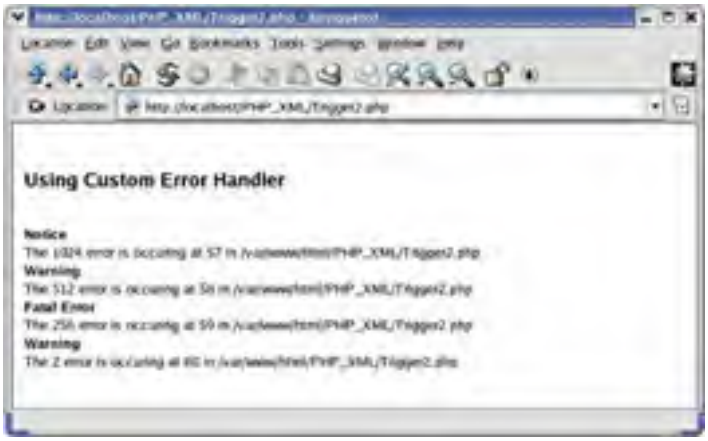


Figure 1-8: Viewing Trigger2.php

Classes and Objects

A class defines the properties and methods of an object. The properties are represented by the variables defined in the class and a method represents a function associated with an object. If student is a class, the variables in the student class represent the attributes of the student object, such as name and age of a student, and a method can be defined to calculate the grade. You can create classes in PHP using the keyword, class. The syntax to define a class is:

Defining a Class

```
<?php
class Class_name
{
   var $variables;
   function class_function($arg1, $arg2)
   {
      // Statements to run within the class function
      $this variables=$arg1;
   }
}
?>
```

The above syntax shows the use of the keyword, class, for creating a class. In the above syntax:

- The Class_name parameter indicates the name of the class that you are creating.
- The $variables parameter indicates the class variable in the class.
- The class_function parameter indicates the name of the class function.
- The keyword, `this`, access the current object properties and methods. Using the this variable, you can access properties to assign values, such as $arg1, to it.

You create objects of a class to use the methods defined in the class. An object is an instance of the class that you use to call the class functions and assign values to the class variables. The syntax to create objects of a class is:

```
$object= new Class_name;
$object->class_function();
```

The above syntax shows the use of the keyword, new, to create an object, $object. The Class_name parameter indicates the name of the class for which you are creating the object. The keyword, this, is used to call the class_function() method of the current class object.

Listing 1-9 shows the process of creating classes and objects in PHP:

Listing 1-9: Using PHP Classes and Objects

```
<?php
class emp
{
   var $name;
   var $address;
   var $dept;
   function assign_info($n,$a,$d)
   {
      $this->name=$n;
      $this->state=$a;
      $this->dept=$d;
   }
   function display_info()
   {
      echo("<p>Employee Name : $this->name");
      echo("<p>State : $this->state");
      echo("<p>Department : $this->dept");
   }
}
$empobj = new emp;
$empobj->assign_info("Angela Jone","California","Accounts");
echo("<center><h2>Displaying Employee Information</h2></center>");
echo("<font size=4>");
$empobj->display_info();
echo("</font>");
?>
```

The above listing defines a class, emp, with the class variables, $name, $state, and $dept. In the above listing:

- The emp class also defines the method, assign_info() and display_info().
- The assign_info() method assigns values to the class variables.
- The display_info() method displays the values assigned to the class variables.

Save the above listing as Classes.php in the PHP_XML folder.

Figure 1-9 shows the output when Classes.php is viewed in the Web browser:

Figure 1-9: Viewing Classes.php

A constructor is a special function in a class that is executed at the time of object creation. You can create a constructor in PHP by defining a function with the same name as the class name.

Listing 1-10 shows how to define a constructor for a class:

Listing 1-10: Creating Constructors with Classes

```
<?php
class student
{
   var $name;
   var $age;
   var $grade;
   function student($n,$a,$g)
   {
      $this->name=$n;
      $this->age=$a;
      $this->grade=$g;
   }
   function display_info()
   {
      echo("<p>Name : $this->name");
      echo("<p>Age : $this->age");
      echo("<p>Grade : $this->grade");
```

```
    }
}
$sobj = new student("Rick Carter","15","A");
echo("<center><h2>Displaying Student Information</h2></center>");
echo("<font size=4>");
$sobj->display_info();
echo("</font>");
?>
```

The above listing defines the constructor for the class, student. In the above listing:

- The constructor, student, is a parameterized constructor because it accepts three arguments, $n, $a, and $g, and assigns them to the variables of the object created of this class.
- The values are passed to the constructor and assigned to the class variables at the time of object initialization.

Save the above listing as Constructor.php in the PHP_XML folder.

Figure 1-10 shows the output when Constructor.php is viewed in the Web browser:

Figure 1-10: Viewing Constructor.php

Working with Files

You can perform the read, write, and append operations on PHP files. You need to open a file before you perform the read and write operations on it. You can open existing files using the PHP function, fopen(). The syntax to use the fopen() function is:

```
$fobj=fopen("FileURL","Mode");
```

In the above syntax:

- The parameter, FileURL, indicates the name of the file that is opened using the fopen() function.
- The parameter, Mode, defines the mode in which you open a file, such as the read, write, and append modes.

Table 1-6 lists the various modes in which you can open a file:

Table 1-6: File Open Modes

Symbol	Mode
Are	Represents read-only mode.
r+	Represents read and write mode.
W	Represents write mode.
w+	Represents read and write mode. Creates a new file if the file does not exist.
A	Represents append mode.
a+	Represents append mode. Creates a new file if the file does not exist.

You can read a file after you have opened it. PHP provides the fread() function to read the content of a file. The syntax of the fread() function is:

```
$text=fread(FileObject, filesize(FileURL));
```

In the above syntax, the parameter, FileObject, represents the name of the file to be read. The filesize() function calculates the size of the file passed to it as parameter, FileURL.

You can write data to a file after you open it in the write or append modes. PHP provides the fputs() function to write data to the file. The syntax to use the fputs() function is:

```
fputs(FileObject, $text);
```

In the above syntax:

- The FileObject parameter represents the file in which data needs to be written.
- The $text parameter indicates the text that is written to the file.

Listing 1-11 shows how to use the file handling functions to display the content of a text file:

Listing 1-11: File Handling Functions

```
<?php
$file='info.txt';
$fobj=fopen($file,"r");
$text=fread($fobj, filesize($file));
echo("<h2><font color='blue'>Displaying Content of a Text File</font><h2>");
echo("<br>");
echo("<font size=3>");
echo($text);
echo("</font>");
fclose($fobj);
?>
```

The above listing shows how to display the content of a text file. In the above listing:

- The fopen() function opens the info.txt file in the read-only mode.
- The fread() function reads the content of the info.txt file.
- The fclose() function closes the file object after the information of the text file is displayed on the Web page.

Save the above listing as File.php in the PHP_XML folder.

Create the info.txt file in the PHP_XML folder. The following code shows the content of info.txt file:

Hello World. This is a test file.

Figure 1-11 shows the output when File.php is viewed in the Web browser:

Figure 1-11: Viewing File.php

PREVIOUS NEXT

Introducing XML

XML is a markup language using which you can store data in a structured format in plain text files, which can be read by many applications. You can reuse data in different types of applications, such as database applications. XML works in combination with HTML to separate data from its presentation. For example, in XML, you can define a name as a combination of first and last name; HTML describes how data should be displayed on the Web page. The documents created in XML are stored with the extension .xml.

In HTML, you cannot use tags other than those that are already defined. Using XML, you can create your own tags.

Structure of XML Document

An XML document consists of user-defined tags. The XML document starts with a processing instruction, <?xml version="1.0">.

The tags that you create in the XML documents are called elements. All XML documents contain a root element, which is the outermost tag in the document. It is mandatory to close all tags that you create in an XML document. XML is case sensitive, and the closing tag must match the opening tag.

Listing 1-12 shows the structure of an XML document:

Listing 1-12: Structure of an XML Document

```
<?xml version="1.0"?>
<Books>
<Book>
<Name>Programming in VC++</Name>
<Author>Stephen Miller</Author>
<Category>Programming</Category>
<Price>50</Price>
</Book>
<Book>
<Name>Designing Websites in Dreamweaver MX</Name>
<Author>Angela Jones</Author>
<Category>Web Design</Category>
<Price>20</Price>
</Book>
</Books>
```

The above listing shows an XML document that stores information on several books. Save the above listing as Books.xml. The <Books> tag is the root element, and the starting and ending element of the document. Other user-defined tags in the listing are:

- `<Title>` tag: Indicates the name of the book.
- `<Author>` tag: Indicates the name of the author.
- `<Category>` tag: Indicates the type of the book.
- `<Price>` tag: Indicates the price of the book.

Figure 1-12 shows the output when you view Books.xml in the Mozilla Web browser:

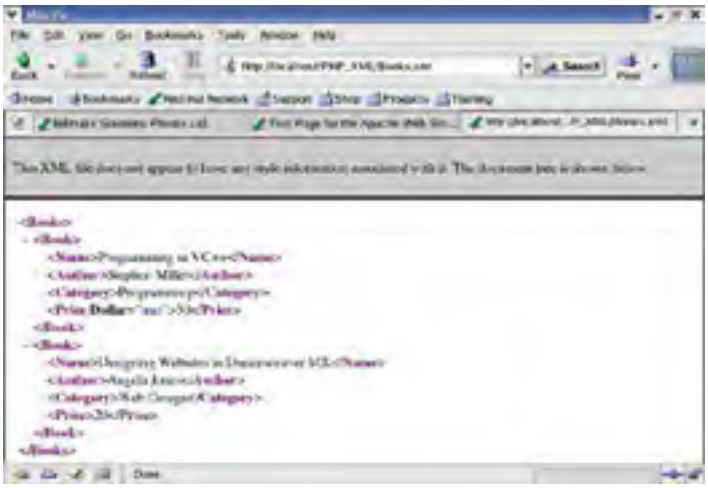


Figure 1-12: Viewing Books.xml

You can also use attributes with XML elements. An attribute provides additional information about the element. For example, you can indicate that the price of the books is in dollars, using an attribute, currency. The following code shows an example of using an attribute in an element:

```
<Price currency="dollar">20</Price>
```

You can also define empty elements in an XML document. An empty element does not contain any text. The following code shows an example of an empty element:

```
<Currency type="dollar"/>
```

You can also insert comments within XML documents. The XML parser does not parse the statements embedded in the comment tags. The following code shows how to insert comments in an XML document:

```
<!--Text for comments-->
```

The comment text is enclosed within the <!-- and --> symbols.

DTD

The rules that an XML document should follow can be defined using DTD. An XML document is valid if it conforms to the rules defined in a DTD document. The declarations in a DTD document include:

- `Element declarations`: Represent the rules for the tags in an XML document.
- `Attribute declarations`: Represent the rules for the attributes in the tags of XML documents.
- `Content declarations`: Represent the rules for text contained in the elements.

You can create two types of DTDs, which are:

- `Internal DTD`: Defines the rules for validating the structure of an XML file, within the XML file.
- `External DTD`: Creates the rules for validating the structure of an XML file in a separate document and stores the file with the extension .dtd.

A DTD begins with the syntax, <!DOCTYPE>, and the rules defining the structure of a document are present within the <!DOCTYPE> tag. The instruction tag, <!ELEMENT>, defines the rules for an element. An example of <!ELEMENT> is:

```
<!ELEMENT Book(Name, Author, Category, Price)>
```

In the above example, the Book element contains four elements, Name, Author, Category, and Price. The <ELEMENT> tag indicates that the <Book> tag must contain the elements, Name, Author, Category, and Price, in an XML document, in the given order.

You can also define the type of content within an element. The XML elements can contain two types of data: Parsed Character Data (PCDATA) and Character Data (CDATA). The XML parser parses PCDATA but does not parses CDATA. You can define the content in an element using the following syntax:

```
<!ELEMENT Name (#PCDATA)>
```

In the above example, the content within the Name element is of PCDATA type.

You declare attribute using the <!ATTLIST> instruction. You use the <!ATTLIST> instruction using the following syntax:

```
<!ATTLIST Price Currency CDATA #REQUIRED>
```

In the above syntax:

- Price is the name of the element for which you define the attribute rules.
- Currency is the name of the attribute for the Price element.
- The CDATA parameter specifies that the attribute value is of character data type.
- The #REQUIRED parameter specifies that it is mandatory to use the Currency attribute with the Price element.

Listing 1-13 shows an internal DTD for an XML document:

Listing 1-13: Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE Books [
<!ELEMENT Books (Book)+>
<!ELEMENT Book (Name, Author, Category, Price)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Price Currency CDATA #REQUIRED> ]>
<Books>
<Book>
<Name>Web Designing in XHTML</Name>
<Author>Angela Jones</Author>
<Category>Web Designing</Category>
<Price Currency="dollar">20</Price>
</Book>
<Book>
<Name>Programming in VC++</Name>
<Author>Stephen Miller</Author>
<Category>Programming</Category>
<Price Currency="dollar">50</Price>
</Book>
</Books>
```

The above listing shows the internal DTD created in an XML file. In the above listing:

- The XML document conforms to the rules defined in the internal DTD.
- The root element is defined as Books, and contains more than one Book element.
- The Book element contains the Name, Author, Category, and Price elements that contain PCDATA.
- The Price element defines the mandatory Currency attribute.

Save the above listing as InternalDTDBooks.xml.

Figure 1-13 shows the output when InternalDTDBooks.xml is viewed in the Mozilla Web browser:

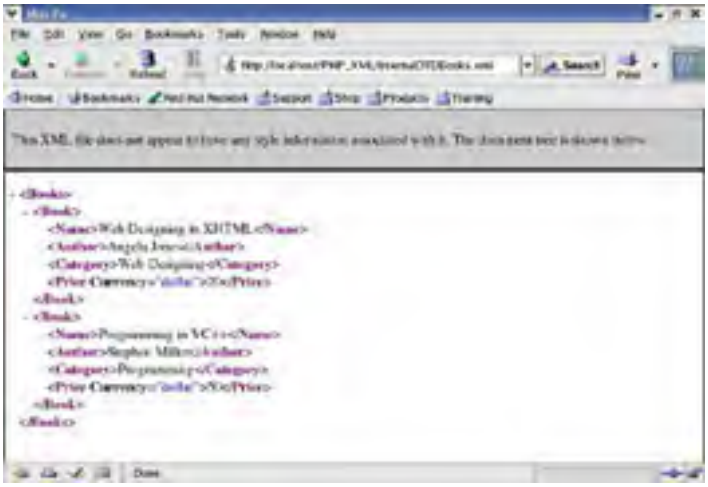


Figure 1-13: Viewing InternalDTDBooks.xml

You can also define an external DTD, where the rules to validate the structure of an XML document are specified outside the XML document. In an external DTD, the rules are defined in a .dtd file, and the reference to the file is provided in the XML file that conforms to the rules of the external DTD.

Listing 1-14 shows an external DTD:

Listing 1-14: The External DTD MyDTD.dtd

```
<!ELEMENT Book (Name, Author, Category, Price)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Category EMPTY>
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Category Ctype CDATA #REQUIRED>
```

The above listing shows an external DTD file. The Book element contains the Name, Author, Category, and Price elements. The Category element is empty and defines an attribute, Ctype, which is mandatory with the Category element.

The <!DOCTYPE> instruction provides a reference to the external DTD document in an XML file.

Listing 1-15 shows the XML file with a reference to the external DTD:

Listing 1-15: Referencing an External DTD

```
<?xml version="1.0"?>
<!DOCTYPE Book "MyDTD.dtd">
<Book>
<Name>Programming in C++</Name>
<Author>Stephen Wright</Author>
<Category Ctype="Programming"/>
<Price>$35</Price>
</Book>
```

The above listing shows how to use the <!DOCTYPE> instruction to provide a reference to the external DTD file, MyDTD.dtd. The structure of the XML document is checked against the rules defined in MyDTD.dtd.

Namespaces

Namespaces are naming conventions that you use to prevent inconsistency between element names. An inconsistency between element names occurs if you use the same name to identify two different elements. You can prevent this inconsistency by assigning a prefix to the name of the elements.

Listing 1-16 shows how to assigning a prefix to an element:

Listing 1-16: Assigning Prefix

```
<doc>
<keyframes>12</keyframes>
<framerate>24</framerate>
<my:frame>
<my:keyframes>100</my:keyframes>
<my:framerate>100</my:framerate>
</my:frame>
</doc>
```

In the above code, you use the my prefix with the frame tag to avoid conflict with the existing HTML tags, <keyframe> and <framerate>. The <my:keyframes> and <my:framerate> tags are encapsulated in </my:frame> tags.

The xmlns attribute is used with a user-defined element, such as <my:frame>, to assign the namespace URI to the element. The syntax of using the xmlns attribute is:

```
<prefix:MyElement xmlns:prefix="NamespaceURI">
```

In the above syntax:

- The prefix parameter represents the name of the prefix that is used with the user-defined element to avoid name conflicts.
- The MyElement parameter represents the name of the user-defined XML element.
- The xmlns:prefix represents the namespace URI. In the xml:prefix attribute, the :prefix parameter represents the prefix that is being used to access the namespace.
- The NamespaceURI parameter is the namespace URI.

Listing 1-17 shows the use of namespaces in an XML file:

Listing 1-17: Namespaces in XML

```
<doc xmlns:document="namespace1"
xmlns:my="namespace2">
<document:keyframes>12</document:keyframes>
<document:framerate>24</document:framerate>
<my:frame>
<my:animationno>1</my:animationno>
<my:keyframes>100</my:keyframes>
<my:framerate>20</my:framerate>
</my:frame>
</doc>
```

The above listing shows how to create a file that contains user-defined XML elements with namespace prefixes applied on the elements.

XML Schemas

An XML schema defines the structure of an XML document and is an alternative to DTD. An XML Schema defines the following:

- Elements that can be used in an XML document.
- Attributes and text that an element can contain.
- Child elements that a parent element can contain, and the order of the child elements in which they appear within a parent element.

The <schema> element represents the root of an XML schema. The syntax to use the <schema> element is:

```
<?xml version="1.0"?>
<xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="namespace1"
elementFormDefault="qualified">
...
</xsi:schema>
```

In the above syntax:

- The prefix, xsi:, represents the XML namespace.
- Data types, such as schema, element, string, and Boolean, used in the XML schema, are present in the www.w3.org/2001/XMLSchema name space.
- The xmlns attribute specifies the default namespace, which is namespace1.
- The elementFormDefault attribute specifies that the namespace qualifier, xsi, should prefix every element declared in the XML document.

XML schemas are of two types, external and internal. An external schema is saved in a separate file with the extension, .xsd. An internal schema is defined within the same XML file that contains the XML elements. After defining an XML schema namespace prefix, you can use the schemaLocation attribute of the <schema> element to refer to an external schema. The schemaLocation attribute takes two values. The first parameter represents the namespace and the second parameter represents the location of the external schema. For example, if the name of the external schema is external.xsd, then you can refer to it in your XML document using the following code:

```
xsi:schemaLocation=" http://www.w3.org/2001/XMLSchema external.xsd"
```

After beginning the <schema> element, you can define various elements that it can contain. The syntax to define a schema element is:

```
<prefix:element name="xxx" type="yyy" default="value" fixed="value"/>
```

The above syntax shows how to define a schema element. In the above syntax:

- Prefix represents the namespace prefix, such as xsi.
- The name attribute represents the name of the element, and the type attribute represents the data type of the element.
- The default attribute specifies the default value for the element.
- The fixed attribute indicates that the element cannot have any other value except the value specified by the fixed attribute.

Some XML schema data types are: xsi:string, xsi:decimal, xsi:integer, xsi:Boolean, xsi:date, and xsi:time.

Listing 1-18 shows how to use the internal XML schema to define the structure of an XML document:

Listing 1-18: Using Internal XML Schema

```
<?xml version="1.0"?>
<xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema"
targetNamespace="namespace1"
xmlns="namespace2"
elementFormDefault="qualified">
<xsi:element name="Employee">
<xsi:complexType>
<xsi:sequence>
<xsi:element name="id" type="xsi:integer"/>
<xsi:element name="name" type="xsi:string"/>
<xsi:element name="address" type="xsi:string"/>
</xsi:sequence>
</xsi:complexType>
</xsi:element>
<Employee>
<id>1</id>
<name>John Smith</name>
<address>Washington, D.C.</address>
</Employee>
</xsi:schema>
```

The above listing defines an XML schema for the <Employee> element that contains three attributes: id, name, and address. In the above listing:

- The <xsi:complexType> element specifies that the <Employee> element contains other XML elements, such as <Id>, <name>, and <address>.
- The <xsi:sequence> element defines various elements that an <Employee> element contains in the order in which they appear in the XML document.

Save the above listing as EmployeeSchema.xml.

Figure 1-14 shows the output when EmployeeSchema.xml is viewed in the Mozilla Web browser:

Figure 1-14: Viewing EmployeeSchema.xml

Chapter 2: PHP and Simple Application Programming Interface for XML

 Download CD Content

Hypertext Preprocessor (PHP) is a server-side scripting language that runs on various platforms, such as Linux and Windows. You use PHP to read and parse an eXtensible Markup Language (XML) document using the Simple Application Programming Interface (API) for XML (SAX) parser. PHP displays the parsed XML document on a Web browser.

This chapter introduces the parsers, and explains how to implement the SAX parser to parse XML documents. It also describes the object-oriented framework of parsers.

Parsing an XML Document

Parsing validates the structure of an XML document by examining its syntax. A parser validates the structure of an XML document using Document Type Definition (DTD), which specifies the valid format for the syntax of the elements contained in an XML document. The XML parser checks the opening and closing tags within the code of an XML document, against the rules specified in DTD.

Various Web browsers, such as Mozilla, Konqueror, and Internet Explorer, contain built-in parsing capabilities to parse XML documents. To provide better readability and presentation of data, you can use PHP to parse the XML documents. PHP 3.0 or latest version supports XML parsing. To parse the XML document, you need to initialize the XML parser in the PHP code. You can use the data of the parsed XML document in other applications.

Introducing Parser

XML parser is a program that checks whether the XML document is a well-formed document. A document that satisfies the syntax and standard rules specified by the World Wide Web Consortium (W3C) for XML is called a well-formed document. There are two types of XML parsers, which are:

- Non-validating parser: Checks if an XML document is well-formed according to the XML syntax rules.
- Validating parser: Checks if the XML document is well-formed and valid according to the rules specified by DTD.

The parser translates the data of an XML document into platform-specific objects. The non-validating parser adopts the event-driven approach of parsing, and the validating parser adopts the tree-based approach of parsing.

In the event-driven parsing approach, the parser processes XML data and XML tags sequentially in the memory, one at a time. The XML parser generates an event that does parsing, when it finds any XML element or data within the elements of an XML document. An example of the event-driven parser is the SAX parser.

In the tree-based parsing approach, the parser processes and organizes an XML document in a hierarchical form, all at one time. It supports Document Object Model (DOM), which is a platform and language independent model. An example of the tree-based parser is the DOM parser. The DOM parser reads an XML document and divides it into various objects, such as elements, attributes, and comments. DOM creates a tree structure for each element of the XML document and stores the structure in the memory. As a result, the DOM parser performs fast searching of any element in the XML document, as compared to the SAX parser.

Creating a Parser in PHP

You can implement an XML parser by including the xml_parser_create() function in the PHP code. The syntax to create an XML parser is:

```
resource xml_parser_create(string encoding)
```

The above code shows that the xml_parser_create() function creates an XML parser, and specifies that the return type of the function is resource. The resource type returns the resources handled by the XML document, which are used by other functions of the document.

The parameter of the xml_parser_create() function is optional, and provides the character encoding schemes in which the XML document is parsed. The character encoding schemes supported by the XML parsers are: ISO-8859-1, UTF-8, and US-ASCII. UTF stands for Universal Character Set (UCS) Transformation Format. The default character encoding format is ISO-8859-1.

PHP 4.0 or later versions also support the creation of an XML parser with namespace support. The syntax to create an XML parser with namespace support is:

```
resource xml_parser_create_ns(string encoding, string name, string namespace, string delimiter)
```

The above code shows that the xml_parser_create_ns() function creates an XML parser that supports the XML namespaces. The function returns the resources handled by the XML document, which are used by other functions of the document. The xml_parser_create_ns() function consists of two optional parameters, name and namespace of the tag. The string delimiter, provided in the xml_parser_create_ns() function, separates the parameters. The default string delimiter is colon.

The xml_parse_free() function in PHP frees the resources handled by an XML parser, and releases the reference of the XML parser. You should include the function before the PHP script ends. If you do not use the xml_parse_free() function for releasing the resources, either the connection with the Web server closes or the segmentation fault error is displayed. The syntax to release the reference of an XML parser is:

```
bool xml_parser_free(resource parser)
```

In the above code:

- The parser parameter specifies the reference of the XML parser to be released.
- The function returns the Boolean values, true or false. The true value indicates the successful execution of the function, which means that the function frees the reference of an XML parser. The false value indicates the unsuccessful execution of the function, if the provided parameter, parser, is invalid.

Working with Well-formed XML Documents

An XML document starts with the XML declaration statement, which is called Processing Instruction (PI). PI specifies the encoding scheme to process an XML document. The code to use the PI statement in an XML document is:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The above code shows the XML version that you need to use in an XML document. The encoding attribute indicates the encoding scheme that you use for creating the XML document. You use the UTF-8 encoding scheme to create Web pages in English. The UTF-8 encoding scheme uses 8 bits, which are compatible with ASCII, to represent a character. You use the UTF-16 encoding scheme when an application uses languages other than English, such as Japanese, Katana, and Cyrillic. UTF-16 uses 16 bits to represent a character.

A well-formed XML document meets the standards and rules for XML provided by W3C. Rules for creating a well-formed XML document are:

- Ensuring that every element in the XML document must have a start and an end tag. For example you need to provide the starting and the ending tags of an element, as shown:

```
<LI>First</LI>
<LI>Second</LI>
```

- Closing an empty tag with a slash mark (/). Empty tags do not have closing tags because they do not contain data. The <IMG> and
 tags are examples of empty tags in XML, as shown:

```
<IMG src="image.gif" /> <BR />
```

The above code shows that the <IMG> and
 tags are empty tags that contain the slash mark preceding the closing angle bracket. The <IMG> tag contains the src attribute that indicates the source of the image file. The
 tag inserts a new line.

- Using quotation marks to provide attribute values, as shown:

```
<P align="right">
```

The above code shows that the <P> tag contains the align attribute with the value right. The value of the align attribute is enclosed within quotation marks.

- Closing the innermost tag before the outermost tag. For example,

```
<NAME> Michael <AGE> 25 </AGE></NAME>
```

The above code shows that the innermost tag, <AGE> is closed before the outermost tag, <NAME>.

- Matching the case of the starting and the ending tags. XML tags are case-sensitive. The mismatch of the cases of the starting and ending tags generates an error, as shown:

```
<NAME> Michael </name>
```

The above code generates an error because the cases of the starting and ending tags are mismatched. The correct code to use the tags is:

```
<NAME> Michael </NAME>
```

The above code shows that both the starting and ending tags are in the uppercase. You can also specify the starting and ending tags in lowercase.

- Including all other elements within the root element. Listing 2-1 shows that the root element contains all other elements of an XML document:

Listing 2-1: Using the Root Element of an XML Document

```
<EMPLOYEEDETAIL>
<EMPLOYEE>
<NAME>Michael</NAME>
<AGE>25</AGE>
<ADDRESS>New York</ADDRESS>
</EMPLOYEE>
<EMPLOYEE>
<NAME>John</NAME>
<AGE>26</AGE>
<ADDRESS>New Jersey</ADDRESS>
</EMPLOYEE>
</EMPLOYEEDETAIL>
```

The above listing shows that <EMPLOYEEDETAIL> is the root element, which contains other elements of the XML document. The <EMPLOYEEDETAIL> element contains two <EMPLOYEE> elements, which also contain other elements, <NAME>, <AGE>, and <ADDRESS>.

- Providing unique names to the attributes of the elements. You cannot create two attributes with the same name. For example:

```
<TITLE heading="Employee Details" heading="Employee Information">
```

The above code shows that the <TITLE> element contains two attributes with the same name, heading. As a result, the <TITLE> element is not valid in a well-formed XML document.

A document is well-formed if it adheres to all the above rules.

Listing 2-2 shows a well-formed XML document:

Listing 2-2: Creating Well-Formed XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<STUDENTDETAIL>
<STUDENT>
<NAME ID="S001">George</NAME>
<AGE>15</AGE>
<ADDRESS>New York</ADDRESS>
<STANDARD>10</STANDARD>
</STUDENT>
<STUDENT>
<NAME ID="S001">John</NAME>
<AGE>15</AGE>
<ADDRESS>New York</ADDRESS>
<STANDARD>10</STANDARD>
</STUDENT>
</STUDENTDETAIL>
```

In the above listing:

- The XML document is well-formed because it adheres to all the rules of a well-formed document.
- The XML document contains the processing instruction at the starting of the document.
- The root element, <STUDENTDETAIL>, contains other elements.
- All tags are closed and are written in the same case.
- The value of the ID attribute of the <NAME> tag is enclosed within quotation marks.

Introduction to SAX

The SAX parser is an event-based, non-validating parser that reads data from the XML document. The current version of SAX is SAX 2. SAX2 processes documents in a sequential manner. It reads a part of the XML document and generates events when it finds an XML tag. It then reads the next part of the XML document. You can use the SAX parser to modify, query, and write an XML document.

Architecture of the SAX Parser

The SAX parser checks the validity of the structure of an XML document. The SAX parser consists of various handlers that are invoked for each XML tag. The handlers are user-defined functions, which are also called callback functions.

Figure 2-1 shows the architecture of the SAX parser:

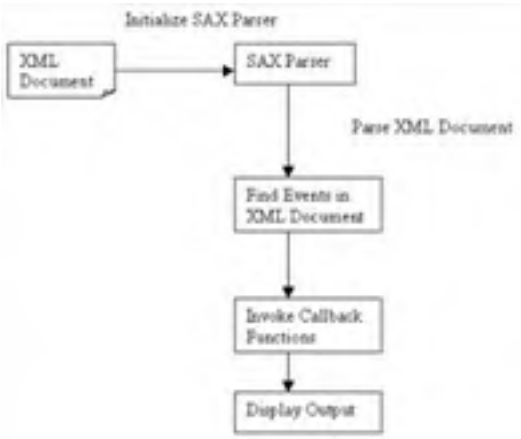


Figure 2-1: Architecture of the SAX Parser

Working with the SAX Parser in PHP

The SAX parser invokes handlers for each opening and closing tag in an XML document. It also invokes handlers for character data and processing instructions of the XML document. To use the SAX parser:

1. Initialize the SAX parser using the PHP function, xml_parser_create(). The code to initialize the SAX parser is:

   ```
   $xparser=xml_parser_create();
   ```

   The above code initializes the SAX parser and creates the xparser variable, which provides a reference to the SAX parser.

2. Identify the events, and set the callback functions to be invoked for the events. The code to identify the events and set the callback functions is:

   ```
   xml_set_element_handler($xparser, "startingHandler", "endingHandler");
   xml_set_character_data_handler($xparser, "cdataHandler");
   ```

   The above code shows that the xml_set_element_handler() function is the built-in function of PHP. In the above code:

   - The xml_set_element_handler() function invokes the callback functions for the opening and closing tags of an XML document.
   - The startingHandler function is the callback function that is invoked when the SAX parser finds an opening tag.
   - The endingHandler function is the callback function that is invoked when the SAX parser finds a closing tag.
   - The xml_set_character_data_handler() function is a built-in function in PHP. The function specifies the callback functions to be invoked for character data within the tags of the XML document.
   - The cdataHandler function is the callback function that is invoked when the SAX parser finds character data within the XML document.

3. Provide the code for the callback functions, startingHandler(), endingHandler(), and cdataHandler(), in the PHP script.

4. Open the XML document using the fopen() function, as shown in the following code:

   ```
   if(!($fp=fopen("student.xml","r")))
   {
      die ("File does not exist");
   }
   ```

   The above code creates the fp variable, which refers to the student.xml file. In the above code:

- The fopen() function opens the student.xml file in the read mode. If the fp variable does not contain the pointer to the XML file, the code displays the error message, File does not exist.
- The die() function is the built-in function of PHP that terminates the execution of the script and displays the message specified as an argument.

5. Parse the XML document using the xml_parse() function, as shown in Listing 2-3:

Listing 2-3: Parsing the XML Document

```
while($data=fread($fp, 4096))
{
 if(!xml_parse($xparser,$data,feof($fp)))
   {
      die("XML parse error: xml_error_string(xml_get_error_code($xparser))");
   }
}
```

In the above code:

- The SAX parser reads the content of the XML document in chunks of 4KB.
- The xml_parse() function parses the XML document until it reaches the end of the XML document.
- The feof() function returns the Boolean value, true, if the end of the document is reached, and notifies the parser to terminate the processing.
- The die() function terminates the execution when an error occurs in parsing the XML document.
- The xml_get_error_code() function returns the error code and the xml_error_string() function returns the error description corresponding to the error code.

6. Release the resources of the XML parser using the xml_parser_free() function of PHP, as shown in the following code:

```
xml_parser_free($xml_parser);
```

The above code releases the XML parser when the execution of the script ends.

Note To initialize the parser with other encoding schemes, use the following code:

```
$xparser=xml_parser_create("UTF-16");
```

Implementing the SAX Parser

The SAX parser consists of various functions, known as handlers. Each handler is invoked when the SAX parser finds certain events, such as opening tag, closing tag, character data, processing instructions, and comments.

For parsing an XML file, you need to provide the XML data file to the SAX parser, as shown:

```
$xfile="student.xml";
```

The above code creates the xfile variable that contains the name of the XML document to be parsed by the SAX parser. You can refer to the student.xml file using the $xfile variable within the PHP script.

To implement the SAX parser, you need to create callback functions for handling all events. PHP passes three parameters to the startingHandler() callback function, which are:

- Reference to SAX parser
- Element name
- Element attributes

Listing 2-4 shows the startingHandler() callback function:

Listing 2-4: Handling the Opening Tag Event

```
function startingHandler($xparser, $element_name, $attributes)
{
   echo "Opening Tag:<b>$element_name</b><br>";
   while (list($key,$value)=each($attributes))
   {
      echo "Attribute:<b><i>$key=$value</i></b><br>";
   }
}
```

In the above listing:

- The parser invokes the startingHandler() function when it finds the opening tag.
- The parser displays the name of opening tags in bold.
- The PHP functions, list() and each(), access the array variables.

- The parser also displays the name and value of the attributes of the tag elements in italics and bold.

Unlike the start tag handler, PHP passes two parameters to the endingHandler() callback function, because it does not contain attributes. The endingHandler() callback function is the end tag handler, which is invoked when the parser finds an end tag. The parameters passed to the end tag handler include the reference to the SAX parser and the element name.

The code to define the endingHandler() callback function, is:

```
function endingHandler($xparser, $element_name)
{
   echo "Closing Tag:<b>$element_name</b><br>";
}
```

The above code shows that the parser invokes the endingHandler() function when it finds the closing tag. It displays the names of the closing tags of the XML document in bold.

PHP passes two parameters to the character data handler. The parameters passed to the character data handler include the reference to the SAX parser and the character data.

The code to define the character data callback function, cdataHandler, is:

```
function cdataHandler($xparser, $cdata)
{
   echo "CDATA: <i><u>$cdata</u></i><br>";
}
```

The above code shows that the cdataHandler() function is invoked when the parser finds text between the opening and closing tags. The cdataHandler() function displays the text between the opening and closing tags in underlined and italicized format.

You can implement the SAX parser in a PHP script, as shown in Listing 2-5:

Listing 2-5: Implementing the SAX Parser

```
<html><head>
<basefont face="Times New Roman">
</head>
<body>
<?php
function startingHandler($xparser, $element_name, $attributes)
{
   echo "Opening Tag:<b>$element_name</b><br>";
   while (list($key,$value)=each($attributes))
   {
      echo "Attribute:<b><i>$key=$value</i></b><br>";
   }
}
function endingHandler($xparser, $element_name)
{
   echo "Closing Tag:<b>$element_name</b><br>";
}
function cdataHandler($xparser, $cdata)
{
   echo "CDATA: <i><u>$cdata</u></i><br>";
}
$xfile="student.xml";
$xparser=xml_parser_create();
xml_set_element_handler($xparser, "startingHandler","endingHandler");
xml_set_character_data_handler($xparser,"cdataHandler");
if(!($fp=fopen($xfile,"r")))
{
   die("File Input/Output error: $xfile");
}
while($data=fread($fp,4096))
{
   if(!xml_parse($xparser,$data,feof($fp)))
   {
      die("XML parser error: xml_error_string(xml_get_error_code($xparser))");
   }
}
xml_parser_free($xparser);
?>
</body>
</html>
```

The above listing shows that the SAX parser of PHP parses the XML document, student.xml. In the above code:

- The $xparser variable indicates the reference to the SAX parser.
- The xml_set_element_handler() function contains the functions, startingHandler() and endingHandler(), for handling the opening and closing tags respectively.
- The xml_set_character_data_handler() function contains the cdataHandler() function for handling the text between the opening and closing tags.

The content of the student.xml file is, as shown:

```
<?xml version="1.0"?>
<studentdata><student><name
id="s001">George</name><age>15</age><address>New
York</address><standard>10</standard></student></studentdata>
```

Note The cdataHandler() function also accepts any white space in the XML file as its parameter.

Figure 2-2 shows the output of Listing 2-5:

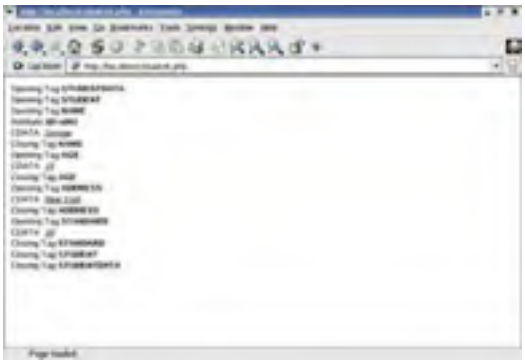


Figure 2-2: Output of Listing 2-5

Using the Expat Parser

The Expat parser is a SAX parser that supports the event-driven approach of parsing a document. The Expat parser is the default parser for the PHP scripting language. This parser contains wrapper classes and filters, which you use to perform advanced processing on XML documents, such as transforming, updating, and querying XML documents.

The PHPXML class is an API that contains wrapper classes, which implement filters in the SAX parser. The class_sax_filters.php file of the SAX parser is an example of a wrapper class.

Note You can download the PHPXML class from http://phpxmlclasses.sourceforge.net/

The AbstractSAXParser class implements the SAX parser. This class checks the XML document and invokes events using the objects of the AbstractFilter class, called listener objects. The methods used by the AbstractSAXParser class are:

- `AbstractSAXParser()`: Creates a parser by passing the XML document as an argument. It is a constructor of the AbstractSAXParser class.
- `Parse()`: Parses the XML document and invokes the StartElementHandler($xml_file, $element_name, $attributes), EndElementHandler($xml_file, $element_name), and CharacterDataHandler($cdata) functions.
- `SetListener()`: Assigns the listeners to a parser object.

The ExpatParser class implements the AbstractSAXParser class to parse an XML document. The constructor of the ExpatParser class accepts the XML document as an argument. The code to implement the ExpatParser class is:

```
$xml_parser=new ExpatParser("file.xml");
$f1=new FilterAddStudent();
$xml_parser->SetListener(f1);
$xml_parser->parse();
```

In the above code:

- file.xml is the name of the XML document, and the xml_parser variable refers to the Expat parser.
- FilterAddStudent is the user-defined class, which acts as a filter that is implemented by the AbstractSAXParser class.
- The SetListener() method accepts the object of the filter class as an argument and sets the filter class as listener of the events generated by the SAX parser.

Filters are user-defined classes that accept SAX events from a parser, process it, and provide the result to other filters or Web browsers. A filter class implements the methods that are defined in the AbstractFilter class. You need to extend the AbstractFilter class to create a user-defined filter. The handlers implemented by a filter class are:

- `StartElementHandler($element_name, $attributes)`: Accepts two arguments, name and attributes, of the element of the starting tag of an XML document. This handler is invoked when a parser finds the starting tag of an element of the XML document.
- `EndElementHandler($element_name)`: Accepts only one argument, because the ending tag of an XML document does not contain attributes. This handler is invoked when a parser finds the ending tag.
- `CharacterDataHandler($cdata)`: Is invoked when a parser finds text within the starting and ending tags.
- `SetListener($object)`: Sets the listener object of the filter. It accepts the object of the filter class as an argument. You need not implement the SetListener() function in the filter class, because the AbstractFilter abstract class already contains the functionality of the SetListener() function.

Filters that do not invoke other events, but display the output on the Web browser, are called finalizers. The FilterOutput() method is a finalizer, which displays the output of the XML document on the Web browser.

Listing 2-6 shows how to implement the AbstractFilter abstract class:

Listing 2-6: Implementing the AbstractFilter Class

```
<?php
include_once("/class_sax_filters.php");
public class FilterAddStudent extends AbstractFilter
{
   function StartElementHandler($element_name, $attributes)
   {
      // Implementation of the start tag handler
   }
   function EndElementHandler($element_name)
   {
      // Implementation of the end tag handler
   }
   function CharacterDataHandler($cdata)
   {
      // Implementation of the character data handler
   }
}
$xml_parser=new ExpatParser("file.xml");
$f1=new FilterAddStudent();
$f2=new FilterOutput();
$f1->SetListener($f2);
$xml_parser->SetListener(f1);
$xml_parser->parse();
?>
```

The above listing shows that the AbstractFilter class implements the FilterAddStudent filter class. The f2 variable refers to the object of the FilterOutput class that displays the output of the XML document on the Web browser.

◄ PREVIOUS NEXT ►

Using SAX to Parse XML Documents

You can parse the XML document using the two methods, SAX and DOM. When you parse the XML document using DOM, it occupies greater memory space to represent data in a tree structure, which results in slow performance. SAX overcomes the drawbacks of the DOM parser, and provides fast and efficient use of memory by reading the document in chunks. The Expat parser implements the SAX parser.

Using SAX to Modify a Document

You can update XML documents by adding elements and attributes, removing elements, and changing the text of the XML document. For example, an XML document contains information pertaining to the students of a school. You need to add information about the students who have enrolled in a particular semester, and remove information about the students who have left the school.

To modify the information in the XML document, you need to create a filter by implementing the AbstractFilter class.

Listing 2-7 shows how to add information pertaining to a student in the XML document, using SAX:

Listing 2-7: Adding Data in XML Document Using SAX

```
<?php
include_once("/class_sax_filters.php");
class FilterAddStudent extends AbstractFilter
{
   var $studentDetail=Array();
   function AddStudent($id, $name, $age, $address, $standard)
   {
      $detail=Array();
      $detail["id"]=$id;
      $detail["name"]=$name;
      $detail["age"]=$age;
      $detail["address"]=$address;
      $detail["standard"]=$standard;
      $this->studentDetail[]=$detail;
   }
   function StartElementHandler($element_name, $attributes)
   {
      $this->listener->StartElementHandler($element_name, $attributes);
      if($element_name=="STUDENTDETAIL")
      {
         foreach($this->studentDetail as $student)
         {
            $this->listener->StartElementHandler("STUDENT", Array("id"
            =>$student["id"]));
            //For student name
            $this->listener->StartElementHandler("NAME", Array());
            $this->listener->CharacterDataHandler($student["name"]);
            $this->listener->EndElementHandler("NAME");
            //For student age
            $this->listener->StartElementHandler("AGE", Array());
            $this->listener->CharacterDataHandler($student["age"]);
            $this->listener->EndElementHandler("AGE");
            //For student address
            $this->listener->StartElementHandler("ADDRESS", Array());
            $this->listener->CharacterDataHandler($student["address"]);
            $this->listener->EndElementHandler("ADDRESS");
            //For student standard
            $this->listener->StartElementHandler("STANDARD", Array());
            $this->listener->CharacterDataHandler($student["standard"]);
            $this->listener->EndElementHandler("STANDARD");
            $this->listener->EndElementHandler("STUDENT");
         }
      }
   }
   function EndElementHandler($element_name)
   {
      $this->listener->EndElementHandler($element_name);
   }
   function CharacterDataHandler($data)
   {
      $this->listener->CharacterDataHandler($data);
   }
}
$f1 = new ExpatParser("student.xml");
$f1->ParserSetOption(XML_OPTION_CASE_FOLDING, 0);
$f2 = new FilterAddStudent();
$f2->AddStudent("S003", "Jack", "14","New York","9");
$f2->AddStudent("S004", "Jim", "14","New York","9");
$f3 = new FilterOutput();
$f2->SetListener($f3);
$f1->SetListener($f2);
$f1->Parse();
?>
```

In the above listing:

- The FilterAddStudent filter is implemented using the AbstractFilter class.
- The FilterAddStudent filter contains the AddStudent() function that adds records of the students to the XML document.
- The AddStudent() function stores the student details in the StudentDetail array, which is accessed by the event handlers to add new student records in the existing XML document.

Listing 2-8 shows the original XML document:

Listing 2-8: Contents of the XML Document

```
<?xml version="1.0" ?>
<studentdata>
<student>
<name id="s001">George</name>
<age>15</age>
<address>New York</address>
<standard>10</standard>
</student>
<student>
<name id="s001">John</name>
<age>15</age>
<address>New York</address>
<standard>10</standard>
</student>
</studentdata>
```

Figure 2-3 shows the output of Listing 2-7:

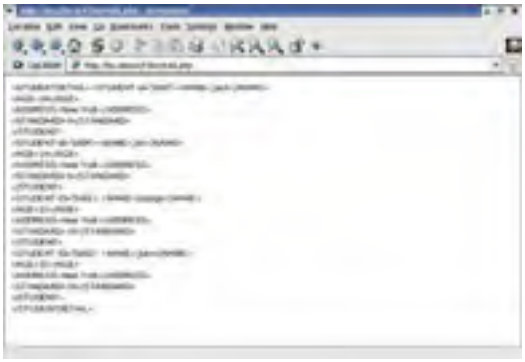


Figure 2-3: Adding Data

Using SAX, you can remove data from the XML document by creating a filter that is implemented by the AbstractFilter class, as shown in Listing 2-9:

Listing 2-9: Removing Data in XML Document Using SAX

```
<?
include_once("/class_sax_filters.php");
class FilterRemoveStudent extends AbstractFilter
{
   var $studentDetail = Array();
   var $flag = 0;
   function RemoveStudent($id)
   {
      $this->studentDetail[] = $id;
   }
   function StartElementHandler($element_name, $attributes)
   {
      if($element_name == "STUDENT")
      {
        if(in_array($attributes["ID"],$this->studentDetail))
         {
            $this->flag = 1;
         }
      }
      if(!$this->flag)
      {
         $this->listener->StartElementHandler($element_name, $attributes);
      }
   }
   function EndElementHandler($element_name)
   {
      if (!$this->flag)
      {
```

```
            $this->listener->EndElementHandler($element_name);
        }
        else
        {
            if($element_name=="STUDENT")
            {
                $this->flag=0;
            }
        }
    }
    function CharacterDataHandler($data)
    {
        if(!$this->flag)
        {
            $this->listener->CharacterDataHandler($data);
        }
    }
}
$fep = new ExpatParser("student.xml");
$frs = new FilterRemoveStudent();
$frs->RemoveStudent("S001");
$fou = new FilterOutput();
$frs->SetListener($fou);
$fep->SetListener($frs);
$fep->Parse();
?>
```

In the above listing:

- The FilterRemoveStudent filter is implemented using the AbstractFilter class.
- The FilterRemoveStudent filter contains the RemoveStudent() function that removes the specified student's records from the XML document.
- The RemoveStudent() function accepts the student id attribute as an argument. The RemoveStudent() function stores student ID in the StudentDetail variable, which is accessed by the event handlers to remove a student record from the existing XML document.

Figure 2-4 shows the output of Listing 2-9:

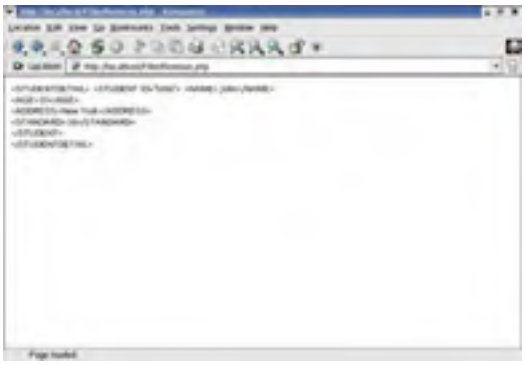


Figure 2-4: Removing Data

Using SAX to Query a Document

Querying a document refers to the process of traversing the XML document and searching for specific information. For example, an XML document contains information pertaining to the students of a school. You can query the XML document to search for the students who live in New York City.

Using SAX, you can query an XML document by creating a filter that is implemented by AbstractFilter, as shown in Listing 2-10:

Listing 2-10: Querying an XML Document Using SAX

```
<?php
include_once("/class_sax_filters.php");
class FilterQueryStudent extends AbstractFilter
{
    var $detail = Array();
    var $stud = Array();
    var $st;
    function RetrieveDetails()
    {
        return $this->detail;
    }
    function StartElementHandler($element_name, $attributes)
    {
        $this->st='';
        if ($element_name <> "STUDENTDETAIL" && $element_name <> "STUDENT")
    {
        $this->st=$element_name;
```

```
      $this->stud[$this->st]='';
   }
}
function EndElementHandler($element_name)
{
   if($element_name == "STUDENT")
   {
      if(((trim($this->stud["ADDRESS"]))=="New York") &&
      (trim($this->stud["AGE"])>=15))
      {
         $this->detail[]=$this->stud;
      }
      $this->stud = Array();
      }
   }
   function CharacterDataHandler($data)
   {
      if($this->st)
      {
         if(!empty($data))
         {
            $this->stud[$this->st] .= $data;
         }
      }
   }
}
$tests=Array();
$f1 = new ExpatParser("student.xml");
$f2 = new FilterQueryStudent();
$f1->SetListener($f2);
$f1->Parse();
$tests=$f1->listener->RetrieveDetails();
$num=count($tests);
for($i=0;$i<$num;$i++)
{
   foreach($tests[$i] as $key => $value)
   {
      print "<b>$key=$value</b><br/>";
   }
}
?>
```

In the above listing:

- The AbstractFilter class implements the FilterQueryStudent filter.
- The FilterQueryStudent filter contains the RetrieveDetails() function, which retrieves the records of students that satisfy the specified condition.
- The EndElementHandler() function checks if the address of the student is New York and the age of the student is greater than or equal to 15.

Figure 2-5 shows the output of Listing 2-10:

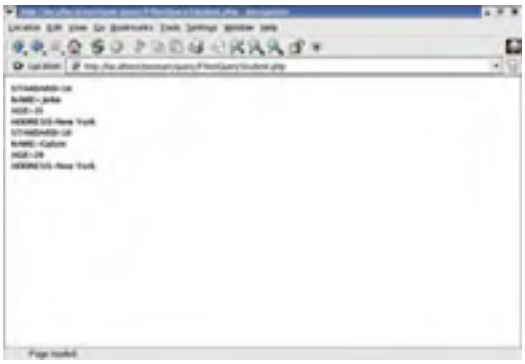


Figure 2-5: Output of Querying a Document

Using SAX to Generate an XML Document

Using SAX, you can generate an XML document from various sources, such as database, text file, and objects. You need to create a parser by implementing the AbstractSAXFilter class for generating the XML document.

For example, you can use the content of the student.txt text file to write an XML document. The student.txt file is shown below:

```
George, English
John, French
Jack, Physics
Jim, French
Jack, English
Jim, Physics
George, Physics
John, Physics
```

Convert the student.txt text file into an XML document, using SAX, as shown in Listing 2-11:

Listing 2-11: Generating XML Data from a Text File

```
<?php
include_once("/class_sax_filters.php");
class StudentParser extends AbstractSAXParser
{
   var $studentDetail = Array();
   function StudentParser($textfile)
   {
      $xml_parser=new ParseStudent();
      $xml_parser->ParseTextFile($textfile);
      $this->studentDetail=$xml_parser->GetStudents();
   }
   function Parse()
   {
      $this->StartElementHandler($this,"STUDENTDETAIL",Array());
      foreach($this->studentDetail as $student => $subjects)
      {
         $this->StartElementHandler($this, "STUDENT",Array());
         $this->StartElementHandler($this, "NAME",Array());
         $this->CharacterDataHandler($this, XmlEntities($student));
         $this->EndElementHandler($this,"NAME");
         $this->StartElementHandler($this, "SUBJECTS",Array());
         foreach ($subjects as $subject)
         {
            $this->StartElementHandler($this, "SUBJECT",Array());
            $this->CharacterDataHandler($this, XmlEntities($subject));
            $this->EndElementHandler($this,"SUBJECT");
         }
         $this->EndElementHandler($this, "SUBJECTS");
         $this->EndElementHandler($this, "STUDENT");
      }
      $this->EndElementHandler($this,"STUDENTDETAIL");
   }
}
// Class that writes the XML data manually.
class ParseStudent
{
   var $studentDetail = Array();
   function ParseTextFile($textfile)
   {
      $fp=fopen($textfile,"r");
      if (!$fp)
      {
         return 0;
      }
      $text=fread($fp, filesize($textfile));
      return $this->ParseText($text);
   }
   function GetStudents()
   {
      return $this->studentDetail;
   }
   function ParseText($text)
   {
      $entries = Array();
      $entry = Array();
      $entries = explode("\n",$text);
      foreach( $entries as $entry)
      {
         $entry = chop($entry);
         $entry = explode(",", $entry);
         if (!isset($this->studentDetail[$entry[0]]))
         {
            $this->studentDetail[$entry[0]]=Array();
         }
         $this->studentDetail[$entry[0]][]=$entry[1];
      }
      return 1;
   }
}
// Function that overwrites the special character with the XML entities.
function XmlEntities($data)
{
   $pos=0;
   $len=strlen($data);
```

```
    $escdata ="";
    for(;$pos<$len;)
    {
       $char=substr($data,$pos,1);
       $num = Ord($char);
       switch($num)
       {
          case 34:
          $char= "";
          break;
          case 38:
          $char = "&";
          break;
          case 39:
          $char = "&apos;";
          break;
          case 60:
          $char = "<";
          break;
          case 62:
          $char = ">";
          break;
          default:
          if ($num < 32)
          $char = ("&#".strval($num).";");
          break;
      }
       $escdata .= $char;
       $pos++;
    }
    return $escdata;
}
$f1=new StudentParser("student.txt");
$f2=new FilterOutput();
$f1->SetListener($f2);
$f1->Parse();
?>
```

In the above listing:

- The parser, StudentParser, parses the student.txt text file, using the ParseStudent parsing class.
- The parsing class reads the text file and stores the content according to the number of lines in the text file.
- The chop() function stores the content of text file in the entry array, and the explode() function splits the text file using a string delimiter.
- The comma delimiter divides each line into two words, and stores these words in the studentDetail array.

Figure 2-6 shows the output of Listing 2-11:

Figure 2-6: Output of Listing 2-11

Note You can create a well-formed XML document by encoding the required functions.

Using SAX to Create PHP Objects from XML

Using SAX, you can create PHP objects from an XML document. The SAX parser creates objects corresponding to each element in the XML document. These objects are represented as PHP objects using the SAX filters, and their references are stored in an array. For example, if an XML document contains information pertaining to students; it can be represented as the student object.

Listing 2-12 shows the content of the XML file that contains information pertaining to students:

Listing 2-12: Content of the XML File

```
<?xml version="1.0"?>
<studentdetail>
<student>
<name>George</name>
<marks>75</marks>
</student>
<student>
<name>John</name>
<marks>85</marks>
</student>
</studentdetail>
```

The above listing shows the elements of the XML document that are converted to the PHP objects.

The parser parses the XML document, finds the <student> tag, creates the student object, and stores its reference in an array. When the parser finds the <name> and <marks> tags, it assigns the value to the name and marks property of the student objects respectively.

Listing 2-13 shows how to create PHP objects from the XML document:

Listing 2-13: Creating PHP Objects from XML Document

```
<?php
include_once("/class_sax_filters.php");
class Student
{
   var $name;
   var $marks;
   function GetName()
   {
      return $this->name;
   }
   function GetMarks()
   {
      return $this->marks;
   }
   function SetName($name)
   {
      $this->name=$name;
   }
   function SetMarks($marks)
   {
      $this->marks=$marks;
   }
}

class FilterNull extends AbstractFilter
{
}
class FilterStudentObject extends AbstractFilter
{
   var $count=0;
   var $student;
   var $sub_element=0;
   var $studentDetail=Array();
   function GetStudents()
   {
      return $this->studentDetail;
   }
   function StartElementHandler($element_name, $attributes)
   {
      if($element_name=="STUDENT")
      {
      $this->studentDetail[]=new Student();
      $this->count=1;
      }
      else
      {
         if($this->count)
         {
            $this->sub_element=$element_name;
         }
      }
      $this->listener->StartElementHandler($element_name, $attributes);
   }
   function EndElementHandler($element_name)
   {
      $this->sub_element=0;
      if($element_name == "STUDENT")
      {
         $this->count=0;
      }
      $this->listener->EndElementHandler($element_name);
   }
```

```
    function CharacterDataHandler($cdata)
    {
       if($this->count && $this->sub_element)
       {
        $getMethod="get".strtoupper(substr($this->sub_element,0,1)).substr
        ($this->sub_element,1);($this->sub_element,0,1)).substr($this->sub_element,1);
        $sub=$this->studentDetail[count($this->studentDetail)-1]->$getMethod();
        $this->studentDetail[count($this->studentDetail)-1]->$setMethod($sub.$cdata);
       }
       $this->listener->CharacterDataHandler($cdata);
    }
}
$f1=new ExpatParser("student.xml");
$f2=new FilterStudentObject();
$f3=new FilterNull();
$f2->SetListener($f3);
$f1->SetListener($f2);
$f1->Parse();
$studentDetail=$f1->listener->GetStudents();
print_r($studentDetail);
$num=count($studentDetail);
print "<br/>";
for($i=0;$i<$num;$i++)
{
   foreach($studentDetail[$i] as $key => $value)
   {
      print "<b>$key=$value</b></br/>";
   }
}
?>
```

In the above listing:

- The parser creates the objects of the student element and stores them in the studentDetail array.
- The Student class creates the functions for retrieving the XML data. The StudentCreateObject filter parses the XML document and creates the objects corresponding to each element.
- The FilterNull class is an empty user-defined class that does not contain any function. This class specifies that the output of the StudentCreateObject filter is not displayed in the XML format.

Figure 2-7 shows the output of Listing 2-13:

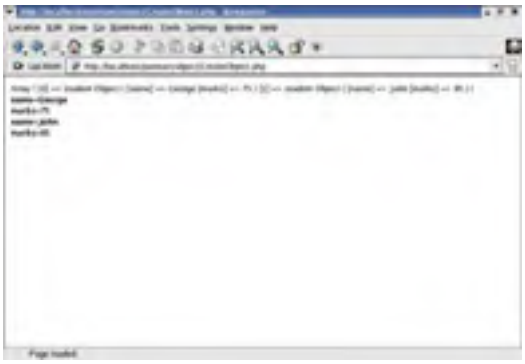


Figure 2-7: Creating PHP Objects

Note The print_r() function displays how the array elements are stored in the array.

◄ PREVIOUS NEXT ►

Using Object-Oriented Frameworks

You can create Web applications using the object-oriented framework in PHP. PHP supports object-oriented programming using classes and objects. In the object-oriented approach, you can use inheritance to change the functionality of the existing class. There are two types of object-oriented frameworks that PHP script can use, extremePHP and SAXParser.

The extremePHP Framework

eXtremePHP (xpl) is a set of object-oriented libraries that helps you create dynamic Web applications. eXtremePHP provides object-oriented framework to implement SAXParser. eXtremePHP supports all features of object-oriented programming, and provides functionality, such as:

- Input/output streams and socket connections
- xpl framework that simplifies the use of HTML tags
- HTML tag classes, form handling, and layout managers
- Utility classes, such as Strings, Dates, Vectors, and Iterators
- Database access
- XML parsing with SAX and DOM processor
- Portable Document Format (PDF) library that supports XML

The minimum requirement for installing the eXtremePHP framework is a Web server running on PHP4. You can download the current version of the framework from the Web site, http://sourceforge.net/projects/extremephp/. The Web site contains the zip version and tar ball of the software.

To implement the eXtremePHP framework in the PHP script, you need to install the eXtremePHP framework in a Linux environment. To install the eXtremePHP framework:

1. Copy the tar file in the document root directory of the Apache Web server using the command:
   ```
   cp eXtremePHP-0.15a.tar.gz /var/www/html
   ```
   The default document root directory of the Apache Web server is /var/www/html.
2. Change the current directory to the document root directory using the command:
   ```
   cd /var/www/html
   ```
3. Untar the tar file of the eXtremePHP framework using the command:
   ```
   gunzip eXtremePHP-0.15a.tar.gz
   tar xvf eXtremePHP-0.15a.tar
   ```
   The above command uncompresses the tar file and creates the xpl directory under the document root directory of the Web server. The xpl directory stores all the class libraries and frameworks of eXtremePHP.
4. Include the /xpl/common/common.inc.php file in the /etc/php.ini file by entering the following command:
   ```
   auto_prepend_file="/var/www/html/common/common.inc.php"
   ```
   The above command automatically includes the common.inc.php file in a PHP document.
5. Create an .htaccess file in the /var/www/html directory, and store the following command in this file:
   ```
   php_value auto_prepend_file /www/var/html/xpl/common/common.inc.php
   ```

The SaxParser Framework

The SAXParser framework helps you create an object-oriented parser to read an XML document and display them in Web browser when the eXtremePHP framework is installed. The SAXParser class of the SAXParser framework provides three functions, StartingTagHandler, EndingTagHandler, and CharacterDataHandler, to handle an XML document. The SAXParser class invokes these functions while parsing the document.

Figure 2-8 shows the SAXParser framework:

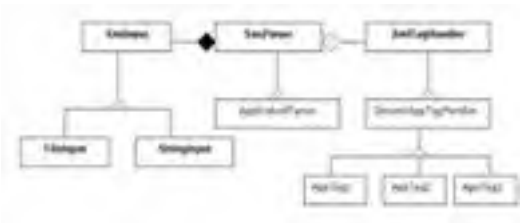


Figure 2-8: The SAXParser Framework

To create a SAX parser, you need to define two new classes, the ApplicationParser class and the GenericAppTagHandler class. The ApplicationParser class inherits the SAXParser class and the GenericAppTagHandler class inherits the XmlTagHandler class. The GenericAppTagHandler class adds tag handlers to the parser, and provides interfaces to global variables required by the tag handlers.

The SAXParser class is based on the white box model and should not be used as a standalone class. The SAXParser class determines the object to be used when handling parsed data. This object-oriented approach offers the advantage of grouping tag-specific logic in a single component. You need to create a base tag handler that is inherited by other application element handlers for grouping them in a single component.

Note The white box model deals with the internal working and the functionality of the SAXParser class.

The XmlTagHandler class contains the common functionality between various handler classes, such as GenericAppTagHandler class and its subclasses. For example, all the element handlers provide HTML tags as the output during the begin and end element handlers. The constructor of the SAX parser accepts a reference of the XMLInput class that specifies the name of the file to be parsed. The XMLInput class specifies the source of the input, which can be a file or string.

You can override the constructor of the SAXParser class by defining a new functionality to the constructor. You need to pass the object of the XMLInput class as an argument to the constructor for defining the new functionality. The code to create the constructor is:

```
function StudentParser($file)
{
   SAXParser::SAXParser(new FileInput($file));
}
```

The above code shows that the constructor of the base class is overridden in the constructor of the derived class.

You can define the handlers using the hook up methods, which the parser invokes automatically. The addHandlers() method is the hook up method of the SAXParser class. A derived class of the SAXParser class implements the hook up method.

If you create a new handler, it must inherit the XmlTagHandler class defined in the API of the SAXParser framework.

PREVIOUS NEXT

Chapter 3: PHP and Document Object Model

Download CD Content

Using the tree-based Document Object Model (DOM) approach to parse XML data, PHP loads the complete XML document in the system memory and provides standard classes to navigate and manipulate the XML document. The DOM implementation in PHP is based on the libxml library, which contains various functions that provide parsing capabilities to modify the XML document.

This chapter explains how to implement DOM in PHP to parse XML documents. This chapter also provides a comparison between the event-based, Simple API for XML (SAX) approach, and the tree-based DOM approach to parse XML documents. Finally, this chapter explains the DOM architecture, standard DOM classes, and the uses of the DOM approach to modify and query XML documents.

Introducing DOM

DOM is a programming interface that you can use to create and edit XML documents. It specifies the logical structure of the document that you can use to add, delete, or modify XML documents.

DOM parses the XML document by creating a hierarchical tree structure of standard objects. Each object encapsulates the methods and properties that you can use to manipulate and navigate the DOM tree. For example, the DomAttribute object encapsulates attribute properties, such as name and value. The basic interface in every DOM object is a node. The tree structure contains multiple nodes that are related to each other in a parent-child relationship.

DOM is platform-independent, and can interact with both HTML and XML documents. You can implement the DOM approach with languages, such as PHP, Java, Python, Visual Basic, Perl, and Delphi.

The W3C has defined DOM specifications in multiple levels. The various levels of DOM specifications are:

- DOM Level 1: Contains the core features of DOM developed by the W3C in October 1998.
- DOM Level 2: Contains DOM features, such as core functions, document traversal, and event handling.
- DOM Level 3: Incorporates features, such as XPath and abstract schemas.

A DOM parser reads the entire XML document into memory, converts it into a hierarchical tree structure, and provides an API to access tree nodes and the contents attached to each node. For example, the DOM parser can represent the contents of emp.xml that stores employee information, such as company name and department name in a hierarchical structure.

Listing 3-1 shows how to create the emp.xml document:

Listing 3-1: Creating an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<Company name="Unique Systems">
<Department>Marketing</Department>
<Level>Middle-Level</Level>
<EmployeeInformation>
<Name first="John" middle="E" last="Williams"/>
<DateOfHiring>10/01/1982</DateOfHiring>
</EmployeeInformation>
</Company>
```

The above listing creates an XML document, emp.xml, which stores employee information, such as employee names, company names, department names, and hiring dates.

The DOM tree contains various nodes interlinked to each other with the parent-child relationship. For example, in the emp.xml document, the Marketing node is the child node of the Department node, and Name is the child node of the Employee Information node. You can parse this XML document into a hierarchical structure using the DOM parser.

Figure 3-1 shows a representation of the emp.xml file as a DOM tree structure generated by the DOM parser:

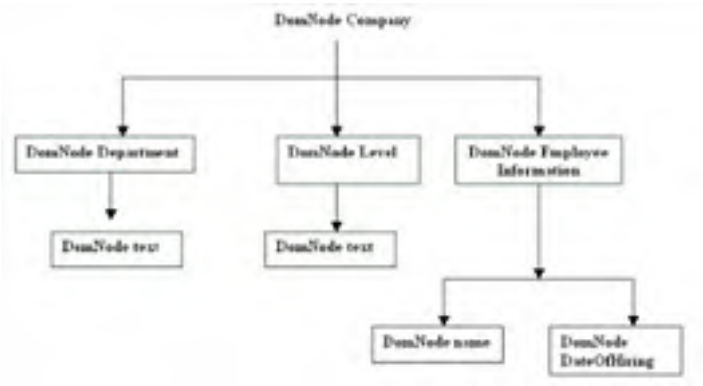


Figure 3-1: DOM Tree Representation of emp.xml

Comparing DOM with SAX

You can parse an XML document using both the DOM and SAX parsers. Both the DOM and SAX approaches have their advantages and disadvantages when working with XML documents.

Table 3-1 lists comparisons between the SAX and DOM parsing approaches:

Table 3-1: Comparison between DOM and SAX

Mode of Comparison	DOM	SAX
Approach	Uses a tree-based approach to parse XML documents. The DOM approach loads the whole XML document in the memory as a tree structure that contains various nodes.	Uses an event-based approach that processes the XML files in a linear manner. The SAX approach parses the XML documents in chunks without creating a DOM tree.
Memory	Consumes system memory as the DOM parser loads the whole XML document in the memory.	Consumes less system memory in comparison to DOM, as it does not load the whole XML structure in memory and parses the XML document in chunks.
Efficiency	Lets you parse XML documents in a non-sequential manner and manipulate complex XML documents, as the DOM parser stores information about how the nodes of a tree are related to each other.	Faster than the DOM approach in parsing simple XML documents, as it processes the XML documents in a sequential manner

The DOM Architecture

The DOM architecture contains modules, where each module defines various DOM API domains, such as XML, HTML, tree events, and Cascading Style Sheets (CSS).

Figure 3-2 shows the block diagram of the DOM architecture:

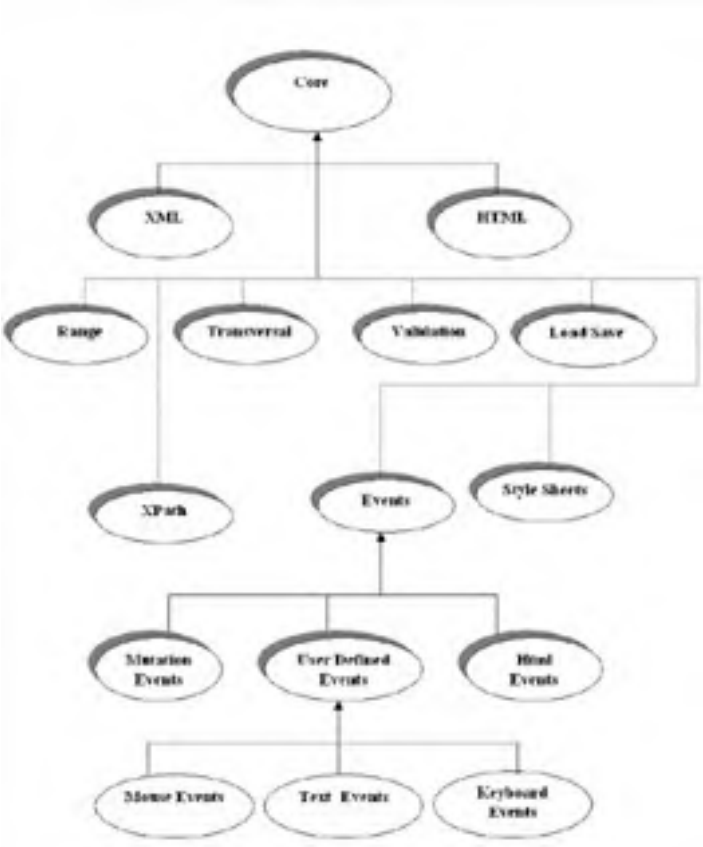


Figure 3-2: The DOM Architecture

The various modules in the DOM architecture are:

- Core: Represents the internal tree-like structure of the document and enables you to move through the hierarchy of the DOM tree elements.

- XML: Provides interface-processing instructions to the DOM parser, such as entities and CDATA.
- HTML: Manipulates HTML documents.
- Events: Defines events that perform XML-tree manipulation.
- Cascading Style Sheets: Manipulates CSS style sheets.
- Load and Save: Provides various parameters that control the load and save operations of XML documents. The load module lets you load the XML document into the DOM tree and save the DOM tree into an XML document.
- Validation: Provides various methods, such as cansetAttribute and canInsertAttribute, which provide validation checks on the DOM documents.
- Xpath: Provides methods to query a DOM tree.

◄ PREVIOUS NEXT ►

Using DOM Classes

DOM provides various built-in classes that let you parse the XML documents in PHP. The PHP DOM parser represents the XML document by creating standard objects that are instances of the following classes:

- DomDocument Class
- DomNode Class
- DomElement Class
- DomText Class
- DomAttribute Class

You can add, delete, or modify the structured content of the XML document by using various properties and methods. The DomDocument object stores the properties of an XML document, such as its name and version.

Listing 3-2 shows the contents of the DOM.xml file:

Listing 3-2: Properties of an XML File

```
DomDocument Object
(
    [name] => #document
    [url] => DOM.xml
    [version] => 1.0
    [encoding] => UTF-8
    [standalone] => -1
    [type] => 9
    [compression] => -1
    [charset] => 1
    [0] => 1
    [1] => 136451608
    [doc] => Resource id #0
)
```

The above listing specifies the values of various properties of the DOM.xml file. You can display the values of the properties of the XML document using the print_r() function. The listing specifies values, such as the version, node type, and character set of the XML document. It also specifies whether the DOM.xml file is compressed or not.

The DomDocument Class

The DomDocument object is the object created after an XML document is created by the DOM parser.

Listing 3-3 shows the structure of the DomDocument class:

Listing 3-3: Structure of the DomDocument Class

```
class DomDocument
{
   Properties:
   version
   encoding
   standalone
   type
   Methods:
   root()
   children()
   add_root( $node )
   dtd()
   dumpmem()
}
```

The above listing shows the structure of the DomDocument class, which contains various properties and methods defined by the DomDocument class.

The properties defined in the DomDocument class specify the version of the XML document, the text encoding, and the type of the document. The standalone property of the DomDocument object specifies the Boolean value whether or not the document is a standalone document. The methods provided by the DomDocument class are:

- `root()`: Returns the root element.
- `children()`: Returns the child node of the document.
- `addroot()`: Creates a new document element and returns the DomElement object.
- `dtd()`: Returns the Document Type Definition (DTD) object. The DTD object contains the basic information of the document and encapsulates properties, such as systemId and name. The systemId property contains the file name of the DTD document.

- `dumpmem()`: Stores the XML structure in a string variable.

Listing 3-4 shows how to parse an XML string using the xmldoc() function:

Listing 3-4: Parsing the XML string

```
<?php
$xml_str = "<?xml version='1.0'?><name>John</name>";
if (!$doc=xmldoc($xml_str))
{
  die("Error in XML");
}
else if ($doc->version > 1.0)
{
   die("Unsupported XML version");
}
else
{
   echo "No Error";
}
?>
```

The above listing parses the string, John, using the xmldoc() function. The xmldoc() function accepts the XML string as an argument and generates the DOM object that represents the XML data.

You can also create the DomDocument object of an XML file. The syntax to create the DomDocument object using the xmldocfile() function is:

```
$doc = xmldocfile("<filename>");
```

In the above code, the xmldocfile() function creates an object of the DomDocument class that represents the XML file.

Listing 3-5 shows how to parse an XML file using the xmldocfile() function:

Listing 3-5: Parsing XML File

```
<?php
$xml_str = "/var/www/html/vishi/test.xml";
if (!$doc = xmldocfile($xml_str))
{
   die("Error in XML");
}
else if ($doc->version > 1.0)
{
   die("Unsupported XML version");
}
else
{
   echo "No Error in Parsing the XML file";
}
?>
```

The above listing parses the xml file, test.xml, using the xmldocfile() function.

The DomNode Class

The DomNode class contains properties, such as name, content, and type of the node. The name property specifies the tag name of the node. The content property specifies the contents stored in the node. The type property represents the integer that refers to the object type. The DomNode class also encapsulates various methods, such as the lastchild(), children(), and parent() methods.

Listing 3-6 shows the structure of the DomNode class:

Listing 3-6: Structure of the DomNode Class

```
class DomNode
{
   properties:
   name
   content
   type
   methods:
   lastchild()
   children()
   parent()
   new_child( $name,$content )
   getattr( $name )
   setattr( $name,$value )
   attributes()
}
```

In the above listing, the structure of the DomNode class is defined. The listing defines various properties and methods that are encapsulated in the DomNode class. The methods defined in the DomNode class are:

- `lastchild()`: Returns the last child node for the node.

- `parent()`: Returns the parent node.
- `children()`: Returns an array of child nodes.
- `new_child()`: Adds a new DomNode to its children.
- `attributes()`: Returns the array of the DomAttribute objects.
- `getattr()`: Retrieves the attributes from the XML document.
- `setattr()`: Sets the attribute value.

The DomElement Class

The DomElement class defines elements of the XML document. A DomElement object represents the name and type of an element of the XML document. The methods contained in the DomElement class are:

- `tagname()`: Returns the element name. The syntax is:
  ```
  string DomElementName->tagname(void);
  ```
- `get_attribute()`: Returns the name and value of the attribute in the element. If no attribute is specified with the method, it returns an empty string. The syntax of the get_attribute() method is:
  ```
  objectDomElement ->get_attribute(string);
  ```
- `set_attribute()`: Adds a new attribute to the node. The syntax of the set_attribute() method is:
  ```
  bool DomElement->set_attribute ( string name, string value)
  ```
- `remove_attribute()`: Removes an attribute from the element structure. The syntax of the remove_attribute() method is:
  ```
  bool DomElement->remove_attribute (string);
  ```
- `children()`: Returns an array of the DomElement object containing the child nodes of the element.
- `parent()`: Returns a DomElement object that is the parent of a node.
- `last_child()`: Returns the last entry of a node child. If no last child is found in the DOM tree, a NULL value is returned.
- `attributes()`: Returns an array of the DomAttribute objects that represent the attributes of a node.

You can distinguish the node type using the type property that contains specific integer values for each node.

Table 3-2 lists various predefined node types that identify the nodes:

Table 3-2: DOM Node Types

Node Type	Specification	Integer Value
XML_ELEMENT_NODE	Specifies an element.	1
XML_ATTRIBUTE_NODE	Specifies an attribute.	2
XML_TEXT_NODE	Specifies text.	3
XML_ENTITY_REF_NODE	Represents an entity reference.	5
XML_PI_NODE	Represents a processing instruction.	7
XML_COMMENT_NODE	Represents a comment.	8
XML_DOCUMENT_NODE	Represents the XML document.	9
XML_NOTATION_NODE	Represents the notation node.	12
XML_CDATA_SECTION_NODE	Represents the cdata section.	4

You can identify the elements of the XML documents using the methods of the DomElement class. You can also count the number of nodes or elements contained in an XML document.

Listing 3-7 shows how to create an XML file that stores employee information:

Listing 3-7: Creating an XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<EMPLOYEEINFORMATION>
<EMPLOYEE>
<NAME ID="E001">George</NAME>
<AGE>35</AGE>
<DEPARTMENT>RESEARCH AND DEVELOPMENT</DEPARTMENT>
<DESIGNATION>BRANCH MANAGER</DESIGNATION>
</EMPLOYEE>
<EMPLOYEE>
<NAME ID="E002">John</NAME>
<AGE>45</AGE>
<DEPARTMENT>HUMAN RESOURCE</DEPARTMENT>
<DESIGNATION>MANAGER</DESIGNATION>
</EMPLOYEE>
```

```
</EMPLOYEEINFORMATION>
```

The above listing shows how to create an XML file containing employee information. Employee information, such as employee name, employee ID, and employee designation, is stored in the emp.xml file. You can generate the DOM tree using the DOM parser that represents the structure of the XML document.

Listing 3-8 shows how to use the classElement objects to generate the structure of the emp.xml file:

Listing 3-8: Structuring an XML Document

```
<?php
$xmlfile = "emp.xml";
if (!$doc = xmldocfile($xmlfile))
{
   die("Error in XML document");
}
$root = $doc->root();
$children = getChildren($root);
$count = 1;
printTree($children);
function printTree($nodeCollection)
{
   global $count;
   echo "<ul>";
   for ($t=0; $t<sizeof($nodeCollection); $t++)
   {
      $count++;
      echo "<li>". $nodeCollection[$t]->tagname;
      $nextCollection = getChildren($nodeCollection[$t]);
      printTree($nextCollection);
   }
   echo "</ul>";
}
function getChildren($node)
{
   $temp1 = $node->children();
   $store = array();
   for ($t=0; $t<sizeof($temp1); $t++)
   {
      if ($temp1[$t]->type == XML_ELEMENT_NODE)
      {
         $store[] = $temp1[$t];
      }
   }
   return $store;
}
echo "Total number of elements in document: $count";
?>
```

In the above listing, the DOM tree of the emp.xml file is generated. The root node is obtained using the root() method. After retrieving the root node of the document, the listing generates other node elements and attributes. The total number of elements in the XML document is also calculated in the above listing. Figure 3-3 shows the output of Listing 3-8:

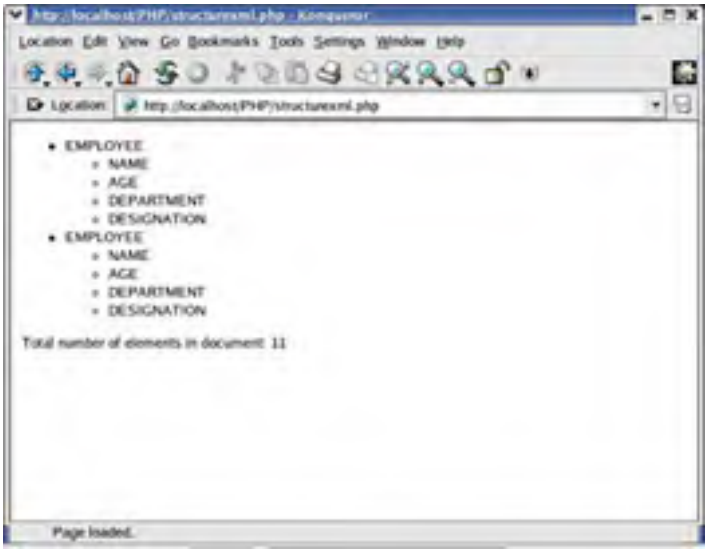


Figure 3-3: The DOM Tree

The DomText Class

You can represent the character data in an XML document using the DomText class. The syntax of the DomText class is:

```
DomText Object
{
   [type]
   [content];
}
```

In the above syntax, the type property specifies the node type, such as XML_TEXT_NODE. You can specify either the node number or the node name with the type. The content property stores the character data. The DOM parser identifies the whitespaces as character data and creates DomText objects for the whitespaces.

Listing 3-9 shows how to retrieve the character data from an XML document using the DomText object:

Listing 3-9: Using the DomText Object

```
<?php
$xml_str = "<?xml version='1.0'?>
<STUDENT>
<NAME>John</NAME>
<NAME>Ronan</NAME>
<ID>OO2</ID>
<NAME>Daniel</NAME>
<ID>003</ID>
<NAME>Mark</NAME>
<ID>004</ID>
<NAME>William</NAME>
<ID>005</ID>
</STUDENT>
";
$data = array();
if(!$doc = xmldoc($xml_str))
{
 die("Error parsing XML");
}
$root = $doc->root();
$nodes = $root->children();
foreach ($nodes as $t)
{
   $text = $t->children();
   if($text[0]->type ==XML_TEXT_NODE)
   {
       if ($text[0]->content != "")
       {
         $data[] = $text[0]->content;
       }
   }
}
print_r($data);
?>
```

The above listing retrieves the text data from an XML document. The DOM parser creates an array to hold the name and the ID of the students. The foreach loop iterates the <STUDENT> elements and adds the character data found while traversing through the DOM tree in the, $data array. Figure 3-4 shows the output of Listing 3-9:

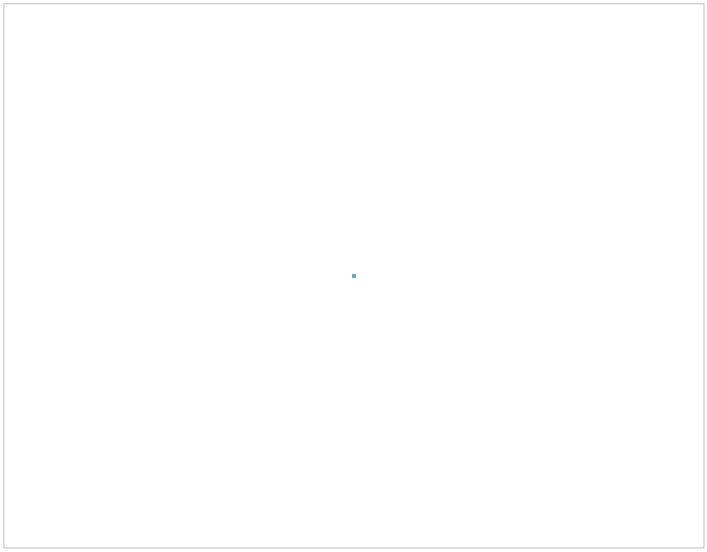


Figure 3-4: The Retrieved Character Data

The DomAttribute Class

The DomAttribute class creates the attr object that returns the attribute of an element.

Listing 3-10 shows the structure of the DomAttribute class:

Listing 3-10: Structure of the DomAttribute Class

```
class DomAttribute
{
   Properties
   name
   value
   Method
   name()
}
```

The above listing specifies the method and properties of the DomAttribute class. The name property sets the attribute name. The value property specifies the value stored in the attribute node.

Listing 3-11 explains the DomAttribute class:

Listing 3-11: Using the DomAttribute Class

```
<?php
$xml_string = "<?xml version='1.0'?>
<EmployeeInfo>
<NAME ID='E001'>George</NAME>,
<NAME ID='E002'>John</NAME>,
<NAME ID='E003'>Angel</NAME>
</EmployeeInfo>";
if (!$doc = xmldoc($xml_string))
{
   die("Error in XML document");
}
// get the root node
$root = $doc->root();
// get its children
$children = $root->children();
// Iterate through child list
for ($x=0; $x<sizeof($children); $x++)
{
   if($children[$x]->type == XML_ELEMENT_NODE)
   {
      // Retrieving the text
      $text = $children[$x]->children();
      $cdata = $text[0]->content;
      if($children[$x]->get_attribute("ID"))
      {
         echo "<br><Employee=" . $children[$x]->get_attribute("ID").">" .
         $cdata .
         "</Employee></br>";
      }
      else
      {
         echo $cdata;
      }
   }
   // if text node
   else if ($children[$x]->type == XML_TEXT_NODE)
   {
      // simply print the content
      echo $children[$x]->content;
   }
}
?>
```

The above listing shows how to implement the DomAttribute class. In the above listing the get_attribute() function retrieves the attribute value of the attribute nodes of the DOM tree. The above listing parses the XML document and then identifies the root and child nodes of the XML document. The DOM parser iterates through the child nodes and verifies the presence of the element node in the DOM tree. The listing retrieves the text node and the contents stored in it. Figure 3-5 shows the output of Listing 3-11:

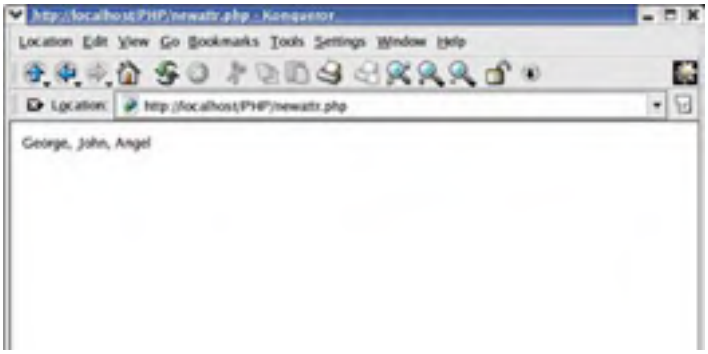


Figure 3-5: Output of using the DomAttribute Class

PREVIOUS NEXT

Using DOM to Manage XML Documents

Using DOM you can manage an XML document by creating a DOM tree of the XML document containing standard objects. You can modify and traverse the DOM tree, and access the attributes, data, and elements of the DOM tree. You can modify the XML document by modifying the properties and methods of the objects. The DOM approach also lets you query XML documents to retrieve specific information. In addition to parsing the XML documents, the DOM approach lets you create XML documents from scratch. Methods such as add_root() and new_child() let you create nodes of DOM tree.

Using DOM to Modify a Document

You can modify the XML document using the DOM approach. Modifying a document includes various tasks, such as adding a new element, modifying the attribute value, or removing an element from the document tree. The DOM approach occupies a large amount of memory to parse documents. As a result, you can modify an XML file which is smaller in size using the DOM approach and the large XML documents using the SAX approach.

To understand how to modify XML documents using DOM, consider an XML file that stores information about a bookstore-shopping cart application, as shown in Listing 3-12:

Listing 3-12: Creating the bookcart.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<Bookcart>
<customer CID="354">
<name>John Anderson</name>
</customer>
<BookInfo>
<book ID="1001">
<bookName>XML Bible</bookName>
<author>Elliotte Rusty Harold</author>
<quantity>2</quantity>
<list_price>33</list_price>
</book>
<book ID="1002">
<bookName>XML in a Nutshell</bookName>
<author>Elliotte Rusty Harold</author>
<quantity>3</quantity>
<list_price>37</list_price>
</book>
<book ID="1003">
<bookName>Advanced PHP Programming</bookName>
<author>George Schlossnagle</author>
<quantity>5</quantity>
<list_price>40</list_price>
</book>
</BookInfo>
</Bookcart>
```

The above listing creates an XML file that stores information about the books that are available at an online bookstore. The XML file stores information, such as book titles, book IDs, author names, quantities available, and list prices.

Listing 3-13 shows how to add information on books available in the bookcart.xml file using the DOM approach:

Listing 3-13: Adding Book to the Shopping Cart

```
<?php
$xmlfile = "bookcart.xml";
$modfile = AddProduct($xmlfile, "1004", "XML and PHP", "William", "4", "30");
print($modfile);
function AddProduct($xml, $ID, $bookName, $author, $quantity, $list_price)
{
   $doc = xmldocfile($xml);
   $root = $doc->root();
   $children = $root->children();
   foreach ($children as $child)
   {
      if ($child->node_type() == XML_ELEMENT_NODE)
      {
         if ($child->tagname() == "BookInfo") {
         $newbook = $doc-create_element("book");
         $newbook->set_attribute("bID", $ID);
         $nname = $doc-create_element("bookName");
         $nametext = $doc-create_text_node($bookName);
         $nname->add_child($nametext);
         $nprice = $doc-create_element("list_price");
         $pricetext = $doc-create_text_node($list_price);
         $nprice->add_child($pricetext);
         $nauthor = $doc-create_element("author");
         $authortext = $doc-create_text_node($author);
         $nauthor->add_child($authortext);
         $nquantity = $doc-create_element("quantity");
         $quantitytext = $doc-create_text_node($quantity);
         $nquantity->add_child($quantitytext);
```

```
            $newbook->add_child($nname);
            $newbook->add_child($nprice);
            $newbook->add_child($nauthor);
            $newbook->add_child($nquantity);
            $child->add_child($newbook);
         }
      }
}
$xml = $doc->dump_mem();
$fp = fopen("/var/www/html/add.xml", "w+");
fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
return $xml;
}
?>
```

The above listing lets you add a new book in the online shopping cart.

You need to pass book details, such as book titles, authors, prices, and quantities as arguments to the AddProduct() function. The cart.xml file is parsed using the xmldocfile() function, and the root and child nodes are retrieved from the document. The AddProduct() function verifies the tag name property of the element node and uses the add_child() function to add new child nodes and attributes to the DOM tree. In the above listing, the dumpmem() method dumps the modified XML document into a string. Figure 3-6 shows the output of Listing 3-13:

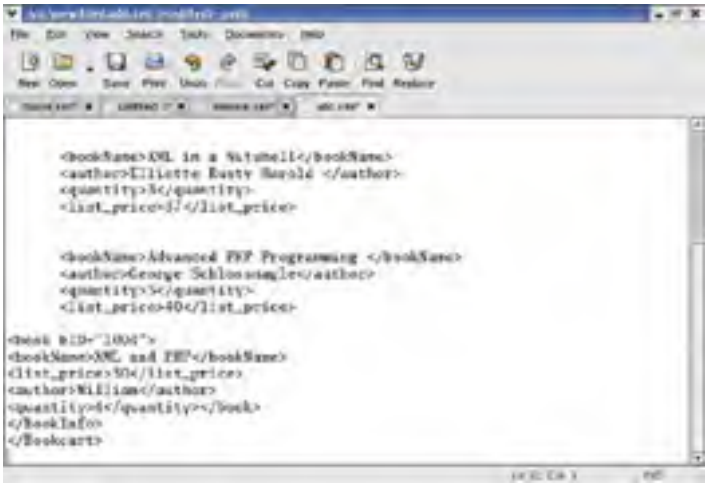


Figure 3-6: Output of Adding Information about Books Available in the DOM Tree

You can also delete elements from the tree structure using the DOM parser. For example, you can delete the information about a book sold from the online shopping cart.

Listing 3-14 shows how to delete elements from the DOM tree.

Listing 3-14: Removing Tree Elements

```
<?php
$xml_file = "bookcart.xml";
$tt=RemoveProduct($xml_file, "1003");
print($tt);
function RemoveProduct($xml, $id)
{
   $doc = xmldocfile($xml);
   $root = $doc->root();
   $children = $root->child_nodes();
   foreach ($children as $child)
   {
      if ($child->node_type() == XML_ELEMENT_NODE)
         {
            if ($child->tagname() == "BookInfo")
            {
               $BookInfo = $child->child_nodes();
               foreach ($BookInfo as $book)
               {
                  if ($book->node_type() == XML_ELEMENT_NODE)
                  {
                     $book_children = $book->child_nodes();
                     $ts = $book->get_attribute("ID");
                     if ($ts == $id)
                     {
                        $book->unlink_node();
                     }
                  }
               }
            }
         }
    }
```

```
    $xml = $doc->dumpmem();
    $fp = fopen("/var/www/html/remove.xml", "w+");
    fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
    return $xml;
}
?>
```

The above listing lets you remove the instances of a particular book from the online shopping cart. In the above listing, all the instances of the book with the book ID 1003 are removed from the cart using the unlink_node() function. Figure 3-7 shows the output of Listing 3-14:

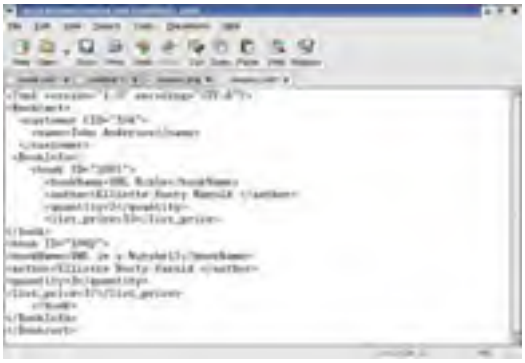


Figure 3-7: Output of Removing Tree Elements

Using DOM to Query a Document

In addition to adding and deleting elements from the XML document tree, the DOM parser also lets you retrieve specific information from an XML document. Querying the XML documents includes the process of traversing through the XML document to search for particular elements or information. For example, you have created an XML document that stores books information, such as author names, number of pages, prices, and titles of the books available at an online bookstore, and you want to retrieve information on a specific book from the XML document. You can create a query to retrieve information from the XML document using the DOM parser.

Listing 3-15 shows how to create an XML document that stores information on books:

Listing 3-15: Creating the book.xml File

```
<?xml version="1.0"?>
<books>
<book>
<title>XML in a Nutshell</title>
<author>Elliotte Rusty Harold</author>
<price>121</price>
<pages>300</pages>
</book>
<book>
<title>Advanced PHP Programming</title>
<author>George</author>
<price>161</price>
<pages>150</pages>
</book>
<book>
<title>Learning XML</title>
<author>John William</author>
<price>231</price>
<pages>200</pages>
</book>
</books>
```

The above listing creates the XML file, book.xml, which stores information about books available for online shopping. The above listing lets you retrieve information, such as author names and book prices from the book.xml file, for a book titled Learning XML, with more than 100 pages.

Listing 3-16 shows how to create the query to retrieve the specified book information:

Listing 3-16: Querying the XML Document using DOM

```
<?php
function get($domnode)
{
   $content = '';
   foreach ($domnode->child_nodes() as $child)
   {
      if ($child->node_type() == XML_TEXT_NODE)
      {
      $content .= $child->content;
```

```
        }
    }
    return $content;
    }
    $doc = xmldocfile("book.xml");
    $root = $doc->document_element();
    foreach ($root->child_nodes() as $element)
    {
       if ($element->node_type() == XML_ELEMENT_NODE and
       $element->tagname() == "book")
       {
          $c1 = false;
          $c2 = false;
          foreach ($element->child_nodes() as $book_element)
          {
             if ($book_element->node_type() == XML_ELEMENT_NODE and
             $book_element->tagname() == "title")
             {
             if (($title = get($book_element)) == "Learning XML")
             {
                $c1 = true;
             }
          }
          if ($book_element->node_type() == XML_ELEMENT_NODE and
          $book_element->tagname() == "pages")
          {
             if (($pages = get($book_element))>100)
             {
                $c2 = true;
             }
          }
          if ($book_element->node_type() == XML_ELEMENT_NODE and
          $book_element->tagname() == "author")
          {
             $author = get($book_element);
          }
          if ($book_element->node_type() == XML_ELEMENT_NODE and
          $book_element->tagname() == "price")
          {
             $price = get($book_element);
          }
       }
       if ($c1&& $c2)
       {
          print ("Author: $author Price:$price<br/>");
       }
    }
}
?>
```

The above listing shows how to query for specific information stored in an XML document using the DOM approach.

The DOM parser traverses through the entire DOM tree to search for the nodes that fulfill the specific conditions. The listing searches for the nodes that are of the element type and then verifies the tag name and the text in the nodes. The listing displays the author name and the price of the book with the title, Learning XML with more than 100 pages.

Figure 3-8 shows the output of Listing 3-16:

Figure 3-8: Output of Querying the XML Document

Using DOM to Write XML Data

In addition to modifying or traversing the XML document, DOM lets you create a new XML document. You can create a root node and add children to the root node to construct the DOM tree. The process of creating an XML document using the DOM parser can be used in the following instances:

- Creating an XML file from a text file.
- Constructing an XML document from various databases such as MySQL and MSXS.
- Combining more than one XML document into an XML document.
- Representing the PHP object properties in the form of an XML document.

The DOM approach provides you with various methods, such as add_root() and new_child(), which lets you create the root and text nodes of a DOM tree. For example, the add_root () method lets you add a root node to the DOM tree, and the set_attribute () method lets you set the attributes for the root node. The new_xmldoc() method lets you create a new XML document using the DOM parser.

Listing 3-17 shows how to create an XML document using the DOM parser:

Listing 3-17: Creating XML File using DOM

```
<?php
$doc = new_xmldoc("1.0");
// Adding root node
$root = $doc->add_root("Book");
// Setting attribute for the root node
$root->set_attribute("Title","XML and PHP");
// Adding children to the root node
$title = $root->new_child("Author","John Williams");
$author = $root->new_child("Price","145");
$date = $root->new_child("PublishingDate", date("d-M-Y", mktime()));
echo $doc->dumpmem();
print "$new_xmldoc";
$fp = fopen("/var/www/html/add.xml", "w+");
fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
return $xml;
?>
```

The above listing creates a DomDocument object, new_xmldoc, generating the first version of an XML document. The root node, Book, and the child nodes, such as Author, Price, and PublishingDate, are added to the DomDocument object. The dumpmem () method dumps the created XML tree into a string.

Figure 3-9 shows the output of Listing 3-17:

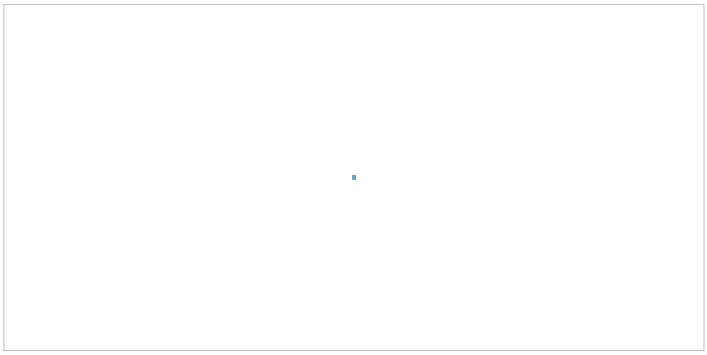


Figure 3-9: Contents of the File Created using DOM

Chapter 4: Understanding Extensible Stylesheet Language for Transformation

 Download CD Content

Extensible Stylesheet Language for Transformation (XSLT) is a language using which you can transform an eXtended Markup Language (XML) document into any other format, such as HTML. XSLT stylesheets change the structure and type of an XML document

This chapter introduces XSLT, and explains its data model, expression, and templates. It also explains the variables and parameters used in XSLT.

Introducing XSLT

XSLT is an application of XML that specifies how data is sent over the Internet using XML. You can use the XML Path (XPath) language to retrieve elements and attributes of an XML document to perform transformation on XML data. During the transformation process, you can sort, add, and delete elements from an XML document using XSLT. You can also change the order of the elements in the output of an XML document using the expressions and templates in XSLT. The XML document that is transformed is called a source tree or the source document and the transformed document is called the result tree or result document.

The XSLT processor parses the source tree, by creating an XSLT tree from the stylesheet that contains the specification of the source tree. The XSLT processor uses the source tree to generate the result tree. Various elements in XSLT, such as stylesheet, value-of, for-each, sort, and text help to create data model of XSLT, which specifies the structure of the source tree.

Data Model

The data model of XSLT specifies the structure of the source tree, which consists of elements and attributes of an XML document. You can also refer to elements in an XML document as nodes. XSLT validates an XML document by creating a tree structure of elements and attributes present in the document. A tree of an XML document consists of various nodes, such as root, element, text, attribute, namespace, processing instruction, and comment.

The root node refers to the root element of the tree structure. An element of an XML document refers to the child node of the root element. The root node contains one or more child nodes but does not contain any text. The root node also contains processing instructions and comment nodes as its child nodes.

The data model of an XSLT consists of various elements, such as:

- `<xsl:stylesheet>`
- `<xsl:value-of>`
- `<xsl:for-each>`
- `<xsl:sort>`
- `<xs:text>`

Each XSLT stylesheet starts with the stylesheet element. The syntax to declare the stylesheet element is:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

In the above syntax, xmlns is the attribute of the stylesheet element that provides a Uniform Resource Identifier (URI) of XSLT. You need to specify the URI, http://www.w3.org/1999/XSL/Transform, as a value of the xmlns attribute, to check that the elements and attributes are specified according to the specifications defined by W3C.

The value-of element displays the value of an element or attribute. The syntax to use the value-of element is:

```
<xsl:value-of select="element_name/@attribute_name"/>
```

In the above code:

- The select attribute of the value-of element indicates the name of an element or attribute that displays its value on a Web browser.
- The attribute name of an element must be prefixed with the @ sign.
- The value-of element is an empty tag that includes the slash mark, /, before the right angle bracket of the tag.

Enter the following code in the stylesheet to use the value-of element:

```
<xsl:value-of select="STUDENT"/>
```

The above code displays the value of the STUDENT element of an XML document. Enter the following code to display the value of an attribute:

```
<xsl:value-of select="@id"/>
```

The above code displays the value of the id attribute.

The for-each element of XSLT processes data each time a specified pattern occurs in the XML document. This element lets you use looping construct in XSLT as in the C or C++ language. The syntax to use the for-each element is:

```
<xsl:for-each select="pattern">
<!-- Data to be processed -->
</xsl:for-each>
```

In the above syntax, the select element can contain three patterns:

- The element pattern: The XSLT processor processes the data for each instance of a given element.
- The root/sub-element pattern: The XSLT processor processes the data each time a specified sub-element is encountered under the root element.
- The ancestor//sub-element pattern: The XSLT processor processes the data each time a specified sub-element is encountered under the ancestor element.

Enter the code in Listing 4-1 to use the for-each element with the root/sub_element pattern:

Listing 4-1: Using the for-each Element

```
<xsl:for-each select="STUDENTDATA/STUDENT">
<font color="red" size=20><b><xsl:value-of select="NAME" /></b></font>
<xsl:value-of select="@id"/>
</xsl:for-each>
```

In the above code:

- The root/sub_element pattern is used, which processes the data when the element encounters the STUDENT sub element of the STUDENTDATA root element.
- The <font> element of the HTML document specifies the font size and color for the value of the NAME element.
- The <b> element of the HTML document displays the value of the NAME element in bold. The value of the id attribute is displayed in normal text.

The sort element of XSLT arranges data in the ascending or descending order. The sort element has four attributes, select, order, case-order, and data-type. The syntax to use the sort-element is:

```
<xsl:sort select="element_name/expression"
order="ascending/descending"
case-order="upper-first/lower-first"
data-type="text/number"/>
```

In the above syntax:

- The select attribute: Indicates the expression on which sorting is performed.
- `The order attribute`: Indicates the order of sorting.
- `The case-order attribute`: Indicates the order of sorting for the uppercase and lowercase characters.
- `The data-type attribute`: Specifies the data type of the expression that is to be sorted.

Listing 4-2 shows the code to use the sort element:

Listing 4-2: Using the sort Element

```
<xsl:for-each select="STUDENT">
<xsl:sort select="AGE" data-type="number" order="ascending"/>
<xsl:value-of select="NAME"/>
<xsl:value-of select="AGE"/>
</xsl:for-each>
```

The above code shows that the for-each element traverses every sub element of the STUDENT element. The code displays the value of the NAME and AGE elements in the ascending order of the age element.

The default value for the order attribute is ascending, and the default value for the data-type is text.

Templates

XSLT contains a collection of template rules that provide specifications to convert an XML document and display the results on a Web browser. A template rule has two parts, which contain information about the transformation process. The first section specifies the part of an XML document to be converted, and the second specifies the format of the output.

The XSLT processor accepts the XML document and templates, processes the data, and generates an output.

The template and apply-templates elements define the template rules in XSLT. The syntax to use the template element is shown in Listing 4-3:

Listing 4-3: Using the syntax Element

```
<xsl:template match="expression">
<!--Data to be processed-->
</xsl:template>
```

In the above syntax, the value of the match attribute indicates an element of an XML document that checks against the template rules.

Various expressions that you can use in the match attribute are:

- `/`: Indicates the root element that matches all the elements, such as processing instruction, comments, and the root element of the XML document. It is also known as document root.
- `element`: Indicates an element of an XML document, which is processed when the XSLT processor encounters it.
- `*`: Indicates a wildcard character that matches any element in an XML document.
- `element1| element2`: Indicates two elements, which are processed when the element1 and element2 are encountered in an XML document.
- element [@attribute]: Processes data when an attribute of an element is encountered.
- `element [@attribute='value']`: Matches all the elements that contain the specified value of the attribute defined in the @attribute attribute.
- `root_element/sub_element`: Processes data when the sub elements of the root element are encountered.
- `ancestor//sub_element`: Processes data when the sub element under the ancestor element is encountered.

The apply-template element enables the XSLT processor to apply the templates, specified by the template element, to the selected XML elements. The syntax to use the apply-template is:

```
<xsl:apply-template [select="expression"]>
```

The above code shows that the template rules apply on the expression specified in the select attribute of the apply-template element. In the syntax, the select attribute is optional.

The default value of the apply-template element is node(), which specifies that the template matches the sub elements of the current node. The expression specified in the select attribute of the apply-template element matches the expression specified in the match attribute of the template element. Each node contains a base URI that resolves the value of the attributes. The URI of the document entity refers to the base URI of the document root node.

Listing 4-4 shows how to use the template and apply-template elements:

Listing 4-4: Implementing Template Rule

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:template match="/">
<xsl:apply-templates/>
</xsl:template>
```

In the above listing:

- The <xsl:template match="/"> element indicates the template rule.
- The value of the match attribute specifies the document root of an XML document.
- The apply-template element indicates that the template affects all the elements of an XML document.
- The listing does not contain a format style, so the output displays the result with the default font size, face, and color.

You can define your own template rules using the template and apply-templates elements, as shown in Listing 4-5:

Listing 4-5: Creating Template Rules

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:template match="/">
<xsl:apply-templates select="STUDENTDATA/STUDENT"/>
</xsl:template>
<xsl:templalte match="STUDENTDATA/STUDENT">
<font color="red">
<LI><xsl:apply-templates/></LI>
</font>
</xsl:template>
</xsl:stylesheet>
```

The above listing shows that the template, STUDENTDATA/STUDENT, is created for the document root. The XSLT processor searches for the STUDENTDATA/STUDENT template. It then sets the font color of the value of all the child elements of the STUDENT element to red, and displays them in bulleted list.

Expression

In addition to the templates, XSLT also contains expressions, which correspond to the nodes of an XML document being transformed. The main functions of the XSLT expressions are:

- Identify the node to be transformed.
- Specify the condition to process the node.
- Insert character data in the result tree.

The XPath expression defines the format of the expression for XSLT. It retrieves data based on specified conditions, and performs calculations on numeric data. The XPath expression can contain various characters, which are:

- `\n`: Indicates a newline character.
- `.`: Indicates a single character.
- `\d`: Indicates a numeric character.
- `\s`: Indicates the whitespace characters, such as horizontal tab, line feed, and carriage return.
- `[a-d]`: Indicates the characters from a to d.
- `*`: Indicates zero or more occurrences of the whitespace characters.
- `+`: Indicates one or more occurrences of the whitespace characters.
- `?`: Indicates zero or one occurrence of the whitespace characters.
- `\?`: Searches for the question mark character, and overwrites the special meaning of the question mark expression.

You can combine the characters in the XPath expression to denote specific meaning. For example, the expression character, \s*, used in the XPath expression, specifies zero or more occurrences of the whitespace characters. The expression character, \s+, specifies at least one or more occurrences of the whitespace characters.

An expression in XSLT uses the following functions:

- `tokenize(string, delimiter)`: Returns the number of strings that are separated by the delimiter.
- `matches(string1, string2)`: Returns the value, true, which specifies that the value of the first parameter is same as the value of the second parameter.
- `replace(string1, string2, string3)`: Searches the second string within the first string and replaces the searched string with the third string specified in the function.

You can specify expressions within a function to change the format of an XML document.

Listing 4-6 shows the use of the expression characters and functions:

Listing 4-6: Using the XSLT Expressions and Functions

```
<xsl:template match="price">
<xsl:match>
<xsl:attribute name="pattern">
<xsl:value-of select='matches(.,".*\$\d\.\d.*")' />
</xsl:attribute>
<xsl:value-of select='replace(., "\$\d\.\d", "\$#.#")' />
</xsl:match>
</xsl:template>
```

The above listing shows that the template changes the price element by adding the pattern attribute. The pattern attribute stores the value, true or false, depending on the result of the match() function. The content of the price element changes according to the result of the replace() function. In the above code:

- The first parameter of the matches() function indicates the entire text of the document.
- The second parameter of the matches() function indicates the expression that specifies any text before and after the dollar sign, followed by a digit, period, and another digit.
- The second parameter of the replace() function indicates the expression that specifies the dollar sign followed by a digit, period, and another digit.
- The XSLT processor replaces the text of the pattern element with the third parameter, $#.#, of the replace() function.

Listing 4-7 implements the expressions and functions of XSLT on the following XML document:

Listing 4-7: Implementing the Expressions and Functions on XML Document

```
<product>
<price>Price of Pen: $2.8</price>
<price>Price of Pencil: $1</price>
</product>
```

In the above code:

- The first instance of the price element matches the expression. As a result, the pattern attribute of the price element stores the value, true.
- The second instance of the price element does not match the expression, because it does not contain a period followed by a digit.

The template rule specified in Listing 4-3 changes the XML document, as shown in the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
<price pattern="true"> Price of Pen: $#.#</price>
<price pattern="false">Price of Pencil: $1</price>
```

</product>

The above code shows that the pattern attribute of the second instance of the price element contains the false value because it does not match the expression. As a result, the text, $1, of the price element is not replaced with the text, $#.#.

◄ PREVIOUS NEXT ►

Using Variables and Parameters in XSLT

A variable is a memory location that stores data, and parameter is a variable that is passed to a function. The <xsl:variable> element helps define variables in XSLT, and the <xsl:param> element helps define XSLT parameters. The <xsl:variable> and <xsl:param> elements contain the name attribute that specifies the name of the variable and parameter.

You create local variables and parameters by defining them in the template rule. You can also create global variables and parameters in XSLT, by creating them as child elements of the <xsl:stylesheet> element. The global variable is accessible throughout the stylesheet.

You cannot change the value of an XSLT variable. You need to create a new variable if you want to change the value of an existing variable. Enter the following code to create an XSLT variable:

```
<xsl:variable name="StudName" select="/StudentDetail/Student>
</xsl:variable>
```

In the above code, the select attribute specifies the XPATH expression and the result of the expression specifies the value of the XSLT variable. The StudName variable indicates an element node, which refers to the Student element of the source document.

Result Tree Fragments

The <xsl:variable> or <xsl:param> element creates a result tree fragment when you refer to the value of a specific element in the result tree. The result tree fragment is a part of the result tree obtained from an XSLT transformation, and is created by the XSLT processor. A result tree fragment should be balanced, which means that each start tag must correspond to an end tag. You can perform a valid string operation on a result tree fragment. For example, the following code shows a part of an XSLT stylesheet:

```
<xsl:variable item="Computer">
<ch>chapter number</ch> books
</xsl:variable>
```

The item variable, in the above code, creates the following result tree fragment:

```
<ch>chapter number</ch> books
```

The result tree fragment, in the above code, consists of the <ch> element, and the trailing string, books, outside the <ch> element.

> **Note** You cannot use the /, //, and [] operators on result tree fragments.

The <xsl:copy-of> element inserts a result tree fragment into the result tree without converting it to a string value. The syntax to use the <xsl:copy-of> element is:

```
<xsl:copy-of select=expression/>
```

The above syntax shows that the select attribute contains an expression that is evaluated. When the evaluation of the specified expression returns a result tree fragment, the <xsl:copy-of> element copies the tree fragment to the result tree.

An XSLT stylesheet consists of one or more template rules, which identify a pattern that you need to match in a source tree, and describe the structure of the result tree. Each template rule is applicable to a specific set of nodes, and executes a set of instructions to create the structure of the nodes in the result tree.

Top-level Variables and Parameters

A top-level variable is a sub-element of the <xsl:stylesheet> element. A top-level variable creates a global variable and parameter. You cannot change the value of a top-level variable. The <xsl:variable> and <xsl:param> elements are the examples of top-level elements. The <xsl:param> element only creates a parameter of the stylesheet, but does not pass the parameter to the stylesheet. You can assign a value to the parameter of the stylesheet using various forms, such as the top-level parameter, the <xsl:with-param> element, or a default value, as shown in Listing 4-8:

Listing 4-8: Assigning the Value to a Parameter

```
<xsl:template match="/">
<xsl:call-template name="Temp'>
<xsl:with-param name="paramName" select="'any-value'"/>
</xsl:call-template>
</xsl:template>
<xsl:template name="Temp">
<xsl:param name="paramName" select='default-value'"/>
<xsl:value-of select="$param"/>
</xsl:template>
```

The above listing shows that the <xsl:call-template> element invokes the Temp template, which accepts the paramName parameter. The Temp template assigns a default value to the parameter.

Variables and Parameters in a Template

XSLT contains the <xsl:variable> element that creates variables in stylesheets. The syntax to declare an XSLT variable is:

```
<xsl:variable name="var" select="0"/>
```

In the above syntax, you declare a variable using an XPath expression. The var variable, declared in the above syntax, stores the numeric value, 0.

Another syntax for declaring an XSLT variable is:

```
<xsl:variable name="var">0</xsl:variable>
```

In the above syntax, the value of the var variable is specified within the starting and ending tags.

You can use a variable either as a top-level element or as a template-level element. A template-level element is defined within the template, and all the elements positioned after the variable declaration can use this element. You can change the value of a variable by declaring it inside the for-each loop, where its value changes on every iteration. The template-level elements remain in effect only within their scope, which is represented by the following-sibling nodes and the descendent nodes of the elements.

The parameter element in XSLT is similar to the variable element. Both these elements share the same namespace and syntax to assign names and values to an element. The keyword, param, denotes the parameter element. The param element consists of a name attribute and a select attribute. The value of the select attribute in a parameter element specifies the default value of the element. This means that if you pass a new value to the select attribute, then the default value is replaced with the new value. You can assign value to a parameter element either from the process that invokes the stylesheet, or from the <xsl:with-param> element.

To pass the value of a parameter to another template, you pass a parameter using the <xsl:with-param> element. The process of passing a parameter to a template calls a specific template using the <xsl:call-template> element.

◄ PREVIOUS NEXT ►

Chapter 5: PHP and XPath

Download CD Content

Extensible Markup Language (XML) Path (XPath) language is a query language that you can use to retrieve selected information from an XML document. It also provides functions to manipulate strings, numbers, booleans, and nodeset values in an XML document. Using XPath, you can create XPath expressions that specify the path of an XML element to be retrieved from an XML document.

Hypertext Preprocessor (PHP) consists of XPath classes that you can use to retrieve XML elements matching specific criteria. You can use XPath expressions with the Document Object Model (DOM) and Simple Application Programming Interface (API) for XML (SAX) parsers implemented in PHP language to determine the location of the XML elements in an XML document. In addition, you can process XML documents using XPath expressions in Extensible Stylesheet Language for Transformation (XSLT) templates.

This chapter describes XPath and how to use XPath expressions in an XML document. It also describes how to use location paths to retrieve specific elements from an XML document. In addition, it explains the method to access XML elements using DOM and SAX parsers and XSLT templates implemented in PHP.

Introducing XPath

An XPath expression specifies the part of an XML document that you need to select and retrieve.

XPath selects a portion of data from an XML document by navigating the hierarchical structure of that document. This query language follows a path to point to a specific node in the XML document hierarchy. XPath searches an XML document from the starting point of the document called the context node, progresses in a specific direction, and arrives at the required point by navigating through the path described by a location path. Location steps in the location path determine the path to navigate an XML document and locate a particular XML element.

Location steps consist of three parts: an axis, node test, and predicates. The axis informs the XPath processor about the process of navigating around an XML document. The node test determines the type of node selected by the location step, along with its name. The predicate filters the set of nodes selected by the axis and the node test. If there are more than one location step in a location path, each location step uses the node in the previous nodeset as its starting point.

Creating Nodes Using XPath

An XML document consists of a succession of elements, each with a start and end tag. Various elements are nested within each other in the document. XPath uses a data model to represent all elements within an XML document as a hierarchy of nodes.

For example, you want to represent the data model in XPath for the customer.xml XML document.

Listing 5-1 shows the contents of customer.xml:

Listing 5-1: The customer.xml Document

```
<?xml version="1.0"?>
<customer>
<companyname>IBG, Inc.</companyname>
<item>Computer</item>
<itemcode>c001</itemcode>
<item_quantity>20</item_quantity>
<price>20000</price>
</customer>
```

The above listing shows the content of an XML document, customer.xml. The document contains information about an item that a company wants to purchase. This information includes the company name, item name, item code, item quantity, and the price.

Figure 5-1 shows the corresponding data model in XPath for the customer.xml document:

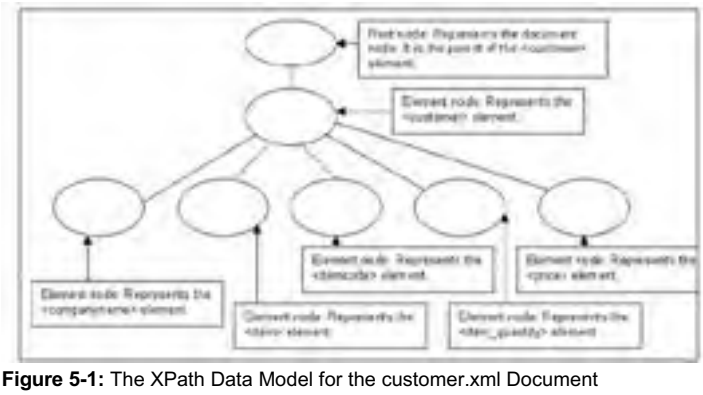


Figure 5-1: The XPath Data Model for the customer.xml Document

This model shows a root node, which is the parent of the element root. The element root is represented by the element node, which represents the <customer> element. The element node that represents the <customer> element contains five child element nodes that represent the <companyname>, <item>, <itemcode>, <item_quantity>, and <price> elements.

In XPath, a node refers to a specific part, such as an element or attribute, of an XML document. Each XML document functions as a tree and consists of a root node, which is the parent of the element node. XPath includes seven types of nodes. Of these seven nodes, only some nodes consist of an expanded name. The expanded name consists of two parts, local part and namespace Uniform Resource Identifier (URI). Local part is a string and namespace URI is either a string or a NULL value.

The namespace URI is a URI reference. An XML document that does not contain a namespace declaration contains a NULL value for the namespace URI. In these cases, the expanded name of an element is same as the element name.

The seven types of nodes available in XPath are:

- Root: Represents the root of the XPath tree. There is only one root node for each XPath tree structure. It does not have an expanded name.
- Element: Represents each element in the XML source document. It contains an expanded name that results from expanding the Qualified Name (QName) in the element tag. The QName is an element name with a prefix added to it. If the QName does not contain a prefix and default namespace, the value of the namespace URI for the expanded name of the element is NULL.
- Attribute: Represents the attributes of an element in the XML source document. It can be empty. The element node is the parent of the attribute node. However, an attribute node is not a child of the parent node.
- Namespace: Associated with an element node. A namespace node exists for each attribute on the element whose name starts with the prefix, xmlns:. A namespace node exists for each attribute on an ancestor element with names starting from xmlns:.
- Processing instruction: Represents the processing instructions in the XML source document. It consists of an expanded name with the local part representing the application to which the processing instruction applies.
- Comment: Represents the comments in the XML source document. It does not have an expanded name.
- Text: Represents the character data in the XML document. Text nodes are not produced for character values that exist inside comments, attributes, and processing instructions. It does not have an expanded name.

Note If a location path starts with child, the resultant nodeset contains neither attribute nor namespace nodes.

Every node other than the root node in the data model consists of one parent. For example, the XML document, customer.xml, consists of all nodes included in it.

Listing 5-2 shows the customer.xml document that contains different types of nodes:

Listing 5-2: The customer.xml Document with Different Nodes

```
<?xml version="1.0"?>
<customer><!—Denotes the element node-->
<companyname company="IBG, Inc."><!—company is the child attribute node.-->
The name of the company is IBG, Inc.!<!—The string denotes the child text node.-->
<customer xmlns=http://www.xmml.com/><!—Denotes the namespace
node-->
</companyname>
<item>Computer</item>
<?customer SELECT * FROM item?><!—Denotes the processing instruction node.-->
</customer>
```

The above listing shows the content of the customer.xml document with a root node, which is the parent of the comment node, <companyname> element node, <item> element node, and the processing instruction node. The root node is not visible in the document. The <companyname> element node consists of a child attribute node, which is the company with the value, "IBG, Inc.". The <companyname> element node also contains a child text node with the value, "The name of the company is IBG, Inc.". The comment node starts with <!—and ends with -->.

Note Two expanded names are equal if they have the same local part, regardless of the value of the namespace URI.

Implementing the XPath Syntax

XPath does not use the XML-based syntax. You cannot include XPath expressions in the URIs or attributes of XML elements. The XPath syntax is the same as file system addressing and enables querying the data in an XML document. XPath uses a pattern expression to:

- Identify the nodes within an XML document.
- Select unknown elements.
- Select the branches of an element.
- Select several paths along the axis.
- Select attributes.

An XPath pattern expression describes a path that contains a list of child element names separated by a slash. The pattern selects those elements from an XML document that matches the path.

Listing 5-3 shows a sample XML document:

Listing 5-3: The sample XML Document

```
<?xml version="1.0"?>
<customer>
<item itemcode="01">
<itemprice price="15.00"></itemprice>
<quantity value="20"></quantity>
</item>
<item itemcode="02">
<itemprice price="30.00"></itemprice>
<quantity value="10"></quantity>
</item>
</customer>
```

The above XML document contains information about items purchased by a customer. The document contains item information, such as the item code, item price, and the quantity.

The following pattern expression selects all <itemprice> element nodes of all <item> element nodes of the <customer> element node:

```
/customer/item/itemprice
```

In the above pattern expression, <itemprice> element is the child element node of the <item> element node, which is the child element of the <customer> element.

> **Note** If a path expression starts with a double slash (//), the resultant nodeset contains all elements described by the path expression.

For example, the path expression //customer selects all customer elements in the customer.xml document.

XPath uses wild card characters, such as *, to select unknown elements in an XML document. The * wild card character selects all elements defined by the preceding path. For example, the expression to select all child element nodes of all <item> element nodes of the <customer> element node is:

```
 /customer/item/*
```

Similarly, the expression to select all <itemprice> elements that are grandchild elements of the <customer> element is:

```
/customer/*/itemprice
```

The pattern expression to select all elements that have two ancestors is:

```
/*/*/itemprice
```

The pattern expression to select all elements in an XML document is:

```
//*
```

You can specify branches on a location step using additional patterns. As a result, these additional patterns further describe an element. For example, the expression that selects the first node representing the <item> element child of the <customer> element node is:

```
/customer/item[1]
```

In the above syntax, the numeric value in the square brackets represents the position of the element in the selected element set. The expression to select the last <item> element belonging to the <customer> element is:

/customer/item[last()]

The expression that selects all <itemprice> element nodes of the <customer> element node with the <price> attribute value 15.00 is:

```
/customer/itemprice[price=15.00]
```

You can select several paths in a document using the pipe (|) operator in XPath. For example, the expression to select all <quantity> and <itemprice> elements of the <item> element of the <customer> element is:

```
/customer/item/quantity|/customer/item/itemprice
```

Similarly, the expression to select all <quantity> and <itemprice> elements in the XML document is:

```
//quantity|//itemprice
```

Using XPath, you can also select attributes in an XML document. The at (@) symbol prefixes all attributes in an XPath expression. The @ symbol signifies that selected elements represent the attributes of an element. For example, the expression that selects all attributes named as price is:

```
//@price
```

The expression that selects all <item> elements with an attribute named itemcode is:

//item[@itemcode]

XPath syntax can be either abbreviated or unabbreviated. The unabbreviated syntax for a location step consists of an axis name and node test separated by a double colon. The syntax may or may not include predicates in square brackets. The child axis includes the child nodes of the context node. This axis is the default axis, and you can eliminate it from the location step. For example, you can write the abbreviated form of an XPath expression as:

```
/customer
```

The above syntax is the same as the following unabbreviated XPath expression:

```
 /child::customer
```

The descendant axis consists of nodes that are descendants of the context node. The descendants can be a child node or a grandchild node. For example, an XPath expression that selects descendant nodes is:

```
/descendant::*
```

In the above syntax, XPath selects all descendant nodes of the root element node.

Note The descendent axis cannot include the attribute and namespace nodes.

PREVIOUS NEXT

XPath Expressions

XPath expressions select a part of data from an XML document using a location path, which notifies XPath about the path that needs to be followed in an XML document. An XPath processor returns one of the following objects after evaluating an XPath expression:

- Nodeset: Represents an unordered collection of nodes.
- Boolean: Represents either TRUE or FALSE.
- Number: Represents a floating-point number.
- String: Represents a sequence of Unicode characters.

Structure of an XPath Expression

A DOM or SAX parser parses an XPath expression by separating the character string into tokens, such as identifiers and numbers. The DOM or SAX parser then parses the resulting tokens. This process is called the tokenization. You can form tokens from a sequence of characters. XPath expressions are formed from these tokens. You can identify various tokens in an XML document using lexical structures, which defines the syntactic structure of an XPath expression. For example, the lexical structure for the node type token in an XPath expression is:

```
Node Type::='comment'|'text'|'processinginstruction'
```

The above structure shows that the type of node specified in an XPath expression can be a comment node, text node, or a processing instruction node.

XPath Functions for Nodeset

A nodeset is a collection of nodes. A nodeset is formed when the | operator combines two or more location steps. The following XPath functions operate on nodesets:

- `count()`: Accepts a nodeset as its argument and returns the total number of nodes present in the nodeset.
- `id()`: Returns a nodeset that contains nodes with the given ID attribute.
- `local-name()`: Returns the local part of the name of a node.
- `last()`: Returns the value of the context size. In the XPath data model, you can determine the context size by counting the number of nodes that represent the same element in an XML document.
- `name()`: Returns a QName representing the name of a node.
- `position()`: Returns the value of the context position, which is the position of the context node in the nodeset selected prior to the predicate that the XPath processor evaluates.
- `namespace-uri()`: Returns a string value that represents the namespace URI in the expanded name of the node.

Types of XPath Expressions

In addition to location path expressions, XPath supports string, numerical, equality, relational, and boolean expressions. You can use these expressions if you do not want to retrieve a nodeset from an XPath data model. You can use these expressions in XPath predicates.

You can use numerical expressions to perform arithmetic operations on numbers in an XML document. XPath converts each operand to a number before performing arithmetic evaluation.

Table 5-1 describes various numerical operators that you can use with numerical expressions:

Table 5-1: XPath Numerical Operators

Numerical operator	Description	Example	Output
+	Addition	2+1	3
-	Subtraction	2-1	1
*	Multiplication	2*1	2
Div	Division	6 div 2	3
Mod	Modulus	6 mod 4	2

The following numerical XPath expression returns all descendant nodes whose price value is divisible by 2:

```
//[price mod 2 ==0]
```

You can use boolean expressions to compare two element nodes in an XML document. XPath converts each operand in the boolean expression to a boolean value before evaluating it. The boolean operators that you can use with an XPath expression are:

- Or: Performs the OR operation.

- And: Performs the AND operation.

For example, the following XPath expression selects all child <books> element nodes that have the title and price child elements:

```
/books[title and price]
```

The following XPath expression selects all child <books> elements that have either the <bookcode> child element node with a value greater than 20 or the <price> child element node with a value less than 500:

```
/books[(bookcode>20) or (price<500)]
```

A string is defined as a sequence of one or more characters. You can also use string expressions in XPath predicates. For example, the following XPath expression selects all <title> child elements that have the name attribute value, Operating System, with the <books> element node as its parent:

```
/books/title[@name='Operating System']
```

FunctionCall XPath Expression

An XPath processor evaluates a FunctionCall XPath expression using a function necessary for evaluating an expression. FunctionName identifies the name of the function that evaluates the expression. All XPath functions are available in the XPath function library. The XPath processor calls a function to evaluate an XPath expression. The processor then evaluates the function arguments by converting each argument to the data type that the function accepts. The XPath processor passes these converted arguments to the called function. The value that the called function returns represents the result of the FunctionCall expression. The called function displays an error if the number of arguments passed to the called function is incorrect or if arguments are not converted to the type required by the called function.

The syntax for the XPath FunctionCall expression is:

```
FunctionCall=FunctionName (arg1, arg2, ..., argn)
```

In the above syntax, the function, FunctionName, operates on n number of arguments.

If the XPath processor calls a string function in an expression, arguments are converted to the string type. Similarly, if the processor calls a number function, arguments are converted to the number type.

Note You cannot convert an argument, which is not a nodeset type, to a nodeset.

XPath includes various functions to perform string operations, such as changing string characters from uppercase to lowercase. Various string functions are:

- `string()`: Converts its arguments to a string value. For example, it converts boolean arguments to string values, TRUE or FALSE.
- `string-length()`: Returns the total number of characters in a string.
- `starts-with()`: Determines whether or not a string starts with another string.
- `substring()`: Returns a portion of a string determined by the numeric arguments that describe the substring. For example, the function, substring('Tokyo', 1, 3) returns the substring, Tok. The numeric value, 1, in this function indicates the starting location and 3 indicates the number of characters in the substring.
- `concat()`: Combines two or more string arguments in the specified sequence. For example, the function, concat('The', 'world') returns the value, The world.
- `contains()`: Checks whether or not one string value contains another string value as its substring. For example, the function, contains('priority', 'prior') returns the value TRUE because the string, priority, contains prior as its substring.
- `normalize-space()`: Removes the leading and trailing whitespace from a string. It also converts the irrelevant whitespaces within a string by converting the sequence of whitespaces to a single space string. For example, the function, normalize-space('Global Systems') returns the string 'Global Systems'.
- `substring-after()`: Returns that portion of a string, which appears after the first occurrence of the specified substring. For example, the function substring-after('unauthorized', 'un') returns the string, authorized, because this string appears after the specified substring, un.
- `substring-before()`: Returns that portion of a string, which appears before the first occurrence of the specified substring. For example, the substring-before('predict', 'dict') function returns the string, pre.
- `translate()`: Accepts a string and two sequences of characters as its arguments. It then replaces the characters of the string by replacing characters in the first character sequence with equivalent characters in the second character sequence. For example, the function translate(SQL, LMNRSPQR, lmnrspqr) provides the output, sql.

XPath also consists of functions that operate on numbers in an XML document. A number represents a floating-point number. It can have any double precision 64-bit format Institute of Electrical and Electronics Engineers (IEEE) 754 value. The number functions available in XPath are:

- `number()`: Converts the arguments to a number. For example, this function converts the boolean argument TRUE to the value 1 and FALSE to 0.
- `round()`: Returns the integer value that is the closest to the numeric value of the argument.
- `floor()`: Returns the largest integer value that is either less than or equal to the argument of the function. This function rounds off a noninteger value to the immediate lower integer value.
- `sum()`: Adds a series of numbers present as a nodeset. This function converts each node in the nodeset to its

corresponding string value and then, converts these string values into corresponding numeric values. This function adds all numeric values and generates the result.

- `ceiling()`: Returns the smallest integer value that is greater than or equal to the function argument. It rounds off a noninteger value to the immediate higher integer value.

Note The round(), floor(), and ceiling() number functions accept only a single argument.

A boolean object type contain either of the two values, TRUE or FALSE. Various boolean functions in XPath are:

- `boolean()`: Converts its argument to a boolean value. For example, if the argument is a number greater than 0, the boolean() function evaluates it to TRUE. Otherwise, it evaluates the number to FALSE. Similarly, if the argument is a string and the string length is 0, the boolean() function returns the boolean value FALSE. Otherwise, it returns the boolean value TRUE.
- `false()`: Returns the boolean value FALSE. While working with boolean values, you can use the false() function in an XPath expression.
- `true()`: Returns the boolean value TRUE. While working with boolean values, you can use the true() function in an XPath expression.
- `lang()`: Returns a boolean value, which checks if the language of the context node is similar to the language supplied as an argument.
- `not()`: Returns the negation of its argument. For example, if the argument is TRUE, the not() function returns the value, FALSE.

Note The language of the context node is defined by inserting the xml:lang attribute in an XML document.

PREVIOUS NEXT

Location Paths

A location path is an XPath expression that notifies the XPath processor about how to navigate around an XML document. It is a sequence of location steps separated by a slash, /. The XPath processor evaluates a location path from left to right starting with an initial context node. Each node that results from the evaluation of one location step represents the context node to evaluate the next location step. XPath, then, combines the results of all location steps and returns selected XML elements.

There are two types of location paths, absolute and relative. You start the absolute location path with a /, but cannot start the relative location path with a /. In an absolute location path, the current nodeset consists of the root node. In a relative location path, the location steps are separated by a /. The / represents a direct parent-child relationship between the nodes involved in the location step.

Each location step selects a nodeset with respect to its context node. The nodes in a nodeset represent the context node for the next location step. For example, the location path, child::customer/child::item, selects the child item element of the child customer element of the context node. The child axis is the default value for an axis.

Creating Location Steps

A location step determines the nodes through which an XML document should be traversed to arrive at a final location. The syntax for a location step is:

```
axis_name::nodetest[predicate]
```

For example, you navigate through an XML source document, companydetails.xml, using XPath.

Listing 5-4 shows the content of the XML document, companydetails.xml:

Listing 5-4: The companydetails.xml Document

```
<company name="Blue Moon Systems">
<Managing Director>Ron Floyd</Managing Director>
<Department name="Administration">
<Name>Tom</Name>
<Age>35</Age>
<Address> 17, landmark plaza, New York</Address>
<EmployeeId>a01</EmployeeId>
</Department>
<Department name="Sales">
<Name>John</Name>
<Age>43</Age>
<Address> 34, landmark plaza, New York</Address>
<EmployeeId>s02</EmployeeId>
</Department>
</company>
```

The above listing shows the content of the companydetails.xml document that contains information about the Blue Moon Systems company. The companydetails.xml document consists of three element nodes: company name, Managing Director, and Department. The Department element node consists of an attribute node, name. For each Department attribute node, there are four child nodes: Name, Age, Address, and EmployeeId.

For example, the location path, child::company name/ child::Department [attribute::name="Sales"] consists of two location steps:

- child::company name
  - Axis: child
  - Node test: company name
  - Predicate: null
- child::Department [attribute::name="Sales"]
  - Axis: child
  - Node test: Department
  - Predicate: [attribute::name="Sales"]

The first location step does not contain a predicate but the second location step includes a predicate. The predicate in the second location step selects the node representing a Department element only if it contains a name attribute with the value, Sales.

Identifying Axes

An axis within an XPath location step determines the direction in which the XPath processor should navigate an XML document. There are two types of XPath axes, forward and backward. A forward axis consists of either the context node or nodes that appear after the context node in an XML document. A backward axis consists of the context node together with nodes that appear before it in an XML document. The backward axis is also known as the reverse axis.

The XPath processor uses the position() function to evaluate the position of a node using the information about the type of axis. If an axis is forward, the position of a node in the nodeset obtained from an XPath tree model is equal to the position of the node in the XML document order. For example, the first child element node in a tree model is the first child element node in an XML document order. If the axis is reverse, the position of a node in the nodeset obtained from an XPath tree model is equal to the position of the node in the reverse document order. For example, the first ancestor element node in a tree model is the last ancestor element in reverse document order. XPath consists of 13 axes. They are:

- `Child`: Contains child nodes of the context node. The attribute and namespace nodes are not the child nodes of the element node to which they belong. A child axis is the default axis and never returns any attribute and namespace nodes in the return nodeset.
- `Parent`: Contains the node that represents the parent of the context node. If the root node of a document is the context node, it does not have any parent.
- `Ancestor`: Contains the ancestors of a context node. The ancestors of a context node include parents of the context node, parents of the parent, and other parents in the hierarchy. The ancestor axis always includes the root node except when the root node itself is defined as the context node. The root node does not have any ancestors.
- `Descendant`: Contains the descendants of a context node. The descendants of a context node include the child nodes of the context node, child nodes of child nodes, and other child nodes in the hierarchy. As a result, the child axis is a part of the descendant axis. It does not contain the attribute and namespace nodes.
- `Descendant-or-self`: Contains descendant nodes together with the context node.
- `Ancestor-or-self`: Contains ancestor nodes together with the context node.
- `Following-sibling`: Contains all element nodes that are sibling of the context node and appear after the context node in an XPath data model. If the context node is either an attribute node or a namespace node, then the following-sibling axis does not include any node. For example, the customerdetails.xml file contains the context node as the node that represents the <Department> element with attribute value, Administration. The XPath expression that uses the following-sibling axis to retrieve the <Department> element node with attribute value, Sales, is:

  ```
  child::Department [@name=Administration]/following-sibling::Department [@name=Sales]
  ```
- `Preceding-sibling`: Contains all element nodes that are sibling of the context node and precedes the context node in the data model. If the context node is either an attribute node or a namespace node, the preceding-sibling axis does not include any node. For example, the customerdetails.xml file contains the context node defined as the node that represents the <Department> element with attribute value, Sales. The XPath expression that uses the preceding-sibling axis to return the <Department> element with attribute value, Administration is:

  ```
  child::Department [@name=Sales]/preceding-sibling::Department [@name=Administration]
  ```
- `Following`: Contains all nodes that appear after the context node in an XML document. It excludes the descendant nodes, attribute nodes, and namespace nodes.
- `Preceding`: Contains all nodes that appear before the context node in an XML document except the ancestor node, attribute node, and the namespace node.
- `Attribute`: Contains the attributes of a context node. It includes a node only if an element node is the context node. In all other cases, the attribute axis is empty.
- `Namespace`: Contains the namespace nodes of the context node. It includes a node only if an element node is the context node. If the context node is other than an element node, the attribute axis is empty.
- `Self`: Contains only the context node.

Figure 5-2 shows the data model for an XML document:

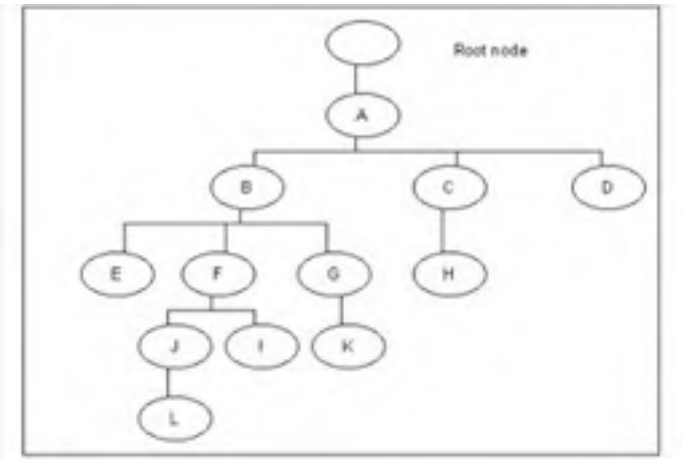


Figure 5-2: Data Model

You can determine various nodes corresponding to a specific axis by choosing a context node.

Table 5-2 lists the nodes corresponding to various axes, choosing node F as the context node:

Table 5-2: Nodes Corresponding to an Axis

Axis Name	Nodes
Self	F
Child	I, J
Parent	B
Ancestor	B, A, root node
Descendant	I, J, L
Descendant-or-self	F, I, J, L
Ancestor-or-self	F, B, A, root node
Following-sibling	G
Preceding-sibling	E
Following	G, K, C, D, H
Preceding	E

The child, parent, descendant, descendant-or-self, following-sibling, following, attribute, namespace, and self axes are forward axes. The ancestor, ancestor-or-self, preceding-sibling, and preceding axes are reverse axes.

Performing the Node Test

A node test describes a test on those XPath tree nodes that an XPath processor selects along the specified location step axis. The nodes that succeed this test act as a nodeset for the next location step. A node test identifies a node within an axis. Various node tests that an XPath processor can perform on XPath expressions are:

- `QName`: Selects nodes that have a QName and represent principal nodes for an axis.
- `node()`: Selects all nodes regardless of their name and type.
- `text()`: Selects all text nodes.
- `comment()`: Selects all comment nodes.
- processing instruction(): Selects all processing instruction nodes.
- `prefix:*`: Selects principal nodes that belong to the namespace defined by a prefix.

Note The * node test selects all principal nodes for an axis. For a child axis, the principal node type is an element node. If the context node includes child nodes other than element nodes, the resultant nodeset does not include those nodes.

Listing 5-5 shows the content of the XML document, course.xml:

Listing 5-5: The course.xml Document

```
<?xml version="1.0" ?>
<!--An example XML representation of the courses in a college-->
<courses>
<college>St. Peters</college>
<branches>
<branch name="EC">
<semester number="1">
<subject name="Digital Circuits">
First subject in semester 1 of EC branch.
</subject>
<subject name="Analog Circuits">
</subject>
<subject name="Electromagnetics">
</subject>
</semester>
<semester number="2">
<subject name="Radar Theory">
</subject>
<subject name="Satellite Communication theory">
</subject>
<subject name="Active Networks">
</subject>
</semester>
</branch>
<branch name="CS">
<semester number="1">
<subject name="Microprocessor">
</subject>
<subject name="Data Structure">
</subject>
<subject name="Computer System Organization">
```

```
</subject>
</semester>
<semester number="2">
<subject name="Operating System">
</subject>
<subject name="Java">
</subject>
</semester>
</branch>
</branches>
<CommonSubjects>
<Common name="PDC"> </Common>
<Common name="NA"> </Common>
<Common name="CP"> </Common>
</CommonSubjects>
</courses>
```

In the above listing, the course.xml document contains information about the semester-wise courses for two disciplines, EC and CS. The course.xml document also lists the subjects common for these two disciplines.

The following location step expressions demonstrate how to implement a node test:

- `child::subject`: Selects the <subject> child nodes of the context node. If the context node is the <semester> element with the number attribute value, 2 in the EC branch, the returned nodeset contains three <subject> child nodes. In addition, if the <courses> element is the context node, the location step expression returns an empty nodeset. This is because the <courses> element consists of the <subject> child node as its descendant node.
- `child::*`: Selects all element nodes that are child nodes of the context node. If the <courses> element is the context node, the expression returns a nodeset that contains the <college>, <branches>, and <CommonSubjects> element nodes.
- `child::text()`: Selects text nodes that are child nodes of the context node. For example, the context node is the <subject> element node with the name attribute value, Digital Circuits in semester 1 of EC branch. The XPath expression, returns the nodeset that consists of one text node having the string value, First subject in semester 1 of EC branch.
- `child::branch/descendant::subject`: Selects <subject> element nodes that are descendant nodes of the <branch> child node of the context node. If you select the <branches> element node as the context node, the expression returns a nodeset that contains 11 <subject> elements. This is because all <subject> element nodes are descendant nodes of the <branches> element node.

In the above examples, the XPath expressions use the relative location path. Alternatively, you can use an absolute location path to retrieve specific data from an XML document. For example, the location step that selects all element nodes that are child nodes of the root node expression is:

```
/child::*
```

There is only one element node for a root node. As a result, the above location step returns a nodeset that consists of only the <course> element node.

Setting Predicates

The location step consists of a predicate that describes the conditions that a node must satisfy to be selected. A predicate acts as a filter for the nodeset that the node test returns. The resultant nodeset of a location step includes nodes that satisfy the predicate. You must write a predicate within square brackets. The comparison operators that a predicate uses to compare a node property with some specific value are: =, !=, >, <, >=, or <=. A node property can be the attribute value or sibling order value of a node, which the position() XPath function returns.

The following location step expressions use Listing 5-4 to demonstrate how to implement predicates:

- `child::subject [position()=2]`: Selects the second <subject> child node of the context node. If the context node is the first <semester> element node in the EC branch, the expression returns the second <subject> element node with the name attribute value, Analog Circuits.
- `preceding::branch [attribute::name]`: Selects the <branch> element nodes that possess a name attribute and precedes the context node. If the context node is the <subject> element node with the name attribute value, Microprocessor, the expression returns the <branch> element node with name attribute value, EC.

Each of the above expressions returns a nodeset that consists of a single node. XPath functions you define in a predicate can also return more than one node. For example, the following location step selects all <subject> element nodes that appear after position 1 and represent child nodes of the context node in Listing 5-4:

```
child::subject [position()>1]
```

If the context node is the <semester> element node with number attribute value, 1 in EC branch, the expression returns second and third <subject> element nodes in the nodeset.

Using XPath with PHP

Extensible Stylesheet Language (XSL) converts an XML document to another data format. Different XSL transformations use same XML document to generate different output, such as an HTML Web page.

Accessing XPath Using DOM

PHP uses a tree-based approach called DOM to parse an XML document. This approach helps create and manipulate the hierarchical tree structure of an XML document. You can implement DOM in any programming language, such as PHP, Java, and Visual Basic (VB). Implementing DOM in PHP, you can access an XML document using XPath. DOM is a parser that accesses and manipulates structured data by representing a document in the form of a tree hierarchy of objects.

Objects represent different structures that occur within an XML document. For example, the Element object represents elements and Attr objects represent attributes. Each object consists of standard properties and methods that navigate the object tree and access specific elements, attributes, or character data. The XPath processor can navigate a document tree using the parent-child relationship that exists between tree nodes. Node properties extract all information required from a document tree.

PHP consists of XPath classes that make the DOM parser flexible. The XPath classes build a collection of nodes that match the criterion specified in an XPath expression. The XPath classes available in PHP are XPathContext and XPathObject.

The XPathContext class sets up a context node for all XPath evaluations. You can create an XPathContext class object by calling the xpath_new_context() function. This function must be passed as a reference to a DOM object. The XPath evaluations provide instances of the XPathObject class.

The xpath_eval() method of the XPathContext class creates an instance of the XPathObject class. The xpath_eval() method accepts the XPath address as its argument. The xpath_eval() method returns an instance that contains the nodeset matching a specified XPath expression. As a result, the PHP XPath implementation to query an XML document parses the document into a DOM tree.

For example, the XML document, books.xml, lists information about books published by a publisher.

Listing 5-6 shows the content of the books.xml document:

Listing 5-6: The books.xml Document

```
<?xml version="1.0"?>
<books>
<book>
<title>Introduction to computers</title>
<publisher>Global Education Ltd.</publisher>
<author>John Mitchell</author>
<price>500</price>
<publishedYear>1993</publishedYear>
</book>
<book>
<title>C Programming</title>
<publisher>Global Education Ltd.</publisher>
<author>Taub Schiling</author>
<price>900</price>
<publishedYear>1996</publishedYear>
</book>
<book>
<title>Operating Systems</title>
<publisher>Global Education Ltd.</publisher>
<author>David Kennedy</author>
<price>1800</price>
<publishedYear>1993</publishedYear>
</book>
</books>
```

In the above listing, the books.xml document contains information about books, such as the book title, author, price, and date of publishing, published by the publisher, Global Education Ltd.

For example, you need to solve the query to search titles and authors of the books published by Global Education Ltd. in 1993. You can execute this XPath query using the PHP document, books.php, as shown in Listing 5-7:

Listing 5-7: Executing the XPath Query Using DOM Implementation in PHP

```
<?php>
$doc=xmldocfile("books.xml");
$xpath=$doc->xpath_new_context();
$output=$xpath->xpath_eval("/books/book[normalize-space(publisher/text())='Global Education
Ltd.' and normalize-space(publishedYear/text())='1993']");
$nodeset=$output->nodeset;
foreach ($nodeset as $node)
{
   foreach ($node->child_nodes() as $children)
   {
      if ($children->node_type()==XML_ELEMENT_NODE)
      {
         if ($children->tagname()=="title")
         {
```

```
            print ("Title:");
            foreach ($children->child_nodes()as $subcontent)
            {
               if ($subcontent->node_type()==XML_TEXT_NODE)
               {
                  print ($subcontent->content);
               }
            }
            print ("::");
         }
         if ($children->tagname()=="author")
         {
            print ("Author:");
            foreach ($children->child_nodes() as $subcontent)
            {
               if ($subcontent->node_type()==XML_TEXT_NODE)
               {
                  print ($subcontent->content);
               }
            }
            print ("<br />");
         }
      }
   }
}
?>
```

In the above listing, the PHP code, books.php, selects the title and author name of all books published by Global Education Ltd. in 1993 from the books.xml document. The xpath_eval() method evaluates the XPath expression provided as its argument.

Figure 5-3 shows the output of the PHP code in the books.php document:

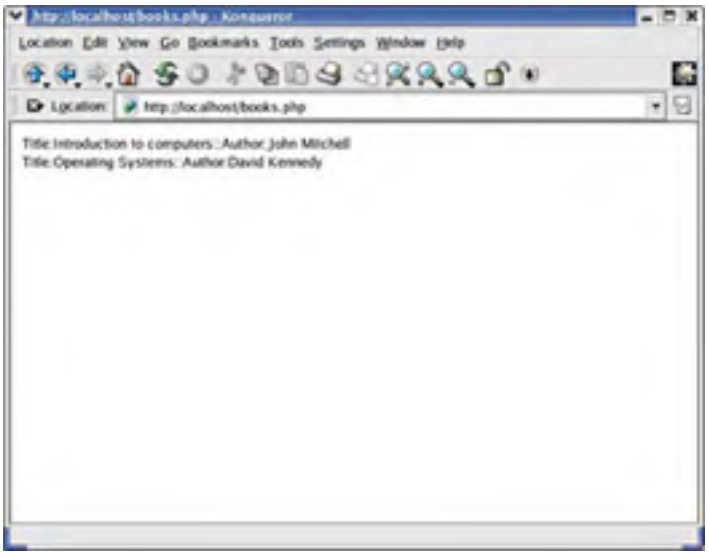


Figure 5-3: Output of the Code in the books.php Document

Note The PHP XPath implementation performs slowly for large documents and is resource intensive.

Accessing XPath Using SAX

SAX is an event-driven model for parsing an XML document. When the SAX parser encounters an XML construct, such as a tag, it generates an event. These events are passed to event handlers, which, in turn, provide access to the content of a document. You can parse an XML document using XPath expressions in PHPs SAX implementation.

You can set handlers for XML location paths using the XML parsing class, path parser (class_path_parser.php). For each handler, you can set up a PHP function that the parser calls whenever it finds an element matching the given location path. When the parser detects an element that matches the location path, the function accepts element names, attributes, and content. A function can handle multiple location paths.

For example, the XML document, employees.xml, contains information pertaining to employees in a company.

Listing 5-8 shows the content of the employees.xml document:

Listing 5-8: The employees.xml Document

```
<?xml version="1.0"?>
<employees>
<employee name='Peter'>
</employee>
<employeeId>e01</employeeId>
<department>administration</department>
<employee name='Julie'></employee>
<employeeId>e02</employeeId>
<department>sales</department>
<employee name='Taub'></employee>
<employeeId>e03</employeeId>
<department>administration</department>
<employee name='David'></employee>
<employeeId>e04</employeeId>
<department>Planning</department>
</employees>
```

You can implement SAX in PHP to retrieve XML elements that match the pattern described by an XPath expression. For example, you can retrieve the name of all employees from the employees.xml document.

Listing 5-9 shows the content of the employees.php document:

Listing 5-9: Retrieving Employee Names

```
<?php
include_once("/class_path_parser.php");
function result($name,$attribs,$content)
{
   print("$name  ");
   print($attribs [name]);
   {
      print ("<br/>");
   }
}
$parser = new Path_parser();
$parser->set_handler("/employees/employee", "result");
if(!$parser->parse_file("employees.xml"))
{
   print("Error:".$parser->get_error()."\n");
}
?>
```

The parse_file() method parses an XML document from a file or a URL. The syntax of the parse_file() method is:

```
boolean parse_file (string $xml)
```

In the above syntax, the parameter, $xml, is the name of the file or URL that contains the file that the parser needs to parse. The parse_file() method returns TRUE, if the file is successfully parsed, otherwise it returns FALSE.

> **Tip** If the parser does not successfully parse a file, the parse_file() method returns an error message. You can use the get_error() method to retrieve the error message.

The set_handler() method processes XML element nodes that match the pattern specified by an XPath expression. The syntax of the set_handler() method is:

```
set_handler (string $path, string $handlername)
```

The above syntax shows that the $path represents the absolute path in an XML document. The $handlername must have three arguments: $name, $attribs, and $content. The $name argument denotes the element name, $attribs denotes element attributes, and $content denotes the text within the element node.

Figure 5-4 shows the output of the PHP code, employees.php:

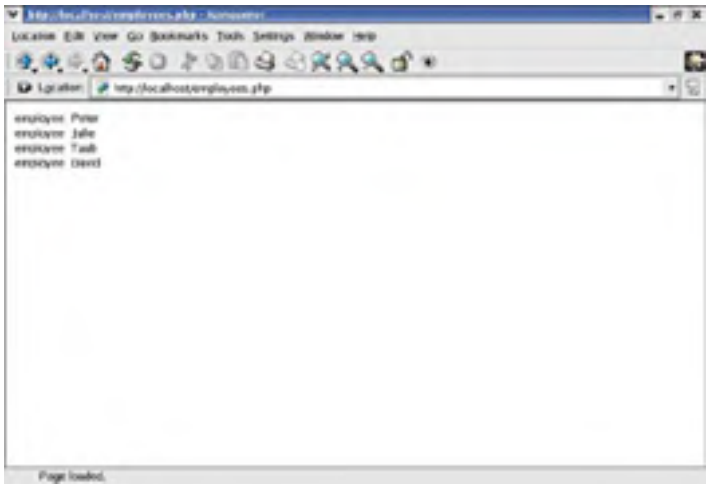


Figure 5-4: Output of the employees.php Code

You can parse an XML file using the Path_parser class and set handlers for specific XML elements defined by an XPath expression. The handler name can accept the element name, attributes, and the content.

Accessing XPath Using XSLT

XSLT is a language that can transform XML documents into a text-based format, such as HTML or another XML-based structure, such as the Web Distributed Data eXchange (WDDX) format. To perform an XSLT transformation, you require an XML document that XSLT should transform, XSLT stylesheet, and an XSLT engine.

The XSLT stylesheet contains the instructions that you need to write to achieve the required transformation. You use XSL, an XML-based language to create stylesheets. You need to define the output layout or result tree and the input document or source tree from where the XSL retrieves data. An XSLT engine transforms an XML document into other document types using a stylesheet. You can use XPath expressions in the XSLT stylesheet to retrieve elements from an XML document and transform these elements into the required format. For example, the XML document, chapter.xml contains information about the headings in a chapter.

Listing 5-10 shows the content of the chapter.xml document:

Listing 5-10: The chapter.xml Document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="chapter.xsl"?>
<chapter>
<h>Heading 1</h>
<h>Heading 2</h>
<h>Heading 3</h>
</chapter>
```

You can create a stylesheet file to convert the chapter.xml document to an HTML document.

Listing 5-11 shows the content of the chapter.xsl stylesheet file:

Listing 5-11: The chapter.xsl Stylesheet File

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<HTML>
<BODY>
<xsl:for-each select="/chapter/h">
<S><xsl:value-of select="."/></S>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

In the above listing, <xsl: for-each> applies to any <h> elements, which are the child nodes of a <chapter> root element. The output contains <S> elements with the value of <xsl:value-of> element as its content.

You need to apply the stylesheet to the chapter.xml document.

Listing 5-12 shows output of the chapter.xml file after you apply the stylesheet to the chapter.xml document:

Listing 5-12: Output After Applying the Stylesheet

```
<HTML>
<BODY>
<S>Heading 1</S>
<S>Heading 2</S>
<S>Heading 3</S>
</BODY>
</HTML>
```

The above listing shows the HTML format for the chapter.xml document.

XPath expressions enable you to specify the location in an XML document and process this information using XSLT. You can process specific XML elements using XPath expressions with DOM, SAX, or XSLT. Parts of XSLT operate using a subset of XPath. You can use this subset to test if an XPath node matches a pattern defined by the location path. XPath operates only on the logical structure of an XML document.

Chapter 6: PHP and XML-Remote Procedure Calls

Download CD Content

Extensible Markup Language-Remote Procedure Call (XML-RPC) is a protocol that allows applications running on various operating systems to make procedure calls on a remote server for executing a procedure.

Implementing XML-RPC in PHP helps build network-based services. You can implement XML-RPC in PHP to encode and decode Remote Procedure Call (RPC) messages and create and manage a remote server. This implies that you can use PHP scripts to invoke procedures on remote servers by transferring messages that are in the XML format.

You can access different remote procedures directly from a PHP script using standard client-server protocols, such as HyperText Transfer protocol (HTTP). After executing a remote procedure, the remote server returns a value, which can be used in other applications.

This chapter explains the XML-RPC protocol, and how to create and transfer a request to a server using XML-RPC. This chapter also explains the data types that the XML-RPC protocol supports.

Understanding XML-RPC

RPC provides a client-server framework that allows methods or procedures on the server to be remotely executed by the client in a secure and efficient manner. It uses HTTP to transfer data between the client and server. The client that invokes the procedure may be either on the same computer as the server or a different computer on the network. You can implement RPC over the Web using XML encoding and the HTTP protocol for transferring data.

The XML-RPC protocol encodes remote procedure calls in XML. It uses HTTP to transfer client requests to a server and receive the responses. XML-RPC uses an XML format to pass the method parameters and receive the returned values from the server. XML-RPC implements client requests and server responses in various languages, such as Lisp, JavaScript, C, Perl, and PHP. XML-RPC supports various data types, such as integer, string, Boolean, and double.

Introducing RPC

RPC is a programming interface that allows a program to use the services of another program available on a remote computer. The calling program transfers a message to the remote program, which executes the procedure and returns the result to the calling program.

Figure 6-1 shows the communication in a network using RPC:

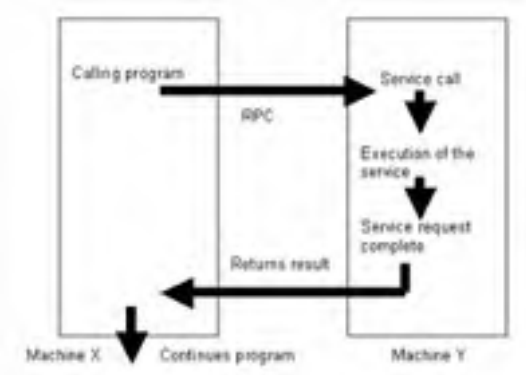


Figure 6-1: Network Communication using RPC

The client process, Computer X, transfers a request to the Computer Y server process, and passes the parameters for the called procedure. The server executes the called procedure and returns the result to the client.

RPC consists of three components, which are:

- An Application Programming Interface (API): Handles procedure calls on the client-side and executes client requests on the server-side.
- A set of rules: Encode and decode RPC requests and responses.
- A network transmission layer: Establishes communication between the client and server.

RPC provides a framework to execute the procedures on a remote computer in an efficient manner. To make a remote procedure call:

1. The RPC client invokes a remote procedure.
2. The RPC client encodes the procedure call request, along with its parameters, in a form suitable for transmission over a network transmission layer, such as HTTP.
3. The network transmission layer transfers the procedure call request to the RPC server in the form of packets.

4. The RPC server extracts the procedure name and its parameters from the request packet.
5. The RPC server invokes the specified procedure and obtains the result.
6. The RPC server encodes the result of the procedure in the form of response packets and transmits the response packets to the RPC client.
7. The RPC client receives the response packet, decodes it, and extracts the return value of the procedure. The RPC client can use the value that the called procedure returns in other programs.

There are two methods to implement an RPC, XML-RPC and Simple Object Access Protocol (SOAP). To create an XML-RPC call, the client issues a request to the server specifying a method name, its parameters, and the server name. The methods that you define in PHP are called XML-RPC methods and the parameters correspond to the arguments that you pass to these XML-RPC methods. The arguments can be of any data type, such as integer and Boolean.

The client packages the request in the XML format and issues an HTTP-POST request that transfers the message to the server. The HTTP-POST request informs the server that the client is ready to transfer data. The server calls the requested method and passes the parameters to this method. The method on the server returns a response, and the server packages the response into the XML format. The server returns the response to the client, which parses the XML package to retrieve the returned value. The value that a remote method returns is called server response.

SOAP is a specification for creating structured data packets that an application needs to transfer across a network. SOAP enables information exchange among the computers in a network. Unlike XML-RPC, it is not necessary that SOAP returns the HTTP status code, 200 OK, to indicate the server response. Instead, the SOAP specification uses standard HTTP error code to identify the successful processing of a client request.

Working with the XML-RPC Protocol

The XML-RPC protocol is a specification with a set of implementations that allow software running on different operating systems to make remote procedure calls. The XML-RPC request and response that you create consists of an HTTP header and an XML body. The use of the HTTP protocol ensures that XML-RPC client requests are synchronous and stateless.

A synchronous XML-RPC client request means that the server immediately responds to the client request. The XML-RPC server transfers the response on the same HTTP layer as the client request. A stateless XML-RPC client request means a client request does not preserve any information from one request to another. For example, if a client invokes a remote method on a server, receives the server response, and again invokes the same method on the server, the client requests are regarded as two different requests.

The XML-RPC client does not perform any other function until it receives the server response. A client request can invoke only a single remote method and each server response can return only a single value. The single value that the XML-RPC server returns can be either an array or a structure with multiple values. There are two types of XML-RPC servers:

- Mini Web serverM: Processes an XML-RPC request.
- XML-RPC listener: Assists the Web server in processing an XML-RPC request.

XML-RPC packages data for information exchange between two computers by defining the structure and format of the procedure calls and responses. XML-RPC is useful in applications where a server provides various services, such as remote procedure access to its clients. To use XML-RPC, you need to install the XML-RPC package on your computer.

Figure 6-2 shows the architecture of XML-RPC:

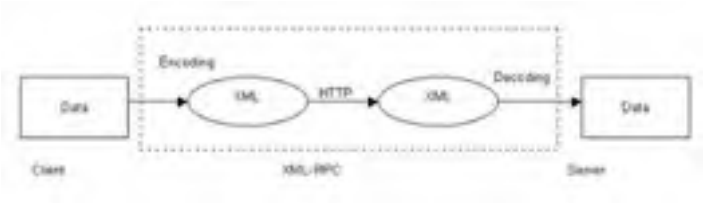


Figure 6-2: Architecture of XML-RPC

The data types that XML-RPC supports are:

- `<boolean>` : Represents Boolean values, 0 or 1. The following syntax shows how to represent the <boolean> data type in XML-RPC:

  ```
  <value><boolean>n</boolean></value>
  ```

  In the above syntax, n represents a Boolean value.

- `<double>`: Represents floating-point data. The following syntax shows how to represent floating-point data in XML-RPC:

  ```
  <value><double>n</double></value>
  ```

  In the above syntax, n denotes a floating-point value.

- `<int> or <i4>`: Represents signed 32-bit integers. You can represent the integer data type in XML-RPC using the following syntax:

  ```
  <value><i4>n</i4></value>
  <value><int>n</int></value>
  ```

  In the above syntax, n denotes an integer value.

- `<string>`: Represents the ASCII string. The following syntax shows how to represent the string data type in XML-RPC:

  ```
  <value>s</value>
  ```

  In the above syntax, s denotes a string value. The string data type is the default value. If you do not indicate the variable data type, the value inside the <value> tag is called as the string value.

- `<base64>`: Represents binary data encoded with BASE64. The following syntax shows how to represent the binary data type in XML-RPC:

  ```
  <value><base64>n</base64></value>
  ```

  In the above syntax, n denotes a binary value.

- `<array>`: Represents an array of data. The <array> element contains a single <data> element, which consists of many <value> elements. The following syntax shows how to represent the <array> element in XML-RPC:

  ```
  <value>
  <array>
  <data>
  <value>n</value>
  ......
  <value>n</value>
  </data>
  </array>
  </value>
  ```

- `<struct>`: Represents an associative array that contains string values. It contains many <member> elements, with each element containing a <name> and a <value> element. The following syntax shows how to represent the <struct> element in XML-RPC:

  ```
  <value>
  <struct>
  <member>
  <name>n</name>
  <value>v</value>
  </member>
  ....
  <member>
  <name>n</name>
  <value>v</value>
  </member>
  </struct>
  </value>
  ```

- `<dateTime.iso8601>`: Represents the date and time according to the International Standard Organization (ISO) 8601 standard. The following syntax shows how to represent date and time in XML-RPC:

  ```
  <value><dateTime.iso8601>YYYYMMDDTHH:MM:SS</dateTime.iso8601></value>
  ```

Note An XML-RPC array data type is similar to an indexed array, and an XML-RPC struct data type is similar to an associative array. For example, the array data type is represented as $array=array (‘one’, ‘two’), and the struct data type is represented as $struct=array (‘var1’=’one’, ‘var2’=’two’).

Listing 6-1 shows the example of an XML-RPC request:

Listing 6-1: XML-RPC Client Request

```
POST /myserver.php HTTP/1.0
User-Agent: Konqueror
Host: 192.148.0.146
Content-Type: text/xml
Content-Length: 256
<?xml version="1.0"?>
<methodCall>
<methodName>getCharacter</methodName>
<params>
<param>
<value>
<string>Phoenix</string>
</value>
</param>
</params>
</methodCall>
```

In the above listing:

- The server transfers the request to the Uniform Resource Identifier (URI), /myserver.php.
- The client calls the getCharacter() method on the server to obtain the number of characters in the Phoenix file.
- The <methodCall> structure consists of the <methodName> element, which specifies the name of the method on the remote server that an XML-RPC client needs to call, which is the getCharacter() method. The <methodName> element is a string that can contain characters such as alphabets, numerics, underscore, period, colon, or slash.
- The server interprets the <methodName> string either as the location of a file or name of a file that contains the program that a client requests for execution.

Note The client request should contain the <params> element even if a remote method does not need any parameters.

The <methodCall> structure includes the <params> element to indicate the parameters that a remote method accepts. The <param> element represents individual arguments of a remote method. The User-Agent specifies the medium through which the client and server communicate. The Host specifies the IP address of the server. In addition to the User-Agent and Host, you should also indicate the Content-Length, which represents the accurate character length of the message.

The path that you describe after the POST method indicates the script that receives the client data. The HOST attribute, in the header of the XML-RPC client request, indicates the name of the server that services the XML-RPC request.

Note You can download the XML-RPC package for PHP either in a zipped format or tar.gz archives from the following Web site: http://xmlrpc.usefulinc.com/php.html.

The XML-RPC server decodes and interprets the client request and provides result to the client. If no error occurs while processing a client request, the XML-RPC server returns the response. Every XML-RPC response should return the 200 OK HTTP status code even if an error occurs while processing the XML-RPC client request. The <methodResponse> structure returns a single value and so the <params> element consists of a single <param> element.

Listing 6-2 shows the example of an XML-RPC response:

Listing 6-2: XML-RPC Server Response to a Successful Client Request

```
HTTP/1.0 200 OK
Connection: close
Server: 192.168.0.146/myserver.php
Content-Type: text/xml
Content-Length: 210
<?xml version="1.0"?>
<methodResponse>
<params>
<param>
<value>
<string>100</string>
</value>
</param>
</params>
</methodResponse>
```

The above listing shows the value that the getCharacter() method returns to the XML-RPC client.

If the server does not process a client request successfully, the <fault> element replaces the <params> element in the <methodResponse> structure.

Listing 6-3 shows the XML-RPC response to an unsuccessful request:

Listing 6-3: XML-RPC Server Response to an Unsuccessful Client Request

```
HTTP/1.0 200 OK
Connection: close
Server: 192.168.0.146/RPC
Content-Type: text/xml
Content-Length: 210
<?xml version="1.0"?>
<methodResponse>
<fault>
<value>
<struct>
<member>
<name>faultCode</name>
<value>
<int>x</int>
</value>
</member>
<member>
<name>faultString</name>
<value>
<string>No characters in the file</string>
</value>
</member>
</struct>
</value>
</fault>
</methodResponse>
```

In the above listing:

- The <fault> element replaces the <params> element, which appears in the server response to a successful client request.
- The <fault> element contains the x fault code along with the fault string that describes the type of fault. The fault string indicates that there are no characters in the Phoenix file.
- The <struct> element contains two elements, faultCode and faultString.

- The faultCode element is a numeric value corresponding to a fault.
- The faultString element is a string that contains the text description of the fault.

Note The <methodResponse> structure cannot simultaneously include the <params> and <fault> elements.

Extending the XML-RPC Protocol

Implementing XML-RPC in PHP provides the introspection feature. The introspection feature lists and describes the services that an XML-RPC server provides. It provides a way to query methods and obtain the description of methods from the server. PHP XML-RPC provides the following introspection features:

- `system.listMethods()`: Generates a list of methods available with the XML-RPC server.
- system.describeMethods(): Describes the server methods. The description includes the expected arguments for the method, method return type, and optional arguments for the method.
- `system.methodHelp()`: Generates documentation for a particular method. The system.methodHelp() method accepts the name of a method that the XML-RPC server implements as its argument and returns a string that describes the use of that method.
- `system.methodSignature()`: Returns the signature for a particular method. The signature describes the data type of the parameters that a client should pass to a method. It also indicates the return type for a method.

In addition to features that the XML-RPC protocol supports, the protocol can support features, such as an asynchronous client request and authentication of Web services. The three extensions of XML-RPC protocol are:

- Asynchronous protocol for XML-RPC
- State-preserving protocol for XML-RPC
- A protocol with authentication for XML-RPC

In the asynchronous XML-RPC protocol, the server does not immediately transfer the response to a client request. The server transfers the response to the client at a time other than the time when the client makes a request. For implementing the asynchronous XML-RPC protocol, both the XML-RPC client and server process should be able to perform the functions of each other. This means that the client process can perform the functions of the server process and the server process can perform the functions of the client process. The client and server processes should contain the code for handling asynchronous responses.

To create a system that uses the asynchronous XML-RPC protocol, the server process should return a unique identifier. To communicate using the asynchronous XML-RPC protocol:

1. The client makes a request to the server.
2. The server returns an ID for the request without processing it.
3. The client receives the ID and continues its current operation.
4. The server processes the client request and creates a result container.
5. The server makes a call to the client by passing the ID and the result of the client request.
6. The client receives the response of the client request.
7. The client processes the response.

The XML-RPC protocol is stateless and does not preserve any information from one client request to another. However, for applications, such as Web-based shopping carts, it is necessary to preserve the server state for use in another client request. For example, a client request creates a particular method on a server and another client request requires a change in the created method. In such situations, it is necessary to preserve the server state. To create subsequent requests that utilize the server states, the client uses the identifiers held either in cookies or in the Uniform Resource Locator (URL) of the page.

You can implement a state preserving system in XML-RPC by representing the first argument of all the remote procedure calls as a session identifier. XML-RPC keeps track of the client requests, and uses the information of an earlier request in the consequent request.

The authentication protocol for XML-RPC ensures that the message content is from an authorized client. As a result, the message cannot be modified during transfer through the HTTP transport layer. The authentication protocol is based on Message Authentication Code (MAC), which is a keyed form of the request message. This means that the client attaches a secret code to the request message.

For many-to-many client/server relationships, it is not possible to maintain databases on each server. As a result, both the client and server maintain keys with a keyserver that authenticates the clients to the servers. If your Web server supports HTTP, the client can invoke a remote method on the server by supplying a user name and password that are validated by the server.

Using XML-RPC for Web Services

A Web service is an application that runs on a Web server and enables the client programs to call remote procedures using the HTTP transport layer. The process of data exchange is similar in Web services and Web browsers. The Web browser transfers a request to a Web site in the HTML form and obtains a Web page in response. A Web service uses the XML format while a browser uses the HTML format to transfer data.

A Web service packages a collection of functions as a single entity, and publishes these functions over the Web to be used by other programs.

A Web service contains three components: service provider, service discovery agency, and service requestor. The service requester requests the service provider for the execution of a Web service. The service requester searches the description through the discovery agency and uses the service description to bind with the service provider to interact with the Web service. The service provider processes a Web service request, provides the Web service description, and publishes it to a requester or discovery agency. The service discovery agency publishes a Web service description to multiple service registries.

Figure 6-3 shows the Web service architecture:

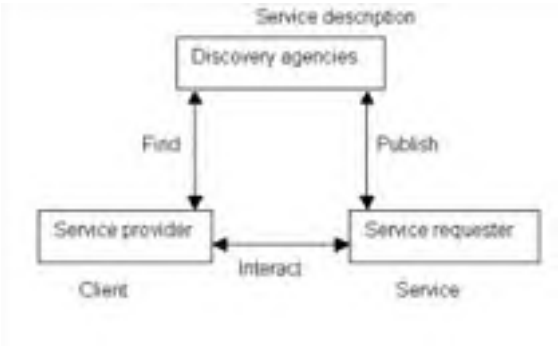


Figure 6-3: The Web Service Architecture

The components of a Web service interact to perform the following functions:

- Define the interface and invocation methods of a Web service.
- Publish the Web service to the intranet or Internet storage locations so that the end users can easily locate the service.
- Locate a Web service.
- Invoke a Web service for use by end users.

A Web service has the following advantages:

- Provides interoperability that helps to execute Web services regardless of the platform used.
- Permits just-in-time integration by dynamically locating and invoking the Web service.
- Allows a change in either a Web service or the application that uses it, without affecting the other components.

XML-RPC helps to access Web services for many end users. A Web service executes procedures available on a remote computer from another computer. It provides interoperability among applications running on different computers. For example, there are paid Web services, where you need to pay a prescribed fee for information exchange. An application can access a Web service over the Internet using standard Web protocols, such as HTTP. A Web service is heterogeneous because it allows communication between a client and server running on different platforms.

The XML-RPC protocol acts as a middleware between Web clients and remote Web services. XML-RPC accepts client requests, translates the request into an XML format, and transfers it using the HTTP protocol to access the remote Web service.

Figure 6-4 shows how to access a Web service using the XML-RPC protocol:

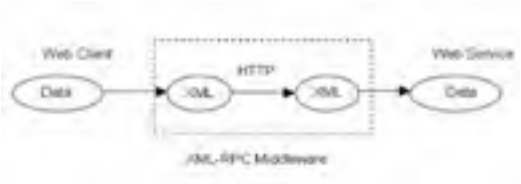


Figure 6-4: Accessing a Web Service Using the XML-RPC Protocol

PREVIOUS NEXT

Implementing XML-RPC in PHP

The XML-RPC client and server classes exist for various languages including PHP. The xmlrpc.inc file provides XML-RPC client functions, and the xmlrpcs.inc file provides XML-RPC server functions. To check if the XML-RPC server successfully executes a client request, include the .inc files in the same directory as the PHP code.

Creating a Client Application

The XML-RPC client transfers a request to execute a particular method on the server. The request message consists of the arguments that the remote method accepts.

To create an XML-RPC client request using the XML-RPC class, the client needs to generate a client object. The client transfers and receives data from the XML-RPC server using the client object. The xmlrpc_client class creates a client object. The information that the client object constructor receives from the server includes:

- The path to the script handling that the XML-RPC client requests.
- The hostname or the IP address of the server.
- The port number of the server.

The syntax of the xmlrpc_client class constructor is:

```
$client=new xmlrpc_client($serverpath, $hostname, $port);
```

Note The port parameter in the xmlrpc_client class constructor is optional. If you exclude the port parameter, its default value is set to 80 when you use HTTP as the transport layer.

The xmlrpc_client class constructor supports the following methods:

- `send() method`: Sends the XML-RPC client message to the server. The syntax of the send() method is:

  ```
  $response=$client->send ($xmlrpc_message, $timeout, $server_method);
  ```

  In the above syntax, $xmlrpc_message is an instance of the xmlrpcmsg class. The $timeout and $server_method parameters are optional. The $timeout parameter is set to 0, and the $server_method parameter to HTTP, if these are not specified in the syntax.

- `setDebug() method`: Enables the client to display the debugging information on the browser. When debugging mode is on, the client object displays the response received from the server. The debugging information helps you locate low-level errors, such as no network connection to the server and the timeout expired errors. You can locate these errors because debugging information displays data that the server returns. The syntax of the setDebug() method is:

  ```
  $client->setDebug($debugmode);
  ```

  ```
  $client->setDebug($debugmode);
  ```

  In the above syntax, the value of the $debugmode parameter is either 0 or 1. The value, 0, of the $debugmode parameter indicates that the client does not display debugging information on the browser. If the $debugmode parameter is set to 1, the client displays the debugging information on the browser.

- `setCredentials() method`: Enables you to set a user name and password for authorizing the client to access a server. The syntax of the setCredentials() method is:

  ```
  $client->setCredentials($user name, $password);
  ```

Note The xmlrpc_client class constructor supports the setCertificate() method, which allows you to define a certificate necessary for establishing a connection over the Hyper Text Transfer Protocol Secure Sockets (HTTPS) protocol.

To create a client request, you can invoke the methods defined in the sample file, server.php, which is downloaded with the xmlrpc library. For example, you can create a client request that calls the interopEchoTests.echoValue method in the server.php file. The interopEchoTests.echoValue method echoes the value passed to it. The interopEchoTests.echoValue method does not accept any parameters.

Listing 6-4 shows an XML-RPC client request created using the PHP language:

Listing 6-4: XML-RPC Client Request in PHP

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
$msg=new xmlrpcmsg("interopEchoTests.echoValue");
$client=new xmlrpc_client("/server.php", "localhost", 80);
$client->setDebug(1); //Debug mode is turned on.
$response=$client->send($msg); //Invokes the remote method.
?>
```

The above listing shows how the client request calls the interopEchoTests.echoValue() method on the server.php server file. The client message does not pass any arguments to the called method. As a result, the server response consists of the client message itself.

Instead of calling methods predefined in the server.php file, you can create a server and define methods within it. For example, create a server, myserver.php, which contains the examples.multiply() method to multiply two integer values.

Listing 6-5 shows an XML-RPC client request in PHP to call the examples.multiply() method:

Listing 6-5: XML-RPC Client Request that Invokes the examples.multiply() Method

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
$msg=new xmlrpcmsg("examples.multiply", array(new xmlrpcval(13, "int"),
new xmlrpcval(2, "int")));
$client=new xmlrpc_client("/myserver.php", "localhost", 80);
$client->setDebug(1);
$response=$client->send($msg);
?>
```

The above listing shows the client request to invoke the examples.multiply() method available in the myserver.php file. The client request passes two integer parameters, 3 and 2, to this method.

Similarly, you can create a client request to add three floating-point values.

Listing 6-6 shows the client request to add three floating-point numbers:

Listing 6-6: The Client Request to Add Three Floating-point Numbers

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
$msg=new xmlrpcmsg("addDouble", array(new xmlrpcval(3.4, "double"),
new xmlrpcval(2.5, "double"), new xmlrpcval(1.2, "double")));
$client=new xmlrpc_client("/myserver.php", "localhost", 80);
$client->setDebug(1);
$response=$client->send($msg);
?>
```

The above listing shows the client request that passes three parameters of double data type to the remote method on the myserver.php server.

Figure 6-5 shows the server response after servicing the request to add three floating-point values:

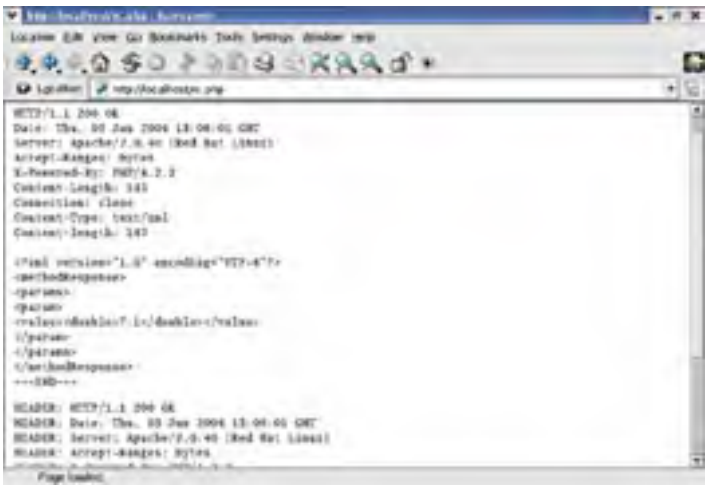


Figure 6-5: The Server Response of the Client Application to Add Double Values

Creating a Request using XML-RPC

After you create the client object, you should pass a message object to it. The XML-RPC client transfers the message object to the server. The message consists of the name of the remote method that a client needs to invoke, and the parameters that this method accepts. To create a client request, you need to create an instance of the xmlrpcmsg class. The xmlrpcmsg class provides the method name along with its parameters that an XML-RPC client needs to invoke on a server. The syntax of the xmlrpcmsg class is:

```
$msg=new xmlrpcmsg($method_name, $parameter_array);
```

In the above syntax, the $method_name parameter is the string that denotes the name of the method to be invoked on the server. The $parameter_array parameter is an array of objects that the xmlrpcval class creates.

An alternative method to represent the xmlrpcmsg class constructor is:

```
$msg=new xmlrpcmsg($method_name);
```

In the above syntax, the client does not pass any parameters to the remote method.

The xmlrpcval class encloses values of the specified data types to obtain the XML-RPC server response. It stores complex values using the XML-RPC data types. The following multiple forms show the use of the xmlrpcval class:

```
$val=new xmlrpcval();
$val=new xmlrpcval($stringvalue);
$val=new xmlrpcval($scalarvalue, "int"|"boolean"|"string"|"double"|"dateTime.iso8601"|"base64");
$val=new xmlrpcval($arrayvalue, "array"|"struct");
```

In the above forms, the first xmlrpc class constructor creates an empty value. The second constructor creates a string value. The third constructor consists of two parameters. The first parameter of the third constructor indicates that the xmlrpcval class creates a scalar value. The second parameter denotes the XML-RPC data type. The fourth constructor contains the $arrayvalue parameter, which is either a simple array or an associative array, depending upon the data type.

You should provide parameters within an array, even if the client has only one parameter to pass to the remote method. It is not necessary to provide parameters for the remote method during the initialization of the xmlrpcmsg class instance. You can use the addParam() method to add the method parameters after the XML-RPC client creates the xmlrpcmsg class instance. The syntax of the addParam() method is:

```
$msg->addParam($xmlrpcval);
```

For example, the code snippet showing the client request using the XML-RPC library classes, xmlrpcmsg and xmlrpcval, is:

```
$msg=new xmlrpcmsg("calculate.area", array(new xmlrpcval(14, "int"),
new xmlrpcval(23, "int")));
```

The above code shows the client request that invokes the calculate.area() method to calculate the area of a rectangle. The client passes two integer parameters, 23 and 14, which denote the length and breadth of the rectangle.

You can also provide a parameter array to the remote method. For example, the following code shows the client request that transfers an array of parameters to the remote method, examples.salary:

```
$input=array("Kim"=>2300, "Erwin"=>5000, "Crisp"=>9000);
while(list($key, $val)=each($input))
{
   $outp[ ]=new xmlrpcval(array("name"=>new xmlrpcval($key),"sal"=>new xmlrpcval($val,
   "int")), "struct");
}
$f=new xmlrpcmsg("examples.salary",array(new xmlrpcval($outp, "array")));
```

The above code shows the client request that invokes the examples.salary() method to sort the employee names on the basis of the salary. The input parameters passed to the remote method are the name and salary of the employee.

Sending a Request to the Server

After the client creates the client object and request message, it transfers the request message to the server. To transfer the client request to a remote server, the client creates a request message and passes it to the send() method. There are two ways to invoke the send() method:

```
$xmlrpc_resp=$client->send($msg);
$xmlrpc_resp=$client->send($msg, $timeout);
```

In the above syntax, the second form describes a timeout parameter after which the message is timed out. You should represent the timeout parameter in seconds. The $msg parameter specifies the method name and its parameters.

If the client request executes successfully, the send() method returns an instance of the xmlrpcresp class. If a low-level error occurs, the send() method returns 0. The low-level error occurs either if the client cannot connect to the server, or if the client cannot transfer the request to the server. The xmlrpcresp class contains the responses to the XML-RPC requests.

> **Note** Turn on the client object debugging mode to check the cause of the low-level error.

An XML-RPC server is a PHP script similar to an XML-RPC client. The XML-RPC server contains methods that a client can access remotely. It is not necessary that the name of the method, which a client uses, should be the same as the method name on the server. PHP maps the Web service API procedure names, and the name of the PHP functions that implement those procedures on the server. For example, the examples.multiply() method that the client invokes is mapped to the multiply() method on the server. The following code shows the multiply() method on the XML-RPC server:

```
function multiply($m)
{
   $s=$m->getParam(0);
   $t=$m->getParam(1);
   return new xmlrpcresp(new xmlrpcval($s->scalarval()*$t->scalarval(), "int"));
}
```

In the above code:

- The multiply() method accepts the xmlrpcmsg class instance as a single argument.
- The getParam() method accesses the parameters of the multiply() method as instances of the xmlrpcval class in the request message.
- An instance of the xmlrpcresp class returns the value of the multiply() method to the calling function.

You can create an instance of the xmlrpcresp class using one of the following syntaxes:

- `new xmlrpcresp (0, $errno, $errmsg)`: Contains three arguments. The first argument, 0, indicates that the value that a server returns is a fault condition. The second argument, $errno, indicates the fault number and the third parameter, $errmsg, indicates the error description.
- `new xmlrpcresp ($xmlrpcVal)`: Returns the value of the method if the client request is successfully executed.

To obtain the response to an XML-RPC client request, the server should execute the invoked method. A dispatch map stores the description of all methods defined in the XML-RPC server. A dispatch map is an associative array that maps the Web service API procedure names with the names of the PHP methods that implement those procedures on the server. The following code shows the dispatch map for the client request that calls the examples.multiply() method on the server:

```
//Represents the signature of the multiply() method.
$multiply_sig=array(array($xmlrpcInt, $xmlrpcInt, $xmlrpcInt));
//Represents the documentation for the multiply() method.
$multiply_doc='Multiplies two integers together and return an result type';
//Represents dispatch map.
$dm=array("examples.multiply"=>array("function"=>"multiply", "signature"=>$multiply_sig,
"docstring"=>multiply_doc));
```

The above code shows the dispatch map for the multiply() method. The dispatch map points to the signature of the multiply() method, along with the documentation for the multiply() method. PHP maps the examples.multiply() method to the PHP function that implements the called procedure on the server.

Each instance of the array in the dispatch map needs the following parameters:

- `function`: Represents the exact name of the PHP function on the server that defines the functioning of the method that the XML-RPC client calls. It is a string value.
- `signature`: Represents the data type of the parameters, along with the method return type, that you need to pass to the method. You can define multiple signatures so that a method can have different number and type of parameters. When the client invokes a particular method, the server verifies if the data type of the arguments passed by the client matches any of the signatures in the dispatch map. An error occurs if the data type of the arguments does not match.
- `docstring`: Represents a string that contains the documentation for a method on the server. The method documentation explains the operation that a method performs.

You need to pass the dispatch map as an instance of the xmlrpc_server class. To get the client request serviced by a server:

1. Include the xmlrpcs.inc file, in addition to the xmlrpc.inc file, in the server PHP script.
2. Create the xmlrpc_server class instance that accepts the dispatch map as its argument.

The xmlrpc_server class instance services the client request.

Listing 6-7 shows the PHP server script that services the client request to multiply two integer values:

Listing 6-7: PHP Server Script to Multiply Two Integers

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
include("/xmlrpc-1.0.99.2/xmlrpcs.inc");
$multiply_sig=array(array($xmlrpcInt, $xmlrpcInt, $xmlrpcInt));
$multiply_doc='Multiplies two integers together and return the result';
function multiply($m) {
$s=$m->getParam(0);
$t=$m->getParam(1);
return new xmlrpcresp(new xmlrpcval($s->scalarval()*$t->scalarval(), "int"));
}
$res=new xmlrpc_server(array("examples.multiply"=>array("function"=>"multiply",
"signature"=>$multiply_sig, "docstring"=>multiply_doc), "interopEchoTests.echoValue"
=>array("function" => "i_echoValue", "docstring" => $i_echoValue_doc)));
?>
```

The above listing shows the server script that services the client request to multiply two integer values.

Similarly, you can create the server that arranges employee names based on their salaries.

Listing 6-8 shows the PHP server script that services the client request to arrange employee names based on their salaries:

Listing 6-8: PHP Server Script to Sort Employees

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
include("/xmlrpc-1.0.99.2/xmlrpcs.inc");
function salary_compare($a, $b)
{
   global $salary_arr;
   $a=ereg_replace("-", "", $a);
   $b=ereg_replace("-", "", $b);
   if ($salary_arr[$a]==$salary[$b]) return 0;
   return ($salary_arr[$a] > $salary_arr[$b]) ? -1 :1;
}
$salary_sig=array(array($xmlrpcArray, $xmlrpcArray));
$salary_doc='Send this method an array of [string, int] structs';
function salary($m)
{
   global $salary_arr, $xmlrpcerruser, $s;
   xmlrpc_debugmsg("Entering 'salary'");
   // get the parameter
   $sno=$m->getParam(0);
   // error string for [if|when] things go wrong
   $err="";
   // create the output value
   $v=new xmlrpcval();
   $agar=array();
   if (isset($sno) && $sno->kindOf()=="array")
   {
      $max=$sno->arraysize();
```

```
        // print "<!-- found $max array elements -->\n";
        for($i=0; $i<$max; $i++)
        {
           $rec=$sno->arraymem($i);
           if ($rec->kindOf()!="struct")
           {
              $err="Found non-struct in array at element $i";
              break;
           }
           // extract name and salary from struct
           $n=$rec->structmem("name");
           $a=$rec->structmem("sal");
           // $n and $a are xmlrpcvals,
           // so get the scalarval from them
           $agar[$n->scalarval()]=$a->scalarval();
         }
         $salary_arr=$agar;
         uksort($salary_arr, salary_compare);
         $outAr=array();
         while (list( $key, $val ) = each( $salary_arr ) )
         {
             // recreate each struct element
             $outAr[]=new xmlrpcval(array("name" =>  new xmlrpcval($key), "salary" => @ new
             xmlrpcval($val, "int")), "struct");
          }
          // add this array to the output value
          $v->addArray($outAr);
    }
   else
    {
      $err="Must be one parameter, an array of structs";
    }
   if ($err)
    {
      return new xmlrpcresp(0, $xmlrpcerruser, $err);
    }
   else
    {
      return new xmlrpcresp($v);
    }
}
$res=new xmlrpc_server(array("examples.salary"=>array("function"=>"salary",
"signature"=>$salary_sig, "docstring"=>salary_doc)));
?>
```

The above listing shows that the PHP server script services the client request to sort employee names based on their salaries. The client calls the examples.salary function, and PHP maps the called function to the salary() method on the server. The salary() method calls the salary_compare() method to compare the salaries of the employees.

XML-RPC messages support various data types. You can create a client request that passes floating-point data as the method parameter.

Listing 6-9 shows the PHP server script that adds three double data type values:

Listing 6-9: PHP Server Script to Add Double Values

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
include("/xmlrpc-1.0.99.2/xmlrpcs.inc");
$add_sig=array(array($xmlrpcDouble, $xmlrpcDouble, $xmlrpcDouble, $xmlrpcDouble));
$add_doc='Add three double values and return a value with type double.';
function add($m)
{
   $s=$m->getParam(0);
   $t=$m->getParam(1);
   $r=$m->getParam(2);
   return new xmlrpcresp(new xmlrpcval($s->scalarval()+$t->scalarval()+$r->scalarval(),
   "double"));
}
$resp=new xmlrpc_server(array("addDouble"=>array("function"=>"add",
"signature"=>$add_sig, "docstring"=>$add_doc)));
?>
```

The above listing shows that PHP maps the addDouble() function to the add() function in the PHP script. The signature of the add function specifies that the function accepts three double values as input and returns a single double type value.

Processing the Server's Response

The XML-RPC server returns a response when it successfully completes a transaction with the client object. The response that the server returns denotes an instance of the xmlrpcresp class. If the XML-RPC server does not complete a successful transaction with the client object, it returns a fault condition. As a result, the instance of the xmlrpcresp class is either a normal response or a fault condition. The XML-RPC response is a fault condition either if the server does not understand the client request or if there is any low-level error.

The xmlrpcresp class provides the following two methods to check if the server response is a fault condition:

- `faultCode() method`: Returns 0, if there is no fault condition. If a fault occurs, it returns the ID number that the server assigns to the fault.
- `faultString() method`s: Returns the description of the fault that occurs.

If the server successfully executes the client request, the instance of the xmlrpcresp class returns the data from the called procedure. The value() method accesses data that an instance of the xmlrpcresp class returns. The syntax of the value() method is:

```
$xmlrpcVal=$resp->value();
```

In the above syntax, $xmlrpcVal is an instance of the xmlrpcval class that contains the value that the server returns. The client process does not use the value that the value() method returns, if the faultCode is nonzero.

If the client request to multiply two integer values executes successfully, the server response returns the required result.

Figure 6-6 shows the server response to the client request for multiplying two integers, 13 and 2:

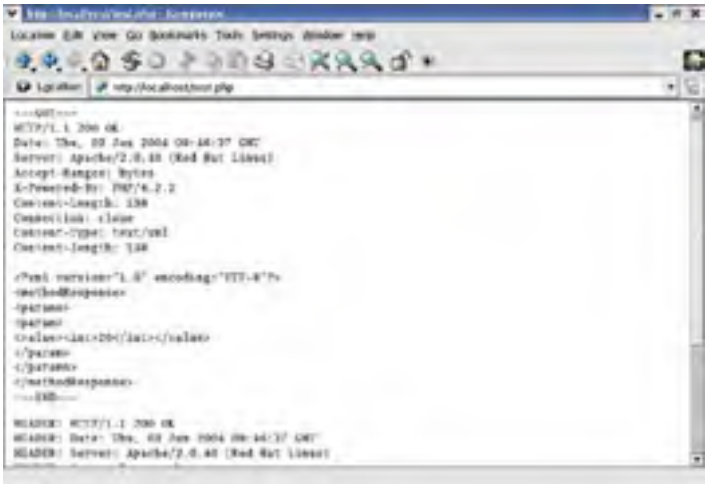


Figure 6-6: The Server Response of Multiplying Two Integers

If the client cannot successfully communicate with the server, a fault condition occurs in the server response.

Figure 6-7 shows the server response to an unsuccessful client request to multiply two integers:

Figure 6-7: Unsuccessful Server Response of Multiplying Two Integers

Similarly, you can obtain the server response for the client request to sort employee names on salary basis.

Figure 6-8 shows the server response with employee names sorted on the basis of their salaries:

Figure 6-8: The Server Response to Sort Employees

When an XML-RPC client makes a request to the server, the server response can be in one of the following forms:

- Low-level I/O error
- Valid XML-RPC response
- XML-RPC error condition

PHP has certain fault codes that XML-RPC classes use to indicate the type of fault. If an XML-RPC error condition occurs, the XML-RPC classes display the corresponding fault code and its description.

Table 6-1 describes the various fault codes:

Table 6-1: Fault Codes in PHP

Fault code	Fault description	Cause of error
1	Unknown method	Occurs when the client requests for a method that is not defined in the server.
2	Invalid return payload	Occurs when the client receives an invalid XML-RPC response from the server.
3	Incorrect parameters passed to method	Occurs when the client passes either incorrect number of parameters or incorrect data type of the parameter to the remote function.
4	Didn't receive 200 OK from the server	Occurs when there is an HTTP transport error.

XML-RPC provides interoperability between different platforms and languages. You can implement XML-RPC in PHP to write client and server scripts. You use the extensions of the XML-RPC protocol to authenticate the end users using the server scripts and provide an asynchronous transmission system.

Chapter 7: PHP and Web Distributed Data Exchange

Download CD Content

Web Distributed Data eXchange (WDDX) is an eXtensible Markup Language (XML)-based technology that helps exchange of data between Web programming languages, such as Hypertext Pre-Processor (PHP), Java, Active Server Pages (ASP), and Perl. WDDX provides a standard format for creating XML-based data structures, which can be easily transmitted across the Internet.

The WDDX module in PHP contains certain functions that convert PHP data to WDDX packets and WDDX packets to PHP data. WDDX functions use WDDX data structures to support information transfer between Web applications. These data structures are platform-independent.

The WDDX Application Programming Interface (API) in PHP supports encoding and decoding functions. Encoding functions convert PHP data to WDDX packets. Decoding functions convert WDDX packets to PHP data.

This chapter describes how to encode and decode data using the WDDX module in PHP. It also explains the data types that WDDX supports.

Introducing WDDX

Using WDDX, you can represent data corresponding to a specific programming language in a language independent format using a process called serialization. Serialization is a process that converts language-specific data structure into a WDDX packet. Deserialization converts a WDDX packet to language-specific data structure. The application that accepts data in the form of WDDX packets uses the deserialization process to retrieve data in its original form.

Understanding WDDX

WDDX API for programming languages provides functions that automatically encode data structures specific to a language into WDDX packets. The WDDX API also provides functions to decode the WDDX packets into language-specific data structures. WDDX packets store data in the XML format. A WDDX packet is created when a Web application converts language-specific data structure into WDDX.

The two important applications of WDDX are server-to-browser data exchange and server-to-server data exchange. In the server-to-browser data exchange, you can retrieve data from an application server, convert data into the WDDX packets using the WDDX API, and transfer the data to the browser in the form of an HyperText Transfer protocol (HTTP) response. In the server-to-server data exchange, the two servers that perform data exchange should have the WDDX API, to map data in a WDDX packet to the corresponding language-specific data structure.

The WDDX API consists of two parts, the WDDX Document Type Definition (DTD) and a set of serialization/deserialization modules. WDDX DTD is a specification that defines WDDX structures. The WDDX serialization/deserialization modules handle translation of data structures specific to a language to platform-independent XML representation or WDDX packets. You can develop WDDX serialization/deserialization modules in programming languages, such as PHP, Perl, ASP, Java, Python, and JavaScript.

WDDX DTD validates a WDDX packet that consists of data stored in the XML format. In the XML format, you enclose the data within the opening and closing tags, similar to HTML. WDDX supports data types similar to the data types in Web programming languages, such as Cold Fusion Markup Language (CFML) and PHP. Data types that WDDX supports are:

- Numbers: Represent floating-point values. Numbers range from +/-1.7E to +/-308.
- Date-time values: Represent date and time values that are encoded according to the International Standardization Organization 8601 (ISO8601) standard. You need not prefix 0 for the single digit values of month, day, hour, minute, or second.
- Strings: Contain arbitrary number of characters within it. A string data type should not contain null values. You can encode a string value using double-byte characters.
- Binary: Represents strings of binary data. The binary data is encoded in Multipurpose Internet Mail Extensions (MIME) base64 format.
- Array: Represents a collection of objects. It is an integer index data type.
- Structure: Represents a collection of objects of arbitrary data type that are indexed using strings.
- Recordset: Represents encapsulated data in a tabular form. A recordset is a collection of rows, in which each row consists of a set of named fields or columns. You can store only simple data types in a recordset. For storing complex data types in the tabular form, you should use an array of structures. The field names in a table should be in the [_.0-9A-Za-z]+ form, where the character, ., represents a period.

Note WDDX supports the null data type. You cannot associate null values with a number or a string.

Installing WDDX

You need to install the expat library available with Apache 1.3.7 or its higher version to use WDDX. The expat library is a C code library that implements a Simple Application Programming Interface for XML (SAX) parser. To install the expat library:

1. Download the expat package from the following URL:

   http://sourceforge.net/projects/expat/.

2. Unzip the package using the following command:
   ```
   tar -xvfz expat-1.95.5.tar.gz
   ```
3. Type the following command to change your working directory to expat-1.95.5:
   ```
   pushd expat-1.95.5
   ```

After you install the expat library, recompile PHP by adding -enable-wddx option to the configuration script.

Note The windows version of PHP has built-in support for the WDDX functions.

To check the installation of WDDX, run any script that uses WDDX functions provided in PHP. The following code uses a WDDX function:

```
<?php
print wddx_serialize_value("WDDX example");
?>
```

The above code produces a WDDX packet that contains the string, WDDX example, in the header tags. This WDDX packet ensures that the WDDX functions are already installed on your computer. If the code does not produce any output, install WDDX on your computer.

Exchanging Data using WDDX

WDDX is based on XML 1.0 version, and uses protocols, such as HTTP, to transfer data between Web applications. You can encode and decode data in the form of WDDX packets using serialization/deserialization functions.

Figure 7-1 shows the process of encoding data to the WDDX format and decoding data from the WDDX format:

Figure 7-1: Data Exchange Using WDDX Packet

The serialization process takes a part of data and converts it into a WDDX packet. Data in the WDDX packet is either a simple data type or complex data type. The simple data types include string, numeric, and Boolean data. The complex data types are arrays, structures, and recordsets. The deserialization process converts data in a WDDX packet to its original data type.

The syntax for the structure that holds data contained in a WDDX packet is:

```
<wddxPacket version='1.0'>
<header></header>
<data>
<data type>n</data type>
</data>
</wddxPacket>
```

In the above syntax, the <data /> tag pair denotes the start and end of serialized data. The <header /> tag contains a comment. The value n enclosed in the <data type /> tag represents a value that corresponds to the specified data type.

You need to enclose the WDDX packet in the <wddxPacket /> tag pair. The <wddxPacket> tag need to include the current version of WDDX. The data types that you can serialize into a WDDX format are: <number>, <Boolean>, <array>, <struct>, and <binary>. The following code shows the WDDX packet that contains a serialized string value:

```
<wddxPacket version='1.0'>
<header></header>
<data>
<string>This is a WDDX packet</string>
</data>
</wddxPacket>
```

The above WDDX packet contains a string value, and the application stores the value inside the tags as a string. If an application needs to deserialize the WDDX packet, it checks the <data type> tag pair to uncover the type of data stored in the packet.

Note In a WDDX packet, the application ignores the space between any pair of tags.

Using WDDX with PHP

PHP consists of WDDX modules that contain serialization and deserialization functions to convert PHP variables into WDDX compatible data structures, and then reconvert them into the PHP format. PHP generates WDDX packets as a single string. For example, you can encode a PHP associative array in WDDX and transfer the array to a Perl script. The Perl script decodes the data in the WDDX packet and uses the data for further processing.

The WDDX API of PHP provides five functions to encode PHP to WDDX packet. These encoding functions are wddx_serialize_value(), wddx_serialize_vars(), wddx_add_vars(), wddx_packet_start(), and wddx_packet_end(). The WDDX API for PHP also provides a function to decode data from a WDDX packet.

Encoding Data using WDDX

The WDDX technology enables data exchange between the Web applications that are created in different programming languages and run on different platforms. The PHP functions that encode data into a WDDX packet are:

- `wddx_serialize_value()`: Encodes a single variable into a WDDX packet.
- `wddx_serialize_vars()`: Encodes more than one variable into a WDDX packet.
- `wddx_add_vars()`: Adds PHP variables to a WDDX packet.
- `wddx_packet_start()`: Creates a new WDDX packet.
- `wddx_packet_end()`: Closes a WDDX packet.

Using the wddx_serialize_value() Function

The wddx_serialize_value() function creates a WDDX packet that contains a single value. The syntax of the wddx_serialize_value() function is:

```
string wddx_serialize_value (mixed var [, string comment])
```

In the above syntax:

- var denotes the name of the variable that you need to add to a WDDX packet.
- The wddx_serialize_value() function either returns a string that contains a WDDX packet, or the value, FALSE, if an error occurs. The wddx_serialize_value() function accepts variables as its first argument and converts the variables to a WDDX packet. The second argument of the wddx_serialize_value() function is optional and adds a comment to a WDDX packet.

Listing 7-1 shows how to use the wddx_serialize_value() function:

Listing 7-1: Serializing a Variable Using the wddx_serialize_value() Function

```
<?php
$country="Australia";
$va=wddx_serialize_value($country);
$fp=fopen("/var/www/html/example1.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to encode the $country variable to a WDDX packet using the wddx_serialize_value() function. The Web browser generates a parsed output and does not display the WDDX tags that are generated during the serialization process. The fwrite() function in the above listing writes the contents of the WDDX packet to the example.xml file.

Run the XML file to display the contents of the WDDX packet.

Figure 7-2 shows how the wddx_serialize_value() function generates a WDDX packet:

Figure 7-2: Using the wddx_serialize_value() Function to Generate a WDDX Packet

You can use the second parameter of the wddx_serialize_value() function to add a comment to the header of the WDDX packet, as shown in Listing 7-2:

Listing 7-2: Adding a Comment to a the WDDX Packet

```
<?php
$country="Australia";
$va=wddx_serialize_value($country, "An example of WDDX");
$fp=fopen("/var/www/html/example2.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows the WDDX packet that contains the comment, An example of WDDX.

Figure 7-3 shows a WDDX packet that contains a comment:

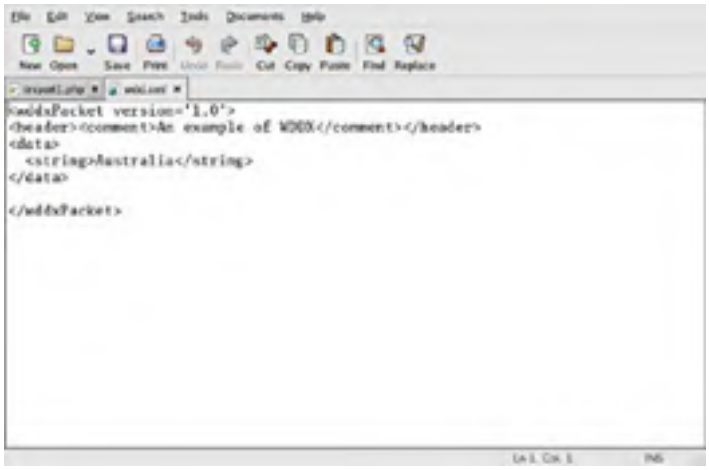


Figure 7-3: WDDX Packet Containing a Comment

You can serialize an array of values using the wddx_serialize_value() function, as shown in Listing 7-3:

Listing 7-3: Serializing an Array Using the wddx_serialize_value() Function

```
<?php
$countries=array("Australia", "India", "Kenya");
$va=wddx_serialize_value($countries);
$fp=fopen("/var/www/html/example3.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to serialize three variables, India, Kenya, and Australia, using the wddx_serialize_value() function.

Figure 7-4 shows the WDDX packet representing an array of values:

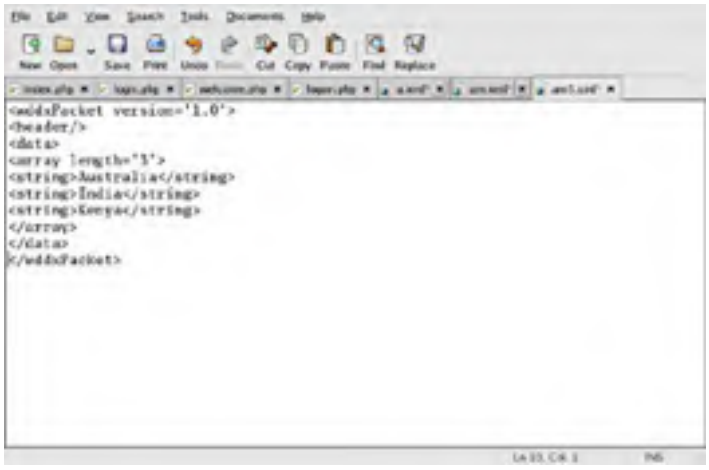


Figure 7-4: WDDX Packet Representing an Array of Values

Using the wddx_serialize_vars() Function

The wddx_serialize_vars() function creates a WDDX packet with a serialized representation of the passed variables. The syntax of the wddx_serialize_vars() function is:

```
string wddx_serialize_vars (mixed var [, mixed..])
```

The wddx_serialize_vars() function accepts one or more variables as its arguments, which can either be strings or arrays containing strings.

Listing 7-4 shows how to serialize a single variable using the wddx_serialize_vars() function:

Listing 7-4: Serializing a Variable Using the wddx_serialize_vars() Function

```
<?php
$country="Australia";
$va=wddx_serialize_vars("country");
$fp=fopen("/var/www/html/example4.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing uses the wddx_serialize_vars() function to serialize a single variable into a WDDX packet.

Figure 7-5 shows the WDDX packet that serializes a variable using the wddx_serialize_vars() function:

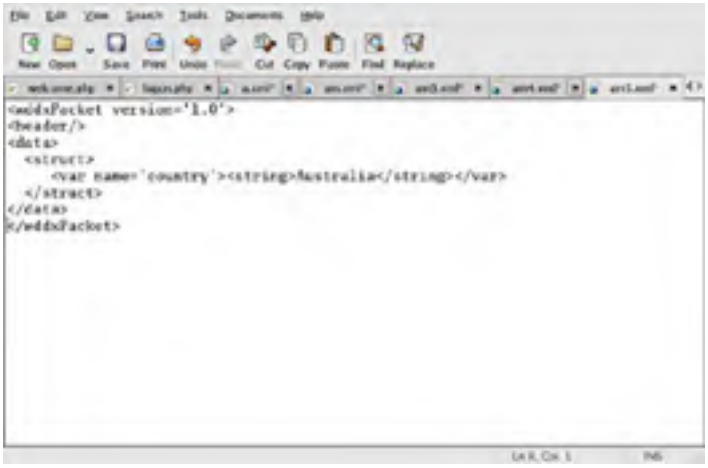


Figure 7-5: WDDX Packet Representing a Single Encoded Variable

Note The serialization of a single variable using the wddx_serialize_vars() function generates a WDDX packet, which is different from the WDDX packet that the wddx_serialize_value() function generates.

Listing 7-5 shows how to use the wddx_serialize_vars() function:

Listing 7-5: Serializing Multiple Values Using the wddx_serialize_vars() Function

```
<?php
$x="This is an example of the wddx_serialize_vars function.";
$y=array("emp_name"=>"Kenny", "emp_id"=>"E01", "emp_dept"=>"Sales");
$va=wddx_serialize_vars("x", "y");
$fp=fopen("/var/www/html/example5.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to serialize two variables, x and y, using the wddx_serialize_vars() function.

Figure 7-6 shows the WDDX packet that serializes the two variables, x and y:

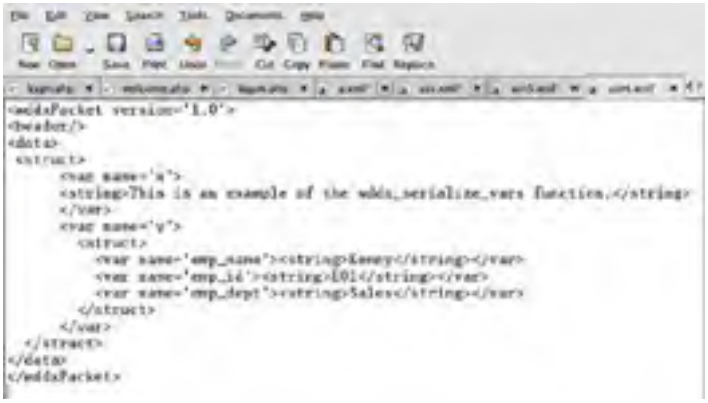


Figure 7-6: WDDX Packet that Serializes Two Variables

You can serialize more than two variables using the wddx_serialize_vars() function, with each variable containing an array of values, as shown in Listing 7-6:

Listing 7-6: Using the wddx_serialize_vars() Function

```
<?php
$student_names=array("Peter", "Crisp", "Ron");
$colleges=array("AEC", "BMBS");
$branches=array("IT", "EC", "EE");
$va=wddx_serialize_vars("student_names", "colleges", "branches");
$fp=fopen("/var/www/html/example6.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to use the wddx_serialize_vars() function to serialize three variables, $student_names, $branches, and $colleges, each containing an array of values.

Figure 7-7 shows the WDDX packet that serializes three variables: $student_names, $branches, and $colleges:

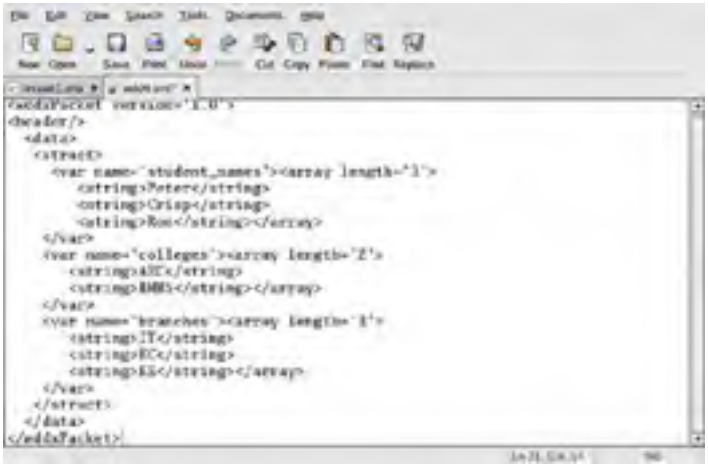


Figure 7-7: WDDX Packet that Uses the wddx_serialize_vars() Function

Using the wddx_add_vars() Function

The wddx_add_vars() function adds one or more variables to a WDDX packet, which is represented by a packet ID. The syntax for the wddx_add_vars() function is:

```
boolean wddx_add_vars(int packet_id, mixed var1 [,mixed var2,...])
```

The wddx_add_vars() function returns either TRUE or FALSE. You should use two additional functions to generate a WDDX packet that uses the wddx_add_vars() function, which are wddx_packet_start() and wddx_packet_end().

The wddx_packet_start() function creates a new WDDX packet with the WDDX structure inside the packet. The syntax of the wddx_packet_start() function is:

```
int wddx_packet_start([string comment])
```

In the above syntax, the int keyword indicates that the return type of the wddx_packet_start() function is an integer.

The wddx_packet_start() function accepts an optional comment string as an argument and returns a packet ID. This function automatically creates a structure definition for including variables inside a WDDX packet.

The wddx_packet_end() function closes a WDDX packet specified by the packet ID and returns a string that contains the WDDX packet. The syntax of the wddx_packet_end() function is:

```
string wddx_packet_end(int packet_id)
```

In the above syntax, the string keyword indicates that the return type of the wddx_packet_end() function is a string. The argument that you pass to the wddx_packet_end() function identifies the WDDX packet that your application should decode.

To create a WDDX packet and add variables to the packet using the wddx_add_vars() function:

1. Create an empty WDDX packet that can hold the data. You can create a WDDX packet using the wddx_packet_start() function.
2. Add data to the WDDX packet. You use the wddx_add_vars() function to add each variable to the WDDX packet.
3. Close the WDDX packet using the wddx_packet_end() function.

The first argument in the wddx_add_vars() function specifies the WDDX packet ID to which you need to add data.

Listing 7-7 shows how to use the wddx_add_vars() function to create a WDDX packet:

Listing 7-7: Creating a WDDX Packet Using the wddx_add_vars() Function

```
<?php
$country="Australia";
$id=wddx_packet_start("PHP");
wddx_add_vars($id, "country");
// contents of the packet are transferred to example7.xml file.
$packet=wddx_packet_end($id);
$fp=fopen("/var/www/html/example7.xml", "w+");
fwrite($fp, $packet, strlen($packet));
?>
```

The above listing shows how to create a WDDX packet using the wddx_packet_start() function. You can add variables to a WDDX packet using the wddx_add_vars() function.

Figure 7-8 shows the variables that you add to a WDDX packet using the wddx_add_vars() function:

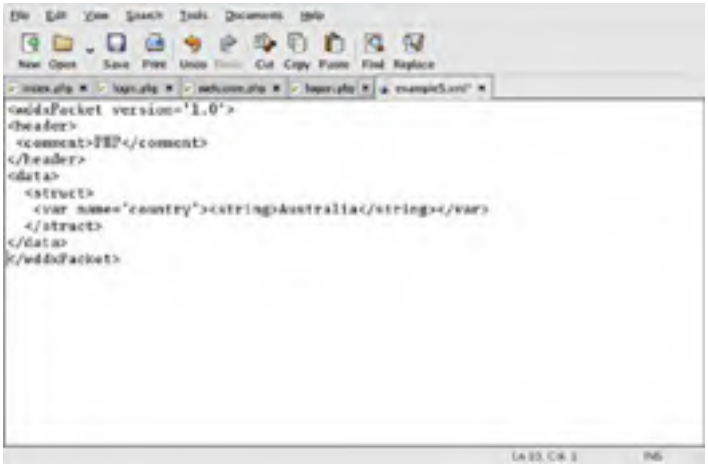


Figure 7-8: Adding Variables to a WDDX Packet Using the wddx_add_vars() Function

You can add more than one variable using the wddx_add_vars() function, as shown in Listing 7-8:

Listing 7-8: Using the wddx_add_vars() Function to Add More than One Variable

```
<?php
$name="Ronald";
$department="Administration";
$Age="35";
$id=wddx_packet_start("PHP example");
wddx_add_vars($id, "name", "department", "Age");
$packet=wddx_packet_end($id);
$fp=fopen("/var/www/html/example8.xml", "w+");
fwrite($fp, $packet, strlen($packet));
?>
```

The above listing shows a WDDX packet that serializes three variables, name, department, and age.

The wddx_serialize_vars() and wddx_add_vars() functions use the <struct> element to store variables and their values. The wddx_serialize_value() function uses the <struct> element to store variables that represent associative arrays.

Figure 7-9 shows the WDDX packet that serializes three variables:

Figure 7-9: Adding Multiple Variables Using the wddx_add_vars() Function

Note The wddx_serialize_value() and wddx_serialize_vars() functions automatically create a WDDX packet, but you should provide a starting and a closing header for the wddx_add_vars() function.

Decoding Data using WDDX

Decoding involves parsing a WDDX packet into an appropriate PHP variable. PHP contains a single function, wddx_deserialize(), to retrieve data from a WDDX packet. The wddx_deserialize() function decodes a WDDX packet. The syntax of the wddx_deserialize() function is:

```
mixed wddx_deserialize(string packet);
```

The above syntax shows that the wddx_deserialize() function accepts a single string argument that represents a WDDX packet. The wddx_deserialize() function returns a number, a string, or an array.

Listing 7-9 shows the deserialization of a PHP variable from a WDDX packet:

Listing 7-9: Deserializing a Single PHP Variable

```
<?php
$country="Australia";
//Serialize the variable using the wddx_serialize_value() function.
$id=wddx_serialize_value($country);
//Write the contents of the WDDX packet into an XML file.
$fp=fopen("/var/www/html/example9.xml", "w+");
fwrite($fp, $id, strlen($id));
//Deserialize the WDDX packet.
$output=wddx_deserialize($id);
//Display the decoded value of the PHP variable.
print($output);
?>
```

The above listing shows how to retrieve data from a WDDX packet using the wddx_deserialize() function.

Figure 7-10 shows deserialized data of the WDDX packet in Listing 7-9:

Figure 7-10: Output of the wddx_deserialize() Function

The wddx_deserialize() function decodes a variable containing an array of values, as shown in Listing 7-10:

Listing 7-10: Deserializing an Array of Values

```
<?php
$names=array("John", "Christine", "Joseph");
//Serializing an array of values.
$id=wddx_serialize_value($names);
//Deserializing the WDDX packet identified by $id.
$output=wddx_deserialize($id);
print_r($output);
?>
```

The above listing shows how to deserialize a WDDX packet that contains an array of values.

Figure 7-11 shows the output of Listing 7-10:

Figure 7-11: Output of a WDDX Packet

◄ PREVIOUS NEXT ►

Chapter 8: Working with Databases using PHP and XML

Download CD Content

You can store data in databases or as eXtensible Markup Language (XML) documents. Hypertext Preprocessor (PHP) converts data stored in the databases into XML documents and vice versa. Applications on the Internet can retrieve and update the data stored in the form of XML documents.

This chapter explains how to store XML documents in a database using PHP. This chapter describes how to export data stored in databases into an XML document and how to import data from an XML document into a database using the Simple API for XML (SAX) approach. This chapter also describes how to convert an XML document into an HTML format.

Storing XML Documents

XML is a standard language to store information on the Internet in a structured manner. XML is a subset of Standard Generalized Markup Language (SGML) and consists of tags to represent data.

An XML document stores data in a tree structure. You can query the contents of the XML document using a semi-structured language, such as XSL Transformations (XSLT). You can use XML documents to create Web applications and exchange information on the Internet.

You can store XML documents in a database and retrieve them. You can store the XML documents in a database by mapping the tree structure of the XML document to the database schemas.

When you map the tree structure of the XML document to the database schema, you design the database schemas according to the Document Type Declaration (DTD) of the XML structure. DTD defines the rules that an XML document should follow. The advantages of using database schemas to store XML documents are:

- You can create XML documents using relational tables, without depending on the DTD.
- You can perform various query operations on the XML documents using index structures. Indexing is of two types, position-based and path-based. Position-based indexing manipulates the elements and attributes to perform a query operations based on their position within the document. Path-based indexing performs the query operation using the path of the elements and attributes in the tree structure.

The disadvantages of using database schemas to store XML documents are:

- The database schemas cannot be designed efficiently in accordance with DTD. Relational databases and object-oriented databases cannot represent DTD to store XML documents efficiently.
- Maintaining the consistency of the database schemas is not possible. You need to change the structure of the database schemas every time there is a change in the structure of the XML document.

Storing XML Documents in Databases

An XML document is represented in a tree structure, which contains various nodes, such as element node, attribute node, and text node. The element node contains a text label, and can contain child nodes, such as the attribute node and the text node. The attribute node contains the name and value of the attribute. An attribute node does not have any child nodes. The text node in an XML document contains character data and represents the value for the element.

The process of storing the XML documents involves decomposing the XML tree structure into relations, which the relational tables can reuse and access.

The database relations that you can use for storing the XML document include element, attribute, text, and path. A relation stores data corresponding to the node of the tree structure. For example, the attribute relation stores the data stored in the attribute node. The various relations and their database attributes are:

- Element Relation: Stores data about the element node using database attributes, such as docID, pathID, index, and pos. The docID represents the document identifier, and pathID represents the path identifier. The index attribute represents the order among the sibling having the same path identifier. The pos attribute specifies the position of the element node with respect to the other nodes.
- Attribute Relation: Stores data about the attribute nodes. The database attributes in the attribute relation are docID, pathID, Attvalue, and pos. The Attvalue attribute specifies the value of an attribute.
- Text Relation: Stores data about the text nodes. The database attributes in the text relation are docID, pathID, value, and pos.
- Path Relation: Stores data about simple paths. The database attributes in the path relation are docID, pathID, Attvalue, and pos.

You can use query languages such as XML Query Language (XQL) to perform data manipulation, such as joins, on the XML data.

Organizing XML Documents with Entities

An XML document consists of storage units called entities, which are a collection of text and markup tags. In an XML document, entities are used to store data. The advantages of organizing an XML document with entities are:

- Ensures consistency in representing repeating text.

- Ensures that every entity instance is exactly the same.
- Limits repetitive typing of same information.

PREVIOUS NEXT

Exporting Database Records to Create XML Documents

Using PHP, you can export and convert the data stored in the database into an XML document. You can also manipulate the data stored in databases using various tools, such as XPath and XSLT.

To export the data from a database in an XML document:

1. Establish a connection with the database.
2. Retrieve data from the database by executing SQL queries.
3. Create an XML document from the retrieved information in the memory of the system.

You can manipulate and parse the generated XML document using Document Object Model (DOM).

Connecting to a Database to Export Data

The first step in exporting data from a database into an XML document is to establish a connection with the MySQL database. PHP provides a specific set of APIs that perform various query operations on the MySQL database. The PHP functions that you use to work with MySQL database server are:

- `mysql_connect`: Connects to the MySQL database. The syntax of the mysql_connect() function is:

```
resource mysql_connect ( [string Host_Name [:port][:/path/to/socket]], string username
[, string password ]]])
```

In the above syntax, you need to pass the hostname, username, and password parameters as arguments to the mysql_connect() function. You can also specify optional parameters, such as the port and path of the MySQL socket.

- `mysql_select_db`: Selects a MySQL database from the specified link identifier. The syntax of the mysql_select_db() function is:

```
bool mysql_select_db(string database, resource[link indentifier])
```

In the above syntax, the mysql_select_db() function returns the value, True, on establishing a connection with the database; and returns the value, False, when a connection is not established with the database.

- `mysql_query`: Performs query operation on the currently selected database. The syntax of the mysql_query() function is :

```
resource mysql_query ( string query [, resource link identifier])
```

In the above syntax, if no resource link identifier is specified, the query is generated from the currently opened linked.

- `mysql_fetch_object`: Returns the result set of the query as an object. The syntax of the mysql_fetch_object() function is:

```
object mysql_fetch_object(resource result, int [result _type])
```

- `mysql_close`: Closes the connection to the database. The mysql_close() function returns a Boolean value. The syntax of the mysql_close() function is:

```
bool mysql_close(resource[link identifier])
```

- `mysql_num_rows`: Returns the number of rows in a result set. The syntax of the mysql_num_rows() function is:

```
int mysql_num_rows(resource result)
```

- `mysql_error`: Returns the numerical value of the error message generated after running the MySQL statement. The syntax of the mySQL_error() function is:

```
string msql_error(resource [link identifier ])
```

- `mysql_db_query`: Selects a database and runs a query on the database. The syntax of the mysql_db_query() function is:

```
resource mysql_db_query(string database, string query, resource[link identifier])
```

Listing 8-1 shows how to establish a connection with the database using the database access functions provided by PHP:

Listing 8-1: Creating a Connection to the Database

```
$connect = mysql_connect("localhost", "root", "root123") or die ("could not connect");
print("Successful Connection");
mysql_select_db("information") or die ("Unable to select database!");
$query = "SELECT * FROM employee";
$result = mysql_query($query)
or die ("query failed, error code = : "  . mysql_errno());
```

In the above listing:

- The mysql_connect() function creates a connection to the MySQL database and returns value, True, when the connection is established.
- You need to provide the hostname, username, and the password as arguments to the mysql_connect() function.

After connecting to a database, you can perform various operations such as create a table, and insert, update, and delete rows from the table.

Listing 8-2 shows how to create a table and insert rows in the MySQL database, information:

Listing 8-2: Creating the Employee table in MySQL

```
mysql> USE information -A;
mysql> CREATE TABLE employee
-> (
-> EmployeeName Varchar(50) NOT NULL,
-> EmpID Varchar(5) NOT NULL,
-> Salary Integer
-> );
```

The above listing creates the Employee table. After creating the Employee table, you can insert rows in the Employee table.

Listing 8-3 shows how to insert rows in the Employee table:

Listing 8-3: Inserting Rows in the Employee Table

```
mysql> INSERT INTO employee
VALUES("Joseph", "1005", 20000);
mysql> INSERT into employee
-> VALUES("John", "1007",25000);
mysql> INSERT into employee
VALUES("Harry", "1008", 35000);
```

The above listing inserts three rows in the Employee table. You can retrieve data from the database using the SELECT command.

Figure 8-1 shows the output of the SELECT command:

Figure 8-1: Contents of the Employee Table

You can export the content of the Employee table into an XML tree using the DOM parser. To do this, you need to first establish a connection with the database, and then retrieve data using the MySQL functions provided by PHP.

Creating an XML Document

After establishing the connection with a database, you need to create an XML document from the database. You can create an XML document from the data retrieved after querying the database, after implementing the DOM functions, such as new_xmldoc().

Listing 8-4 shows how to create an XML document from the data retrieved from the MySQL database:

Listing 8-4: Creating the XML Document

```
<?php
$connection = mysql_connect("localhost", "root", "root123") or die ("Unable to connect!");
mysql_select_db("information") or die ("Unable to select database!");
$query = "SELECT * FROM employee";
$result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
if (mysql_num_rows($result) > 0)
{
   // Create the DomDocument object.
   $doc = new_xmldoc("1.0");
   // Add root node.
   $root = $doc->add_root("EmployeeInfo");
   // Iterate through result set.
   while(list($EmployeeName, $EmpID, $Salary) = mysql_fetch_row($result))
   {
      // Create item node.
      $rec = $root->new_child("employee", "");
      $rec->set_attribute("EmpID", $EmpID);
      $rec->new_child("EmployeeName", $EmployeeName);
      $rec->new_child("Salary", $Salary);
```

```
    }
    // Print the tree.
    echo $doc->dumpmem();
}
?>
```

In the above listing:

- An XML document is created from the data obtained after querying the Employee table stored in the information database.
- The new_xmldoc() function creates a new XML document.
- The new_child() function allows you to add child nodes, such as EmpName and Salary, to the EmployeeInfo root node.
- The dumpmem() function converts the generated XML document into a string that can be printed using the echo command.

Figure 8-2 shows the output of Listing 8-4:

Figure 8-2: Contents of XML File

You can also export data from multiple tables simultaneously. In PHP, you can create an XML document from the data retrieved after performing the query operation on multiple tables. For example, you want to display the Employee designation and the department along with other personnel information stored in the Employee table. The Employee information, such as designation and department, are stored in the Employee_Info table in the information database.

Figure 8-3 shows the contents of the Employee_Info table created in the information database:

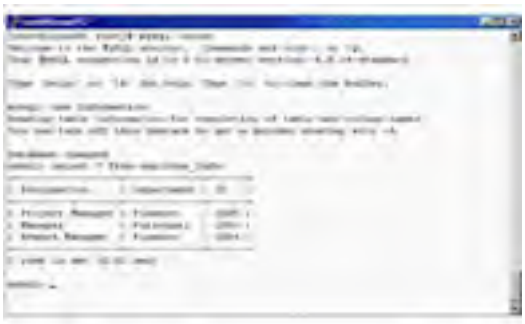


Figure 8-3: Contents of Employee_Info Table

Listing 8-5 shows how to export the data retrieved from the database after querying the Employee table and Employee_Info table:

Listing 8-5: Exporting Data Retrieved from Multiple Tables

```
<?php
$connection = mysql_connect("localhost","root","root123") or die ("Unable to connect!");
mysql_select_db("information") or die ("Unable to select database!");
$query = "SELECT * FROM employee";
$result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
if (mysql_num_rows($result) > 0)
{
   // Create the DomDocument object.
    $doc = new_xmldoc("1.0");
   // add root node
   root = $doc->add_root("EmployeeInfo");
   // Iterate through result set.
   while(list($EmployeeName, $EmpID, $Salary) = mysql_fetch_row($result))
   {
      // Create item node.
      $rec = $root->new_child("employee","");
      $rec->set_attribute("EmpID", $EmpID);
```

```
        $rec->new_child("EmployeeName", $EmployeeName);
        $rec->new_child("Salary", $Salary);
        // Add node from table Employee_Info.
         $query2 = "SELECT Designation,Department FROM Employee_Info where ID = '$EmpID'";
         $result2 = mysql_query($query2) or die ("Error in query: $query2. " . mysql_error());
        // Print each track as a child node of <tracks>.
        while(list($Designation,$Department) = mysql_fetch_row($result2))
        {
           $rec->new_child("Designation","$Designation");
           $rec->new_child("Department","$Department");
        }
    }
    // Dump XML document to a string.
}
$varf = $doc->dumpmem();
echo $varf;
$fp = fopen("/var/www/html/out.xml", "w+");
fwrite($fp, $varf, strlen($varf));
?>
```

The above listing shows how to export data retrieved from the database after querying the Employee and Employee_Info tables. In the above listing:

- The new_xmldoc() function generates an XML document.
- Data from the Employee and Employee_Info tables are inserted in the xml file created.
- The nodes, such as designation, department, and employee are added to the root node after querying the Employee_Info table.
- The dumpmem() function converts the XML document into a string, $varf.
- The generated XML file is stored as a separate file using the fwrite() function.

Figure 8-4 shows the output of Listing 8-5:

Figure 8-4: Output of Exporting Data from Multiple Tables

Closing the Connection to a Database After Exporting Data

You need to close the MySQL connection after you have generated the result set from the MySQL database. The mysql_close() function lets you close the MySQL connection.

Listing 8-6 shows how to use the mysql_close() function to close database connection:

Listing 8-6: Closing Connection to MySQL Database

```
<?php
$connection = mysql_connect("localhost", "root", "root123")
or exit ("Could not connect");
print ("Connected successfully");
//Close connection.
mysql_close($connection);
?>
```

In the above listing:

- The mysql_close() function closes the connection to the MySQL database after generating the result.
- The link identifier is specified as the argument to the mysql_close() function.
- The mysql_close() function closes the last active link to the MySQL server by default.

Formatting XML Document into HTML Format

In addition to exporting data into an XML document, you can transform the data from the database into HTML format. To format a generated XML document into HTML format using the SAX parser:

1. Generate the XML document after retrieving the records from the database.
2. Transform the generated XML document into HTML pages.

You can transform an XML document into HTML format using the SAX parser. SAX is an event-based approach that parses an XML document in chunks, and processes the tags as the parser encounters them in the document. You can export the content of the Employee table into an XML file and then convert the XML document into HTML format.

Listing 8-7 shows how to generate the XML document from the database and convert the XML document into HTML:

Listing 8-7: Generating XML and Converting into HTML

```
<?php
// Defining the database parameters.
$hostname = "localhost";
$username = "root";
$password = "root123";
$database = "information";
$table = "employee";
//  Exporting the database records and generating a XML document.
// Querying the database.
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($database) or die ("Unable to select database!");
$query = "SELECT * FROM $table";
$res = mysql_query($query) or die ("Error in query: $query. " . mysql_error());
if(mysql_num_rows($res) > 0)
{
   // Creating the DomDocument object.
   doc = new_xmldoc("1.0");
   // Adding the root node.
   $root = $doc->add_root("table");
   $root->set_attribute("name", $table);
   // Creating nodes.
   $struct = $root->new_child("struct", "");
   $data = $root->new_child("data", "");
   // Create elements for each field name, type and length.
   $fields = mysql_list_fields($database, $table, $connection);
   for ($t=0; $t<mysql_num_fields($fields); $t++)
   {
      $field = $struct->new_child("field", "");
      $name = mysql_field_name($fields, $t);
      $length = mysql_field_len($fields, $t);
      $type = mysql_field_type($fields, $t);
      $field->new_child("name", $name);
      $field->new_child("type", $type);
      $field->new_child("length", $length);
   }
   // Move on to getting the raw data (records).
   // Iterate through result set.
   while($row = mysql_fetch_row($res))
   {
      $record = $data->new_child("record", "");
      foreach ($row as $field)
      {
         $record->new_child("item", $field);
      }
   }
   // Dumping the XML tree as a string.
   $xml_string = $doc->dumpmem();
}
// Closing the database connection.
mysql_close($connection);
// Converting the XML document into an HTML page using SAX parser.
// Array to hold HTML markup for starting tags.
$startTags = array(
'TABLE' => '<html><head></head><body><table border="1"
cellspacing="0"cellpadding="5">',
'STRUCTURE' => '<tr>',
'FIELD' => '<td bgcolor="silver"><font face="Times Roman" size="-1">',
'RECORD' => '<tr>',
'ITEM' => '<td><font face="Arial" size="-1">',
'NAME' => '<b>',
'TYPE' => '  <i>(',
'LENGTH' => ', '
);
// Array to hold HTML markup for ending tags.
$endTags = array(
'TABLE' => '</body></html></table>',
'STRUCTURE' => '</tr>',
'FIELD' => '</font></td>',
'RECORD' => '</tr>',
'ITEM' => ' </font></td>',
'NAME' => '</b>',
'TYPE' => '',
```

```
'LENGTH' => ')</i>'
);
// Call this when a start tag is found.
function startElementHandler($parser, $name, $attributes)
{
   global $startTags;
   if($startTags [$name])
   {
      // Look up array for this tag and print corresponding markup.
      echo $startTags[$name];
   }
}
// Call this when an end tag is found.
function endElementHandler($parser, $name)
 {
   global $endTags;
   if($endTags [$name])
   {
      // Look up array for this tag and print corresponding markup.
      echo $endTags [$name];
   }
}
// Call this when character data is found.
function characterDataHandler($parser, $data)
{
   echo $data;
}
// Initialize the parser.
$xml_parser = xml_parser_create();
// Set callback functions.
xml_set_element_handler($xml_parser, "startElementHandler", "endElementHandler");
xml_set_character_data_handler($xml_parser, "characterDataHandler");
// Parse the XML document.
if (!xml_parse($xml_parser, $xml_string, 4096))
{
   $ec = xml_get_error_code($xml_parser);
   die("XML parser error (error code " . $ec . "): " . xml_error_string($ec) .
   "<br>Error occurred at line " . xml_get_current_line_number($xml_parser) . ", column "
   .
   xml_get_current_column_number($xml_parser) . ", byte offset " .
   xml_get_current_byte_index($xml_parser));
}
xml_parser_free($xml_parser);
?>
```

The above listing converts the content of the Employee table into an XML document using the DOM approach. The generated XML document is transformed into HTML format using the SAX parser.

You need to define various database parameters, such as hostname, username, and password, to establish a connection to the database. After establishing a connection with the database, you need to process queries to retrieve data from the database.

In the Listing 8-7:

- The XML tree is created using the new_xmldoc() function, which belongs to the DOMDocument class.
- The nodes are added to the XML tree after being retrieved from the result set.
- The dumpmem() function lets you dump the generated XML tree in a string. You can also store the XML tree in a file. For example, in Listing 8-7, you use the following code to store the XML document in the export.xml file:

```
$fp = fopen("/var/www/html/export.xml", "w+");
fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
```

Figure 8.5 shows the contents of the export.xml file:

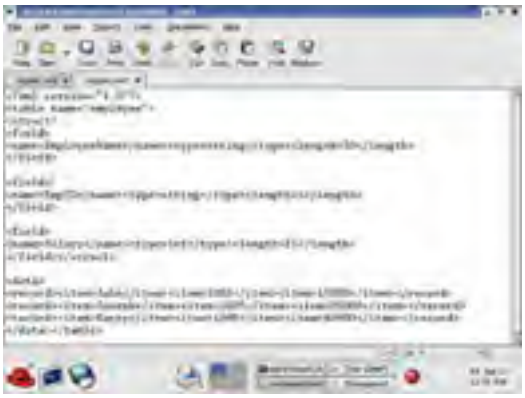


Figure 8-5: The export.xml File

After the XML document is generated using the DOM approach, you need to close the database connection using the mysql_close() function.

Figure 8-6 shows the output obtained after running Listing 8-7 from the Web browser:

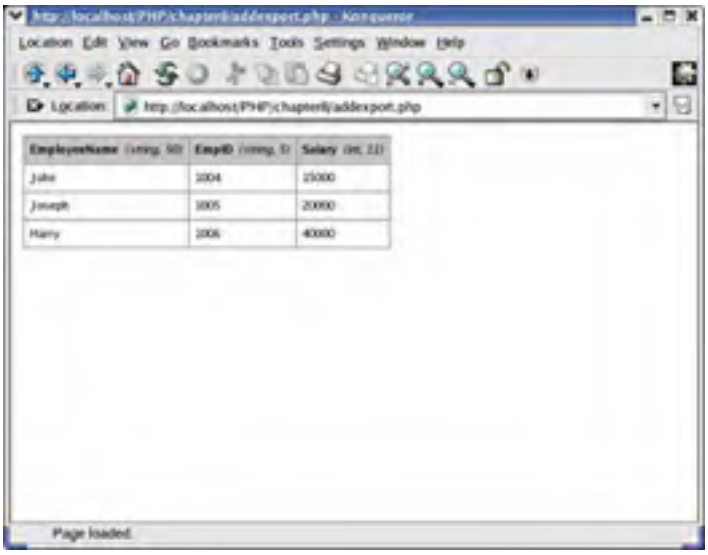


Figure 8-6: Generating the Contents of Employee Table in HTML

PREVIOUS NEXT

Importing XML Data to a Database

In addition to exporting database records to XML documents, PHP lets you import data from an XML document into a database. Using the DOM and SAX approach, you can construct SQL queries and insert data into a database from an XML document. To import XML data into a database:

1. Parse the XML data using the DOM or the SAX approach.
2. Create a list of field value pairs.
3. Insert the field value pairs in the database.
4. Run the database query to test whether data is inserted or not and close the database connection.

Connecting to a Database to Import Data

You need to establish a connection with the MySQL database to import data from an XML document to the database. The following code shows how to connect to the MySQL database:

```
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($db) or die ("Unable to select database!");
```

In the above code, a connection to the database is established using the mysql_connect() function. You need to specify the hostname, username, and password as arguments to the mysql_connect() function.

Parsing an XML Document

To import XML document into a database, you need to parse the XML document. You can parse the XML document using both the SAX and DOM approach. For example, to import the XML document, student.xml file, that stores the student information in a database, you need to parse the student.xml file.

Listing 8-8 shows the contents of the student.xml file:

Listing 8-8: Content of the student.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
<item>
<name>John</name>
<age>15</age>
<address>New York</address>
<standard>10</standard>
</item>
<item>
<name>Joseph</name>
<age>16</age>
<address>New York</address>
<standard>12</standard>
</item>
<item>
<name>Mary</name>
<age>14</age>
<address>New Jersey</address>
<standard>8</standard>
</item>
</list>
```

The above listing shows the content of the student.xml file that stores student information, such as name, age, address, and standard.

You need to create a table in the database that can store the data imported from the student.xml file.

Listing 8-9 shows how to create the student table in the information database:

Listing 8-9: Creating the Student Table

```
use information
mysql> CREATE TABLE student
-> (
-> name Varchar(30) NOT NULL,
-> age Integer NOT NULL,
-> address Varchar(30) NOT NULL,
-> standard Integer
-> );
```

The above listing creates the student table, in which you can insert data from the student.xml file.

Listing 8-10 shows how to import the student.xml file into a database, by parsing the student.xml file using the SAX approach:

Listing 8-10: Parsing the student.xml File using SAX

```
<?php
$cTag = "";
// Array to hold the values for the SQL statement.
$values = array();
$elements = array("name", "age", "address", "standard");
// XML file to parse
$xmlFile = "student.xml";
// Database parameters
$hostname = "localhost";
$username = "root";
&password = "root123";
$database = "information";
$table = "student";
// Processes on encountering a opening tag in the XML.
function startElementHandler($parser, $n1, $attributes)
{
   global $cTag;
   $cTag = $n1;
}
// Processes on encountering a closing tag in the XML.
function endElementHandler($parser, $n1)
{
   global $values, $cTag;
   // Import database link and table name.
   global $connection, $table;
   // if ending <item> tag
   // Implies end of record.
   if (strtolower($n1) == "item")
   {
      // Generating the query string.
      $query = "INSERT INTO student";
      $query .= "(name, age, address, standard) ";
      $query .= "VALUES(\"" . join("\", \"", $values) . "\");";
      // Processing the query.
      $result = mysql_query($query) or die ("Error in query: $query. " .
      mysql_error());
      // Reset all internal counters and arrays.
      $values = array();
      $cTag = "";
   }
}
// Processes on encountering a closing tag in the XML.
function characterDataHandler($parser, $data)
{
    global $cTag, $values, $elements;
   // lowercase tag name
   $cTag = strtolower($cTag);
   // Look for tag in $elements[] array.
   if (in_array($cTag, $elements) && trim($data) != "")
   {
      array
      $values[$cTag] = mysql_escape_string($data);
   }
}
// Initializing the SAX parser.
$xml_parser = xml_parser_create();
// Set callback functions.
xml_set_element_handler($xml_parser, "startElementHandler", "endElementHandler");
xml_set_character_data_handler($xml_parser, "characterDataHandler");
// Open connection to database.
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($database) or die ("Unable to select database!");
// read XML file
if (!($fp = fopen($xmlFile, "r")))
{
   die("File I/O error: $xmlFile");
}
// parse XML
while ($data = fread($fp, 4096))
{
   // error handler
   if (!xml_parse($xml_parser, $data, feof($fp)))
   {
      $error_code = xml_get_error_code($xml_parser);
      die("XML parser error (error code " . $error_code . "): " .
      xml_error_string($error_code) . "<br>Error occurred at line " .
      xml_get_current_line_number($xml_parser));
   }
}
?>
```

The above listing shows how to parse the student.xml file using the SAX approach. In the above listing:

- The instance of the SAX parser is initialized using the xml_parser_create() function, and is configured to call various functions, such as startElemenHandler() and endElementHandler().

- The startElemenHandler() function executes when a starting tag is processed in the XML document, and the endElementHandler() function executes when the closing tag in the XML document is processed.

You need to pass the tag name and the parser as arguments to the tag handler functions, such as startElemenHandler() and endElementHandler(). The characterDataHandler() function executes when the character data in the XML document is processed. You need to specify the Character Data (CDATA) text as arguments in the characterDataHandler() function.

The SAX parser retrieves and stores data from the student.xml file in the associative array, $values. The SAX parser retrieves data after finding character data having element name, such as name, age, address, and standard. The SAX parser creates a query string from the values obtained from the $values array.

You can also parse the XML document using the DOM approach.

Listing 8-11 shows how to parse the XML document using DOM:

Listing 8-11: Parsing XML Document using DOM

```
<?php
$xml_file = "student.xml";
$doc = xmldocfile($xml_file);
$name = array();
$age = array();
$address = array();
$standard = array();
$root = $doc->root();
$nodes = $root->children();
$a=0;
$b=0;
$c=0;
for($x=0;$x<sizeof($nodes);$x++)
{
   if ($nodes[$x]->type==XML_ELEMENT_NODE)
   {
      $text=$nodes[$x]->children();
      for($i=0;$i<sizeof($text);$i++)
      {
         $temp=$text[$i]->children();
         for($j=0;$j<sizeof($temp);$j++)
         {
            echo $temp[$j]->content;
         }
      }
   }
}
?>
```

The above listing shows how to parse the student.xml file using the DOM approach.

Closing the Connection to a Database After Importing Data

You need to close the database connection after importing the XML document in the database. The code to close the database connection is:

```
xml_parser_free($xml_parser);
mysql_close($connection);
```

The above code shows how to close the connection to the database using the mysql_close() function. The xml_parser_free() function frees the parser and returns true if $xml_parser is valid otherwise returns the value, False.

Alternative Method to Import XML Documents

Using PHP, you can import XML documents in a database without specifying the table name, field names, and values in the SQL statement. For example, you want to import the datastudent.xml file into a table, stud.

Listing 8-12 shows the contents of the datastudent.xml file:

Listing 8-12: Contents of datastudent.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<table name1="stud">
<record>
<age>12</age>
<address>New York</address>
<standard>12</standard>
<name>Mary</name>
</record>
<record>
<age>14</age>
<address>New York</address>
<standard>10</standard>
<name>John</name>
</record>
<record>
<age>15</age>
<address>New Jersey</address>
<standard>12</standard>
```

```
<name>Tom</name>
</record>
</table>
```

The above listing stores student information, such as name, record, address, age, and standard. The name of the table, in which the contents of the datastudent.xml file are imported, is specified as an attribute to the <table> element of the XML document. The student information, such as name, address, standard, and age, are specified with the <record> element.

Listing 8-13 shows how to import an XML document in a database without specifying the table and field names:

Listing 8-13: Importing the XML Document

```
<?php
// Initialize some variables
$cTag = "";
$fields = array();
$values = array();
// XML file to parse
$xml_file="datastudent.xml";
// Database parameters
// Get these via user input
$hostname = "localhost";
$username = "root";
$password = "";
$db = "information";
// Called when parser finds start tag.
function startElementHandler($parser, $name1, $attributes)
{
   global $cTag, $table;
   $cTag = $name1;
   // Get table name.
   if (strtolower($cTag) == "table")
   {
      foreach ($attributes as $v)
      {
         $table=$v;
      }
   }
}
// Called when parser finds end tag.
function endElementHandler($parser, $name1)
{
   global $fields, $values, $count, $cTag;
   // Import database link and table name.
   global $connection, $table;
    if (strtolower($name1) == "record")
    {
      // Generate the query string.
      $query = "INSERT INTO $table";
      $query .= "(" . join(", ", $fields) . ")";
      $query .= " VALUES(\"" . join("\", \"", $values) . "\");";
      // Execute query
      mysql_query($query) or die ("Error in query: $query. " .mysql_error());
      // Reset all internal counters and arrays.
      $fields = array();
      $values = array();
      $count = 0;
      $cTag = "";
    }
 }
// Called when parser finds cdata
function characterDataHandler($parser, $data)
{
   global $fields, $values, $cTag, $count;
   if (trim($data) != "")
   {
      // Add field-value pairs to $fields and $values array.
      // The index of each array is used to correlate the field-value pairs.
      $fields[$count] = $cTag;
      // Escape quotes with slashes
      $values[$count] = mysql_escape_string($data);
      $count++;
   }
}
// Initialize parser
$xml_parser = xml_parser_create();
// Set callback functions.
xml_set_element_handler($xml_parser, "startElementHandler", "endElementHandler");
xml_set_character_data_handler($xml_parser, "characterDataHandler");
// Open connection to database.
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($db) or die ("Unable to select database!");
// read XML file
if (!($fp = fopen($xml_file, "r")))
```

```
{
   die("File I/O error: $xml_file");
}
// Parse XML
while ($data = fread($fp, 4096))
{
   // Error handler
   if (!xml_parse($xml_parser, $data, feof($fp)))
   {
      $ec = xml_get_error_code($xml_parser);
      die("XML parser error (error code " . $ec . "): " . xml_error_string($ec) .
      "<br>Error occurred at line " . xml_get_current_line_number($xml_parser));
   }
}
// All done, clean up!
xml_parser_free($xml_parser);
mysql_close($connection);
?>
```

The above listing imports the datastudent.xml file without specifying the table name and field name values in the PHP script. You need to specify the hostname, username, and password of the MySQL database to connect to the database.

In the above listing, the xml_parser_create() function initializes the SAX parser, and the SAX parser processes various handler functions, such as xml_set_elemet_handler and xml_set_character_data_handler.

Figure 8-7 shows the output of Listing 8-13:

Figure 8-7: Contents of Table Imported from the XML Document

◄ PREVIOUS NEXT ►

Chapter 9: Creating an Online Shopping Cart Application

Download CD Content

The online shopping cart application allows an end user to search for a specific book in a database, place an order for the book, and purchase the book online. This application contains two sections, Admin and Client. The Admin section lets you administer the application by managing information related to the books, such as creating and removing a book category, adding a book to a category, and modifying the book information. The Client section lets the end user perform online transactions, such as searching for books based on either author or book title, and buying books.

This chapter discusses the architecture and database structure of the online shopping cart application. This chapter shows how to create the shopping cart application using PHP as the scripting language and MySQL as the database. This chapter discusses the process of administering the data related to books, such as adding and removing a new category, adding and removing a book from the database, viewing the book information, and placing an order to purchase the selected books. The chapter also describes the process of performing online transactions, such as buying a book, or searching for specific books based on the author name or book title.

Application Architecture

The architecture of the online shopping cart application contains two sections, Admin and Client. In addition, this application uses a database created in MySQL server to store data.

The Admin and Client Sections

The Admin section of the online shopping cart application performs administrative tasks to maintain the state of the application, such as adding or removing a category and adding or removing a book from a category. The Client section enables the end user perform activities related to online transactions, such as registering with the application, navigating through the various available categories to search for specific books, and adding the selected books to the cart.

Figure 9-1 shows the architecture of the online shopping cart application:

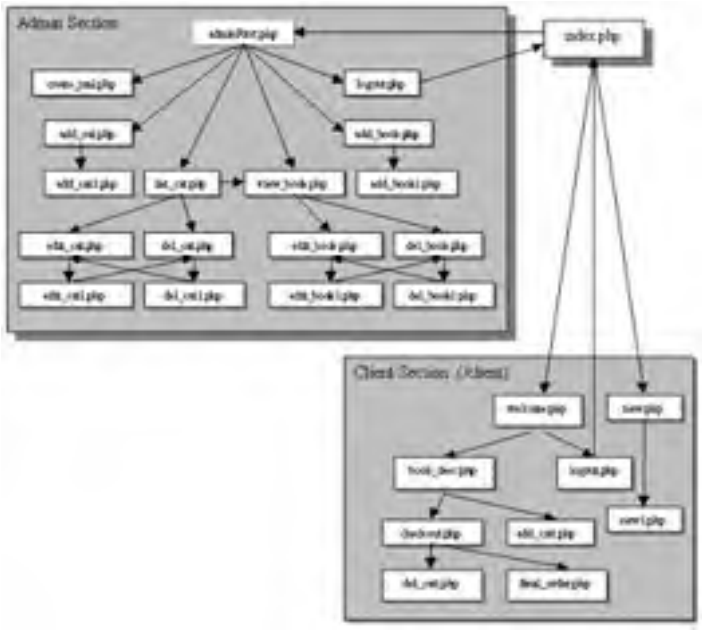


Figure 9-1: Online Shopping Cart Application Architecture

The index.php page is the first page of the application that allows the end users to log on to the application. The end user logs on as an administrator, using the administrative user ID. The index.php file contains a hyperlink to the new.php file that registers new end users and creates new client accounts.

The adminFirst.php page is the first Web page of the Admin section. It contains hyperlinks to six other Web pages, which perform specific operations related to the administration of the application. The Web pages include:

- The logout.php Web page is displayed when the end user logs out from the application.
- The add_book.php Web page adds a book in a category. The view_book.php Web page displays the details of a book.
- The list_cat.php Web page lists all the existing categories of the books. The list_cat.php Web page contains hyperlinks to the edit_cat.php and del_cat.php Web pages.
- The add_cat.php Web page adds a new category to the list of existing categories.

- The create_xml.php Web page generates the orders by creating an Extensible Markup Language (XML) file.
- The view_book.php Web page also lets you edit and delete a book from a category. It contains hyperlinks to the edit_book.php and del_book.php Web pages.

The edit_cat.php Web page receives the required parameters from the end user to edit a category and calls the edit_cat1.php Web page to edit a category. The del_cat.php Web page receives the required parameters from the end user to delete a category and calls the del_cat1.php Web page to delete a category.

The edit_book.php Web page receives the required parameters from the end user to edit the book information and calls the edit_book1.php Web page to edit a book. The del_book.php Web page receives the required parameters from the end user to delete a book and calls the del_book1.php Web page to delete a book.

The Web pages of the client section are stored in the client folder of the application. The index.php Web page directs all registered clients with valid account IDs to the welcome.php Web page. This is the first page of the client section. The Welcome.php Web page contains hyperlinks to the logout.php and book_desc.php Web pages. The logout.php Web page lets a client log out from the application and go back to the index.php Web page. The book_desc.php Web page contains a hyperlink to the checkout.php and add_cart.php Web pages. The add_cart.php Web page adds the selected book in the cart of the end user or client. The check_out.php Web page shows the status of the cart and contains a link to the final_order.php Web page, which displays the order number of the client.

Database Structure

The online shopping cart application uses the MySQL database, which consists of the category, book, user_profile, order1, transaction, and tmp tables. The category table stores information related to the categories of the books, such as category ID and name of the category.

Table 9-1 lists the structure of the category table:

Table 9-1: Structure of the category Table

field name	data type
tbl_id	int(11), primary key, auto_increment
item_type	varchar(50), unique, NOT NULL

The book table stores information about the books available in the online shopping cart application. The book table consists of information, such as book ID, category, book title, and author name.

Table 9-2 lists the structure of the book table:

Table 9-2: Structure of the book Table

field name	data type
item_no	varchar(20), primary key, NOT NULL
item_type	varchar(50) references category(item_type)
title	varchar(60), NOT NULL
author	varchar(60), NOT NULL
price	float, NOT NULL

The user_profile table stores the registration information of the end users registered using the online shopping cart application. The user_profile table stores information, such as the user name, password, address, phone number, and credit card information.

Table 9-3 lists the structure of the user_profile table:

Table 9-3: Structure of the user_profile Table

field name	data type
name	varchar(40), NOT NULL
user_id	varchar(20), primary key
password	varchar(20), NOT NULL
address_line1	varchar(40), NOT NULL
address_line2	varchar(40), NOT NULL
city	varchar(20), NOT NULL
country	varchar(20), NOT NULL
pin	varchar(20), NOT NULL
email_id	varchar(20), NOT NULL
phone_number	varchar(20), NOT NULL
card_no	varchar(20), NOT NULL

card_type	varchar(20), NOT NULL
expirty_date	varchar(20), NOT NULL
fax_number	varchar(20), NOT NULL

The order1 table stores the order information, such as the order number, book ID, and user ID.

Table 9-4 lists the structure of the order1 table:

Table 9-4: Structure of the order1 Table

field name	data type
tbl_id	int(11) references category(tbl_id)
order_no	int(11) primary key auto_increment
item_no	varchar(20) references book(item_no)
user_id	varchar(40) references user_profile(user_id)

The tmp table stores the session information for each order.

Table 9-5 lists the structure of the tmp table:

Table 9-5: Structure of the tmp Table

field name	data type
order_no	int(11), primary key auto_increment
user_id	varchar(20), NOT NULL
item_no	varchar(20), NOT NULL
sesid	varchar(50), NOT NULL
date	date, NOT NULL

Creating the Database and Tables

The online shopping cart application uses the shop.sql script to create the required databases and tables. You need to run the shop.sql script from the command prompt of the MySQL server to create the databases and the tables. The code to run the shop.sql script from the command line is:

```
mysql shop < shop.sql
```

In the above code, the shop parameter specifies the name of the database, and shop.sql is the script file that consists of the SQL statements to create the tables required to work with the online shopping cart application.

Listing 9-1 shows the content of the shop.sql script:

Listing 9-1: Creating Database Tables

```
# Table structure for table 'category'
CREATE TABLE category (
   tbl_id int(11) NOT NULL auto_increment,
   item_type varchar(20) NOT NULL,
   PRIMARY KEY (tbl_id),
   UNIQUE item_type (item_type)
);
# Table structure for table 'book'
CREATE TABLE book (
   item_no varchar(20) NOT NULL,
   item_type varchar(20) NOT NULL references category(item_type),
   title varchar(60) NOT NULL,
   author varchar(60) NOT NULL,
   price float DEFAULT '0' NOT NULL,
   PRIMARY KEY (item_no)
);
# Table structure for table 'user_profile'
CREATE TABLE user_profile (
   name varchar(40) NOT NULL,
   user_id varchar(20) NOT NULL,
   password varchar(20) NOT NULL,
   address_line1 varchar(40) NOT NULL,
   address_line2 varchar(40),
   city varchar(20) NOT NULL,
   country varchar(20) NOT NULL,
   pin varchar(20) NOT NULL,
   email_id varchar(20) NOT NULL,
   phone_number varchar(20) NOT NULL,
   card_no varchar(20) NOT NULL,
   expiry_date varchar(20) NOT NULL,
card_type varchar(20) NOT NULL,
   fax_number varchar(20) NOT NULL,
```

```
   PRIMARY KEY (user_id)
);
# Table structure for table 'order1'
CREATE TABLE order1 (
   tbl_id int(11) NOT NULL references category (tbl_id),
   order_no int(11) auto_increment,
   item_no varchar(20) NOT NULL references book(item_no),
   user_id varchar(40) NOT NULL references user_profile(user_id),
   PRIMARY KEY (order_no)
);
# Table structure for table 'tmp'
CREATE TABLE tmp (
order_no int(11) NOT NULL auto_increment,
   user_id varchar(20) NOT NULL,
   item_no varchar(20) NOT NULL,
   sesid varchar(50) NOT NULL,
   date date DEFAULT '0000-00-00' NOT NULL,
   PRIMARY KEY (order_no)
);
# Table structure for table 'transaction'
CREATE TABLE transaction (
   order_no int(11) NOT NULL auto_increment,
   user_id varchar(20) NOT NULL,
   date date DEFAULT '0000-00-00' NOT NULL,
   status varchar(20) NOT NULL,
   PRIMARY KEY (order_no)
);
```

The above listing creates the book, category, order1, user_profile, tmp, and transaction tables in the shop database. After creating all the tables, you need to insert a record in the user_profile table that contains the user ID and password. Use the following code to insert a row in the user_profile table:

```
INSERT INTO user_profile VALUES ( 'admin', 'admin', 'admin', '', '', '', '', '', '', '', '', '', '',
```

The above code creates an account that is used by the end user working as an administrator, with admin as the user ID and password.

◄ PREVIOUS NEXT ►

The Admin Section of the Online Shopping Cart Application

The administration process for the online shopping cart application involves adding a category, modifying a category, deleting a category, adding a new book, updating book information, and removing a book from the shop database.

The files of the Admin section are stored in the project folder. The client folder, present in the project folder, stores all the files required to perform the online transactions, such as the new.php file used for client registration. The project folder, apart from storing the client folder, stores the files required for administration, such as the add_cat.php file used for creating a new category. You need to specify the directory path of the files of the admin section in the address bar of the Web browser, to administer the data for the online shopping cart application. For example, if the PHP files are stored in the document root, /var/www/html, of the Web server, the path to administer the online shopping cart application is, http://localhost/Project/index.php.

Accessing the Shopping Cart Application

To work with the application, end users need to log on either as a client or as an administrator. The index.php file, present in the project folder, represents the login page. The user ID and password for the administrator is admin. The end user needs to register to create a new client account. If the end user is logged in with the admin account, only then the end user can work as an administrator.

Listing 9-2 shows the index.php file that creates a login form to accept the user name and password information from the end user:

Listing 9-2: The index.php File

```
<html>
<body>
<table border="0" width="100%" cellpadding="1" cellspacing="1" bgcolor="#FFFFFF">
<tr><td><?php
print "<form method=\"get\" action=\"login.php\">
<tr bgcolor=\"#E8E8E8\" height=\"4\">
<td align=\"right\">
<font face=\"arial\" size=2 color=\"#FF0000\">USER NAME:</font>
</td>
<td align=\"left\">
<INPUT TYPE=\"text\" NAME=\"user_id\">
</td></tr>
<tr bgcolor=\"#E8E8E8\" height=\"4\">
<td align=\"right\">
<font face=\"arial\" size=2 color=\"#FF0000\">PASSWORD:</font></td>
<td align=\"left\">
<INPUT TYPE=\"password\" NAME=\"passwd\">
</td></tr>
<tr bgcolor=\"#E8E8E8\" height=\"4\">
<td align=\"center\" colspan=\"2\">
<INPUT TYPE=\"submit\" value=\"LOGIN\"></td></tr>
<tr bgcolor=\"#E8E8E8\" height=\"1\">
<td width=\"100%\" valign=\"middle\" align=\"center\" colspan=\"2\"> </td>
</tr>
<tr bgcolor=\"#E8E8E8\" height=\"1\">
<td width=\"100%\" valign=\"middle\" align=\"center\" colspan=\"2\">
<a href=\"client/new.php\">New User</a>
</td> </tr></form>";
print"</td></tr></table></body></html>";
?>
</tr>
<?php
include "bottom.php";
?>
```

The above listing creates a login page to accept the user name and password information. The user name and password determines whether the end user is an administrator or a client.

Figure 9-2 shows the login page created by the index.php file:

Figure 9-2: Viewing index.php File in Mozilla Web Browser

To log on as an administrator, end users need to use the login ID, admin. The password for the admin login ID is also admin. The administrator can check or change the password of the admin login in the user_profile table. When end user clicks the Submit button, the login.php file is processed to verify the user name and password.

Listing 9-3 shows the content of the login.php file:

Listing 9-3: Verifying the User Name and Password Information

```
<?php
$user_id=$_GET['user_id'];
$passwd=$_GET['passwd'];
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select * from user_profile where user_id='$user_id' and
password='$passwd'", $db);
$row = mysql_fetch_array($result);
$user=$row["user_id"];
if ($user=="admin")
{
   print "<html><head><title></title><head><body>
   <form name=\"sess\" action=\"adminFirst.php\" method=\"post\">
   </form>
   <script>
   document.sess.submit();
   </script>
   </body></html>";
}
else
{
   print"<html><head><title></title><head><body>
   <form name=\"sess\" action=\"client/logon.php\" method=\"get\">
   <input type=\"hidden\" name=\"login\" value=\"$user\">
   </form>
   <script>
   document.sess.submit();
   </script>
   </body></html>";
}
?>
```

The above listing verifies the submitted user name and password. If the specified user name and password do not match with the data in the tables, an error message is displayed to the end user. If the admin login ID is used, the home page of the administrative interface, adminFirst.php, is displayed in the Web browser. If any other login ID is used, the logon.php file present in the client folder of the application is displayed in the Web page.

Creating the Home Page for the Administrative Interface

All the pages in the Admin section of the application are divided into three sections: top, middle, and right.

Figure 9-3 shows the structure of the Web pages of the Admin section:

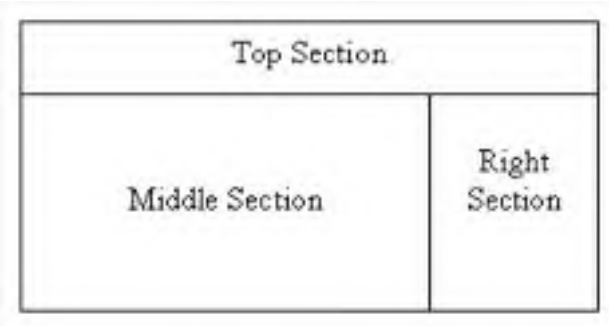


Figure 9-3: Structure of Web Pages of the Admin Section

```
<?php
include "tophtml.php";
include "middle.php";
include "right.php";
?>
```

The above code shows three PHP files included in the adminFirst.php file. The first file, tophtml.php, represents the top section.

Listing 9-4 shows the content of the tophtml.php file that the administrator can use to create the top section of the home page:

Listing 9-4: Creating the Top Section

```
<html>
<head>
<title>Shop Admin</title>
</head>
<body>
<div align="center">
<center>
<table border="0" width="600" cellspacing="0" cellpadding="0" bgcolor="#FFFFFF">
<tr>
<td width="600" height="20" colspan="5" bordercolor="#00FF00" bgcolor="#008000">
<p align="center"><b><font face="Arial" size="3" color="#FFFFFF"> SHOP
(ADMIN)</font></b></td>
</tr>
<tr height="2">
<td width="600" height="2" colspan="5" bgcolor="#000080"></td>
</tr>
```

The above listing creates the top section of the home page for the administrative interface of the online shopping cart application.

The middle section is represented by the middle.php file. The following code shows the content of the middle.php file:

```
<tr width="498" height="300" >
<td colspan="2" align="center">Select any option from the menu.</td>
```

The above code defines the first column of the row that contains the middle and right sections.

The right section displays the administrative options, such as adding a category, retrieving the category list, adding a book, and changing the password.

Listing 9-5 shows the content of the right.php file to create the right section of the home page:

Listing 9-5: Creating the Right Section of the Home Page

```
<td width="1" height="300" bgcolor="#000080"></td>
<td width="100" height="300" valign="top">
<table border="0" width="99" cellpadding="1" cellspacing="1" bgcolor="#FFFFFF">
<?php
echo "<tr  bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"logout.php\">
<font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B> LOGOUT</font></a></td></tr>
<tr bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"add_book.php\"><font face=\"verdana\" size=\"1\"
color=\"#FFFFFF\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B>Add New
Book</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"add_cat.php\"><font
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B>Add
Category</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a
href=\"create_xml.php\"><font face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font
style=\"font-size:11px;\" face=\"Helvetica\" color=\"#FFFFFF\"><B>Generate
Order</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"list_cat.php\"><font
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B> Category
List</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><b><font face=\"verdana\"
size=\"1\" color=\"#FFFFFF\">Select category</font></b></td></tr>";
$db=mysql_connect('localhost','root','');
if (!$db)
{
echo "Error When connecting to Database";
<td width="1" height="300" bgcolor="#000080"></td>^M
<td width="100" height="300" valign="top">^M
<table border="0" width="99" cellpadding="1" cellspacing="1" bgcolor="#FFFFFF">
<?php
echo "<tr  bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"logout.php\">
<font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B> LOGOUT</font></a></td></tr>
<tr bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"add_book.php\"><font face=\"verdana\" size=\"1\"
color=\"#FFFFFF\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B>Add New
Book</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"add_cat.php\"><font
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B>Add
Category</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a
href=\"create_xml.php\"><font face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font
style=\"font-size:11px;\" face=\"Helvetica\" color=\"#FFFFFF\"><B>Generate
Order</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"list_cat.php\"><font
```

```
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B> Category
List</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><b><font face=\"verdana\"
size=\"1\" color=\"#FFFFFF\">Select category</font></b></td></tr>";
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
mysql_query("use $db");
$result = mysql_query("select item_type from category");
while($row = mysql_fetch_array($result))
{
   print "<tr bgcolor=\"#E1E1E1\"><td align=\"center\"><a
   href=\"view_book.php?item_type=$row[0]&p=0\"><font style=\"font-size:11px;\"
   face=\"Helvetica\"
   color=\"#FF0000\"><B>$row[0]</font></a></td></tr>";
}
?>
<tr bgcolor="#FFFFFF"><td
align="center"> </font></a></td></tr>
</table>
</td>
<td width="1" height="300"
bgcolor="#000080"></td></tr></table></body></html>
```

The above listing creates the right section of the home page for the administrative interface of the online shopping cart application.

Figure 9-4 shows the home page for the administrative interface of the online shopping cart application.

Figure 9-4: Home Page of the Administrative Interface

Logging Out as an Administrator

When the administrator clicks the LOGOUT link, the logout.php file logs you out from the online shopping cart application.

Listing 9-6 shows the content of the logout.php file that lets the administrator log out from the online shopping cart application:

Listing 9-6: The logout.php File for the Administrator

```
<?php
include "tophtml.php";
echo "<tr width=\"600\" height = \"300\"><td width = \"600\" colspan = \"5\">";
?>
<table border="0" width="100%" height="300" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td align="center" colspan="5"> <FONT SIZE="3" FACE="verdana"
color="#FF3300"><b>Logout successfully</b></FONT></td></tr>
<tr bgcolor="#E8E8E8"> <td align="center" colspan="5"><a
href="index.php">Sign in again</a></td></tr>
</table>
</td></tr></table>
```

Figure 9-5 shows a message, confirming that the administrator has logged out from the application:

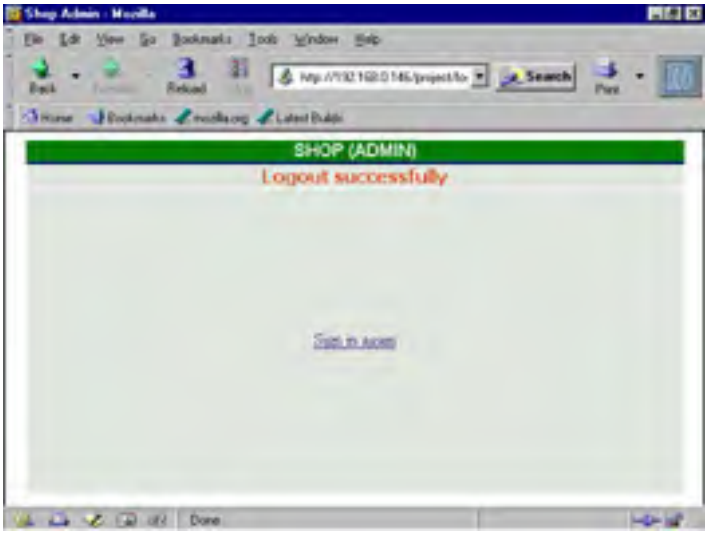


Figure 9-5: Message Confirming Administrative Log Off

Adding a New Category

When the administrator clicks the Add Category link shown in Figure 9-4, the Web server executes the add_cat.php file that displays a Web page to accept information for a new category.

Listing 9-7 shows how to create a Web page to add a new category in the database:

Listing 9-7: Creating a Web Page to Add a New Category

```
<?php
include "tophtml.php";
?>
<tr width="499"><td colspan="2">
<form name="addbook" action="add_cat1.php" action="GET">
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>ADD CATEGORY</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Category
Name: </font></td>
<td width="50%" valign="middle" align="left"><input type="text"
name="item_type"></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Add
Category" onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php include "right.php" ?>
<script>
function check()
{
   var error="false";
   if((document.addbook.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   if (error=="false")
   {
      document.addbook.submit();
   }
}
</script>
```

The above listing creates a Web page to accept information from the administrator to add a new category to the database.

Figure 9-6 shows the Web page to add a new category to the database table:

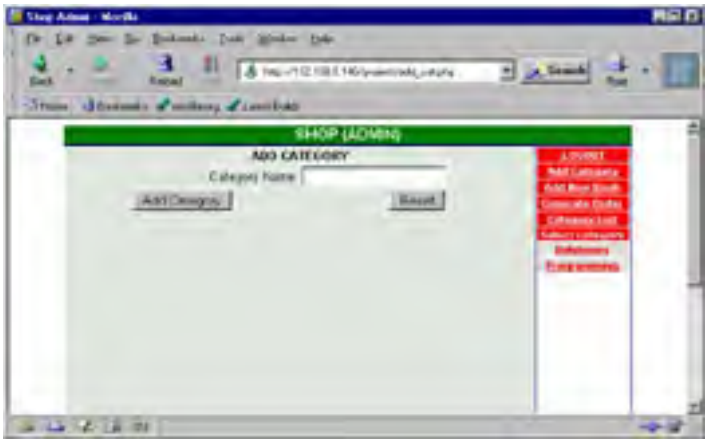


Figure 9-6: Web Page to Add a New Category

When the administrator clicks the Add Category button, the add_cat1.php file is executed by the Web server to add a new category to the database, as shown in Listing 9-8:

Listing 9-8: Adding a New Category to the Database

```
<?php
include "tophtml.php";
echo "<tr width=\"600\" height=\"300\"><td width=\"499\" colspan = \"2\">";
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$temp=$_GET['item_type'];
if(!$result_in=mysql_query("insert into category(tbl_id, item_type) values('NULL', '$temp')",
$db))
{
   $error=mysql_error();
}
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" height="4"><font face="Arial" size="2"><b>
<?php
if($error)
echo "<font color=\"#FF0000\">$error</font><br><br><a
href=\"javascript:history.back();\">Go Back</a>";
else
echo "<font color=\"#000000\">You have added category successfully</font>";
?>
</b></font></td></tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center"> 
</td></tr>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing inserts the category information in the database. If the category is added successfully, the message, You have added category successfully, is displayed to the administrator; else, an error message is displayed.

Adding a New Book

When the administrator clicks the Add New Book link shown in Figure 9-4, the add_book.php file creates a Web page to accept information for a new book from the administrator.

Listing 9-9 shows the add_book.php file that represents a Web page to add a new book to the database table:

Listing 9-9: The add_book.php File

```
<?php
include "tophtml.php";
?>
<tr width="499" ><td colspan="2">
<form name="addbook" action="add_book1.php" action="post">
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
```

```
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>ADD NEW BOOK</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book No:
</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="item_no"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Category:
</font></td>
<td width="50%" valign="middle" align="left">
<!--
<INPUT TYPE="text" NAME="item_type">
-->
<SELECT NAME="item_type">
<?php
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_cat=mysql_query("select * from category", $db);
while ($row_cat=mysql_fetch_array($result_cat))
{
   print "<option value=\"$row_cat[1]\">$row_cat[1]</option>";
}
?>
</SELECT>
</td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book
Title: </font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="title"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book
Author: </font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="author"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book
Price($): </font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="price"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Add Book"
onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php include "right.php"?>
<script>
function check()
{
   var error="false";
   if((document.addbook.item_no.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book No.");
   }
   /*
   if((document.addbook.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   */
   if((document.addbook.title.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Title");
   }
   if((document.addbook.author.value == "") && (error == "false"))
```

```
    {
       error="true";
       alert("Please Enter Author name.");
    }
    if((document.addbook.price.value == "") && (error == "false"))
    {
       error="true";
       alert("Please Enter Book Price.");
    }
    if (error=="false")
    {
       document.addbook.submit();
    }
}
</script>
```

The above listing creates a Web page to accept information to add a new book to the database, as shown in Figure 9-7:

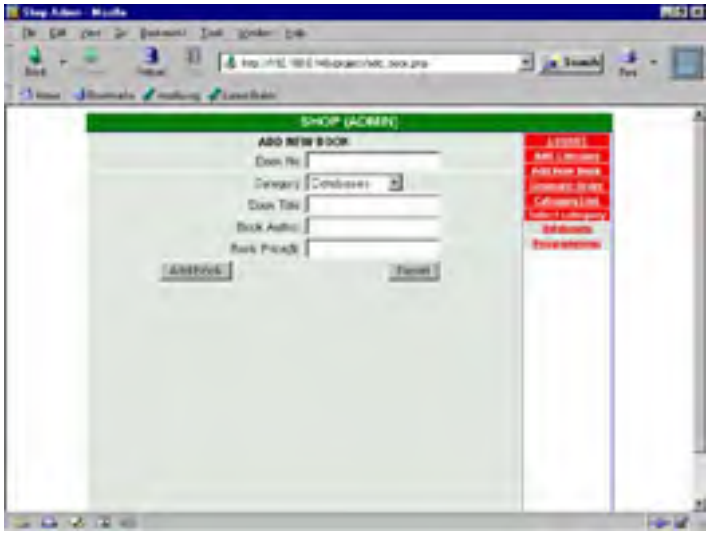


Figure 9-7: Web Page to Add a New Book

When the administrator clicks the Add Book button, the add_book1.php file adds a new book to the database.

Listing 9-10 shows the content of the add_book1.php file:

Listing 9-10: The add_book1.php File

```
include "tophtml.php";
$db=mysql_connect('localhost','root','');
echo "<tr width=\"600\" height=\"300\"><td width=\"499\" colspan=\"2\">";
if (!$db)
{
   echo "Error When connecting to Database";
}
$item_no=$_GET['item_no'];
$item_type=$_GET['item_type'];
$title=$_GET['title'];
$author=$_GET['author'];
$price=$_GET['price'];
mysql_select_db("shop", $db);
if(!$result_in=mysql_query("insert into book(item_no,item_type, title, author, price)
values('$item_no', '$item_type', '$title','$author',$price)", $db))
{
   $error=mysql_error();
}
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" height="4"><font face="Arial" size="2"><b>
<?php
if($error)
echo "<font color=\"#FF0000\">$error</font><BR><BR><a
href=\"javascript:history.back();\">Go Back</a>";
```

```
else
echo "<font color=\"#000000\">You have added one book successfully</font>";
?>
</b></font></td></tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center"> 
</td></tr>
</table>
</td>
<?php
include "right.php"
?>
```

The above listing inserts the book data in the database. If the book is added successfully, the message, You have added one book successfully, is displayed. If an error occurs, an error message is displayed in the middle section of the Web page.

Generating Orders

End users can place an order to buy the required books. The order is stored in an XML file. When the administrator clicks the Generate Order hyperlink, the Orders.xml XML file is created if it does not already exist in the /var/www/html directory. The create_xml.php file generates the Orders.xml file.

Listing 9-11 shows the content of the create_xml.php file:

Listing 9-11: The create_xml.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan = \"2\" align = \"center\">";
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select item_no, date, status, transaction.order_no from transaction, order1
where status='Pending' and transaction.order_no=order1.order_no", $db);
$doc = new_xmldoc("1.0");
$root = $doc->add_root("Orders");
while ($row=mysql_fetch_array($result))
{
   $result2=mysql_query("select title, author, price from book where item_no=$row[0]", $db);
   $row2=mysql_fetch_array($result2);
   $record = $root->new_child("Order", "");
   $record->set_attribute("number", $row[3]);
   $record->new_child("BookName", $row2[0]);
   $record->new_child("BookID", $row[0]);
   $record->new_child("Author", $row2[1]);
   $record->new_child("Price", $row2[2]);
   $record->new_child("Status", $row[2]);
   $record->new_child("Order Date", $row[1]);
}
$fp = fopen("/var/www/html/Orders.xml", "w+");
$rt= $doc->dumpmem();
fwrite($fp, $rt, strlen($rt));
echo "XML File Created";
?>
</td>
<?php
include "right.php"
?>
```

The above listing creates the Orders.xml file in the var/www/html directory. If the file is created successfully, the message, XML File Created, is displayed in the middle section of the Web page.

Listing Categories

When the administrator clicks the Category List hyperlink in the right panel of the home page, shown in Figure 9-4, the list_cat.php file is processed to retrieve and display all the category names.

Listing 9-12 shows how to create a Web page to display the names of all the categories:

Listing 9-12: The list_cat.php File

```
<?php
include "tophtml.php";
?>
<tr width="499"><td colspan="2">
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
```

```
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Category Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from category", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   //if ($j < $rec_per_page) {
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_cat.php?tbl_id=$row[0]\"><font face=\"Arial\"
   size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"del_cat.php?tbl_id=$row[0]&item_type=$row[1]\"><font face=\"Arial\"
   size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print "<tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php include "right.php"?>
```

The above listing retrieves a list of all the categories that exist in the database.

Figure 9-8 shows the output when list_cat.php is viewed in the Web browser:

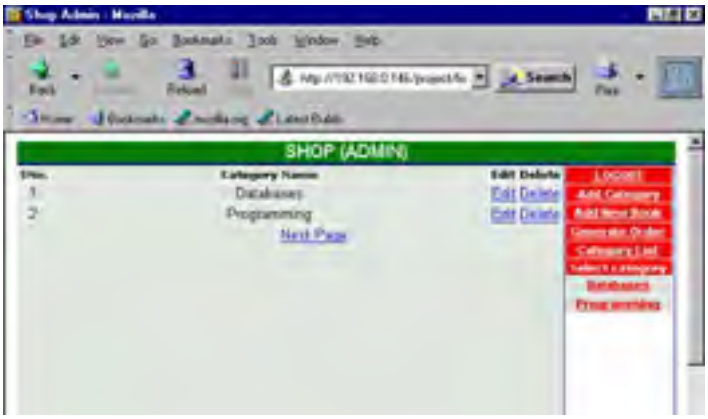


Figure 9-8: Web Page to Display All Category Names

Modifying a Category

When the administrator clicks the Edit link corresponding to a category name, as shown in Figure 9-8, the edit_cat.php file is processed. The edit_cat.php file creates a Web page that accepts data to update the category name.

Listing 9-13 shows the content of the edit_cat.php file:

Listing 9-13: The edit_cat.php File

```
<?php
include "tophtml.php";
$db=mysql_connect('localhost','root','');
echo "<tr width=\"499\"><td colspan = \"2\">";
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$temp=$_GET['tbl_id'];
$result=mysql_query("select * from category where tbl_id=$temp", $db);
$row=mysql_fetch_array($result);
$temp_val=$row[1];
?>
<form name="editcat" action="edit_cat1.php" action="post">
<input type="hidden" name="tbl_id" value='<?php echo "$temp" ?>'>
<input type="hidden" name="old_item_type" value='<?php echo "$row[1]"?>'>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>EDIT CATEGORY</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter
Category Name</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="item_type"
value='<?php echo "$temp_val" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Edit
Category" onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php
include "right.php"?>
<script>
function check()
{
   var error="false";
   if((document.editcat.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   if (error=="false")
   {
      document.editcat.submit();
   }
}
</script>
```

The above listing displays a Web page to modify the existing categories.

Figure 9-9 shows the Web page that appears when the administrator clicks the Edit hyperlink corresponding to the Databases category:

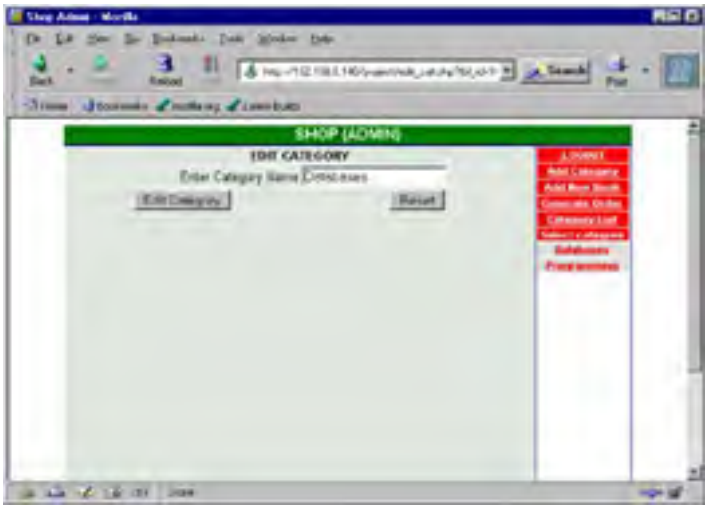


Figure 9-9: Modifying a Category Name

When the administrator clicks the Edit Category button, the edit_cat1.php file is processed to modify the data for an existing category in the database table.

Listing 9-14 shows the content of the edit_cat1.php file:

Listing 9-14: The edit_cat1.php File

```
<?php
include "tophtml.php";
$db=mysql_connect('localhost','root','');
echo "<tr width=\"499\"><td colspan=\"2\">";
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$temp1=$_GET['item_type'];
$temp2=$_GET['tbl_id'];
$result_catup=mysql_query("update category set item_type='$temp1' where tbl_id='$temp2'", $db);
$result_catup=mysql_db_query("$db","update $book_tbl set item_type='$item_type' where
item_type='$old_item_type'");
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Category Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from category", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   //if ($j < $rec_per_page) {
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a href=\"del_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
```

```
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing modifies the category name in the database table.

Deleting a Category

When the administrator clicks the Delete link corresponding to a category name, as shown in Figure 9-8, the del_cat.php file, which deletes the selected category from the database, is processed.

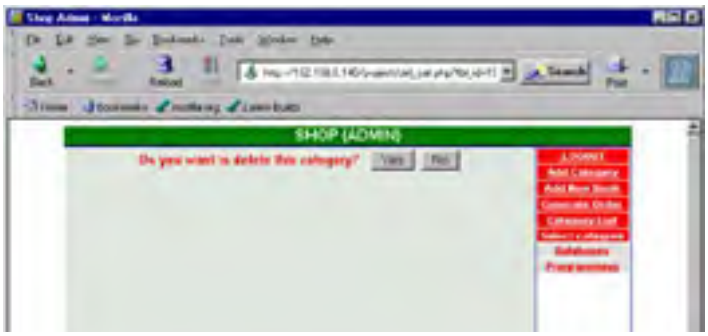
Listing 9-15 shows the content of the del_cat.php file that prompts the administrator before deleting a category from the database table:

Listing 9-15: The del_cat.php File

```
<?php
include "tophtml.php";
$temp2=$_GET['tbl_id'];
$temp1=$_GET['item_type'];
echo "<tr width=\"499\"><td colspan=\"2\">";
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="100%" valign="middle" align="center" colspan="4">
<form name="delcat" action="del_cat1.php" method="GET">
<input type="hidden" name="tbl_id" value='<?php echo "$temp2" ?>'>
<input type="hidden" name="item_name" value='<?php echo "$temp1" ?>'>
<font face="arial" size="2" color="#FF0000"><b>Do you want to delete this category?
</b></font>  <input type="submit" name="del"
value="Yes">  <input type="submit" name="del" value="No">
</form>
</td>
</tr>
<?php
print "<tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing creates a Web page, which prompts the administrator before deleting a category from the database.

Figure 9-10 shows the Web page that appears when the administrator clicks the Delete hyperlink corresponding to the Programming category, and verifies whether or not to delete the selected category:



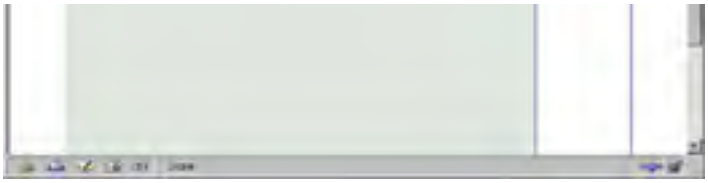


Figure 9-10: Web Page to Delete a Category

When the administrator clicks the Yes button, the del_cat1.php file is processed to delete the category from the database table, as shown in Listing 9-16:

Listing 9-16: The del_cat1.php File

```
<?php
include "tophtml.php";
$tbl_id=$_GET['tbl_id'];
$del=$_GET['del'];
$item_type=$_GET['item_name'];
echo "<tr width=\"499\"><td colspan = \"2\">";
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
if ($del=="Yes")
{
   $result_catdel=mysql_query("delete from category where tbl_id='$tbl_id'", $db);
}
$result_book=mysql_query("delete from book where item_type='$item_type'", $db);
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Category Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from category", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   //if ($j < $rec_per_page) {
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a href=\"del_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
```

```
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing deletes a category from the database table. The administrator is redirected to the Web page that displays the updated category list.

Modifying Book Data

When the administrator clicks any of the category items displayed in the right section of Figure 9-10, the view_book.php file is executed by the Web server to display a list of all the books present in that category.

Listing 9-17 shows the content of the view_book.php file:

Listing 9-17: The view_book.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan=\"2\">";
$item_type=$_GET['item_type'];
echo $item_type;
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Book Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select * from book where item_type='$item_type'", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[2]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_book.php?tbl_id=$row[0]\"><font face=\"Arial\"
   size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"del_book.php?tbl_id=$row[0]&item_type=$row[1]\"><font face=\"Arial\"
   size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"        
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
```

```
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing displays a Web page that contains information about all the books present in a category.

Figure 9-11 shows the output of the view_book.php file when the Database category is selected from the right section of the adminFirst.php Web page:

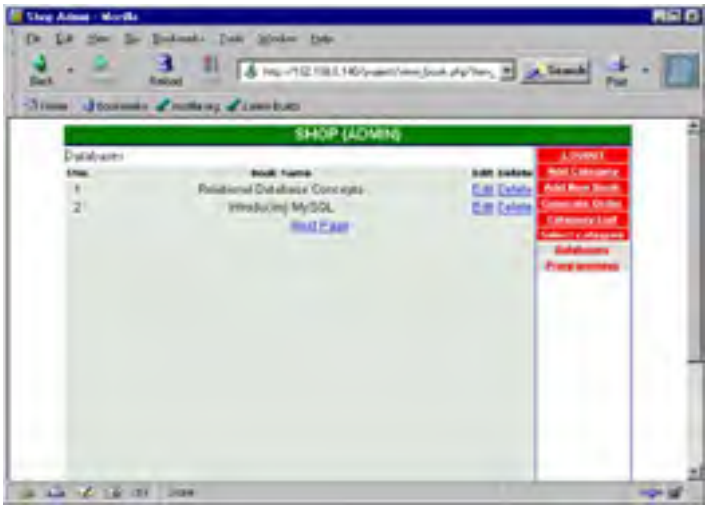


Figure 9-11: Books Present in the Database Category

The administrator needs to click the Edit hyperlink to modify the data of a book in the database. When the administrator clicks the Edit hyperlink corresponding to a book name, the edit_book.php file is executed, which lets the administrator enter new data for the selected book.

Listing 9-18 shows the content of the edit_book.php file:

Listing 9-18: The edit_book.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan=\"2\">";
$db=mysql_connect('localhost','root','');
$tbl_id=$_GET['tbl_id'];
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select * from book where item_no='$tbl_id'", $db);
$row=mysql_fetch_array($result);
?>
<form name="addbook" action="edit_book1.php" action="GET">
<input type="hidden" name="item_no" value='<?php echo"$tbl_id" ?>'>
<input type="hidden" name="item_type" value='<?php echo"$item_type" ?>'>
<input type="hidden" name="p" value='<?php echo"$p" ?>'>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>EDIT YOUR BOOK</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial"> Book
No.</font></td>
<td width="50%" valign="middle" align="left"><?php echo"$row[0]" ?></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Category</font></td>
<td width="50%" valign="middle" align="left">
<!--
<INPUT TYPE="text" NAME="item_type"  value='<?php echo"$row[1]" ?>'></td>
```

```
-->
<SELECT NAME="item_type">
<?php
$result_cat=mysql_query("select * from book", $db);
while ($row_cat=mysql_fetch_array($result_cat))
{
   if ($row[1] == $row_cat[1])
   print "<option value=\"$row_cat[1]\" selected>$row_cat[1]</option>";
   else
   print "<option value=\"$row_cat[1]\">$row_cat[1]</option>";
}
?>
</SELECT>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Title</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="title"
value='<?php echo"$row[2]" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Author</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="author"
value='<?php echo"$row[3]" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Price ($)</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="price"
value='<?php echo"$row[4]" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Edit Book"
onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php
include "right.php" ?>
<script>
function check()
{
   var error="false";
   if((document.addbook.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   if((document.addbook.title.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Title");
   }
   if((document.addbook.author.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Author name.");
   }
   if((document.addbook.price.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Price.");
   }
   if (error=="false")
   {
      document.addbook.submit();
   }
}
</script>
```

The above listing creates a Web page to accept modified data for an existing book from the administrator, as shown in Figure 9-12:

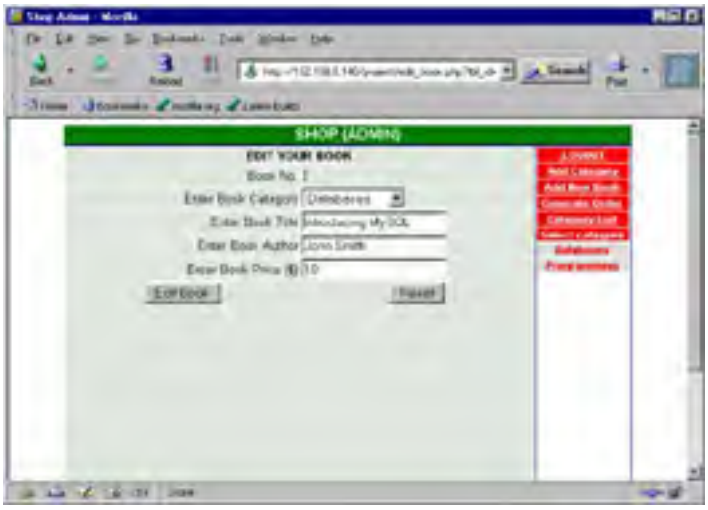


Figure 9-12: Modifying Book Data

When the administrator clicks the Edit Book button, the edit_book1.php file executes to modify the data for an existing book in the database table.

Listing 9-19 shows the content of the edit_book1.php file:

Listing 9-19: The edit_book1.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan=\"2\">";
$item_no=$_GET['item_no'];
$item_type=$_GET['item_type'];
$author=$_GET['author'];
$price=$_GET['price'];
$title=$_GET['title'];
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_edit=mysql_query("update book set
item_type='$item_type',title='$title',author='$author',price=$price where
item_no='$item_no'", $db);
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="32" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="64" valign="middle" align="center"><font size="1"
face="verdana"><b>Item
No.</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Category</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Title</b></font></td>
<td width="68" valign="middle" align="center"><b><font size="1"
face="verdana">Author</font></b></td>
<td width="33" valign="middle" align="center"><b><font size="1"
face="verdana">Price($)</font></b></td>
<td width="39" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="39" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from book where item_type='$item_type'", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"32\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"64\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[0]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
```

```
   size=\"2\">$row[2]</font></td>
   <td width=\"68\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[3]</font></td>
   <td width=\"33\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[4]</font></td>
   <td width=\"33\" align=\"center\"><a
   href=\"edit_book.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Edit</font></a></td>
   <td width=\"6\" align=\"center\"><a href=\"del_book.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Delete</font></a></td>
   </tr>";
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"8\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"8\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing modifies the book data in the database table and redirects the administrator to the Web page that displays the books data for the selected category.

Removing a Book

The administrator needs to click the Delete hyperlink corresponding to a book name, as shown in Figure 9-11, to delete a book from a category. The del_book.php file executes to delete the book from the category when the administrator clicks the Delete hyperlink. The del_book.php file represents a Web page that prompts the administrator before deleting the book from the database.

Listing 9-20 shows the content of the del_book.php file:

Listing 9-20: The del_book.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\" ><td colspan=\"2\" align=\"top\">";
$item_no=$_GET['tbl_id'];
?>
<table border="0" width="100%" height="100%" cellpadding="1" cellspacing="1"
bgcolor="#ffffff">
<tr bgcolor="#E8E8E8" height="4">
<td width="32" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="64" valign="middle" align="center"><font size="1"
face="verdana"><b>Item
No.</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Category</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Title</b></font></td>
<td width="68" valign="middle" align="center"><b><font size="1"
face="verdana">Author</font></b></td>
<td width="33" valign="middle" align="center"><b><font size="1"
face="verdana">Price($)</font></b></td>
</tr>
<?php
$j=0;
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result = mysql_query("select * from book where item_no='$item_no'");
```

```
while($row = mysql_fetch_array($result))
{
   $j++;
   print " <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"32\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"64\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[0]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[2]</font></td>
   <td width=\"68\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[3]</font></td>
   <td width=\"33\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[4]</font></td>
   </tr>";
   print " <tr bgcolor=\"#E8E8E8\" height=\"4\" >
   <td width=\"100%\" align=\"center\" colspan=\"6\"><font face=\"Arial\" size=\"2\"
   color=\"#FF0000\"><b>Do you want to delete this
   record.  </b>
   <a
   href=\"del_book1.php?item_no=$row[0]&item_type=$row[1]&title=$row[2]&p=$p&
   del=y\">Yes</a>   <a
   href=\"del_book1.php?item_no=$row[0]&item_type=$row[1]&p=$p&del=n\">
   No</a></font></td></tr>";
}
?>
<tr height="400" bgcolor="E8E8E8"><td colspan="6"> </td></tr>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing creates a Web page that prompts the administrator before deleting a book from the database.

Figure 9-13 shows the Web page that lets the administrator delete a book from the shop database:

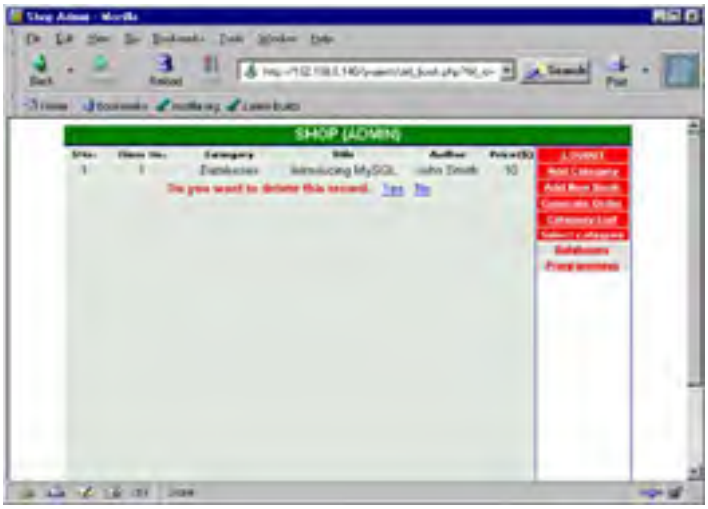


Figure 9-13: Deleting a Book

Listing 9-21 shows the content of the del_book1.php file:

Listing 9-21: The del_book1.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td align=\"center\" colspan=\"2\">";
$title=$_GET['title'];
$del=$_GET['del'];
$db=mysql_connect('localhost', 'root', '');
if(!$db)
{
   echo "Error when connecting to database";
}
mysql_select_db("shop", $db);
if ($del == 'y')
{
   mysql_query("delete from book where title='$title'", $db);
   print " Book Deleted !";
```

```
}else
print " Book is not deleted.";
echo "</td>";
include "right.php";
//print
"<script>location.replace(\"view_book.php?item_type=$item_type&p=$p\")</script
>";
?>
```

The above listing deletes a book from the database table if the administrator clicks the Yes hyperlink in Figure 9-13. If the administrator clicks the No hyperlink, the message, Book is not deleted, is displayed.The administrator is redirected to the Web page that displays the books for a selected category.

Working with the Online Shopping Cart Application

The Client section contains the Web pages that enables the end user as clients to perform activities, such as buying a book by adding it to the shopping cart, and performing the search operation based on author name or book title. All the pages in the Client section of the application are divided into three sections: top, right, and bottom. The top and bottom sections are same for all the Web pages of the Client section. The right section differs for each Web page.

The first Web page of the client section is represented by the welcome.php file. The following code shows the content of the welcome.php file:

```
<?php
echo "Welcome";
include "tophtml.php";
include "right.php";
include "bottomhtml.php";
?>
```

The above code shows that the welcome.php file includes three other files: tophtml.php, right.php, and bottomhtml.php.

Listing 9-22 shows the content of the tophtml.php file:

Listing 9-22: The tophtml.php File

```
<?php
$linkcol="#FF0000";
$linkcol1="#0000FF";
$bg="#7194D5";
$bg1="#4E6CA7";
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Book Shop</title>
<style type="text/css">
<!--
a {font: 10pt Arial, Verdana, Helvetica, sans-serif; text-decoration: none}
p {font: 12pt Arial, Helvetica, sans serif}
UL {font: 12pt Arial, Helvetica, sans serif}
-->
</style>
<script>
function usercheck()
{
   var user_id = document.newusr.user_id.value.replace(/\s+/,"");
   var passwd = document.newusr.passwd.value.replace(/\s+/,"");
   var cpasswd = document.newusr.cpasswd.value.replace(/\s+/,"");
   var name = document.newusr.name.value.replace(/\s+/,"");
   var address_line1 = document.newusr.address_line1.value.replace(/\s+/,"");
   //var address_line2 = document.newusr.address_line2.value.replace(/\s+/,"");
   var city = document.newusr.city.value.replace(/\s+/,"");
   var country = document.newusr.country.value.replace(/\s+/,"");
   var pin = document.newusr.pin.value.replace(/\s+/,"");
   var email_id = document.newusr.email_id.value.replace(/\s+/,"");
   var phone_number = document.newusr.phone_number.value.replace(/\s+/,"");
   var fax_number = document.newusr.fax_number.value.replace(/\s+/,"");
   var card_no = document.newusr.card_no.value.replace(/\s+/,"");
   var expiry_date = document.newusr.expiry_date.value.replace(/\s+/,"");
   var card_type = document.newusr.card_type.value.replace(/\s+/,"");
   var error="no";
   var message;
   var focus;
   if ((!user_id) && (error == "no"))
   {
      message = "Please enter User ID";
      focus="user_id";
      error="yes";
   }
   if ((!passwd) && (error == "no"))
   {
      message = "Please enter  Password";
      focus="passwd";
      error="yes";
   }
   if ((!cpasswd) && (error == "no"))
   {
      message = "Please enter Confirm Password";
      focus="cpasswd";
      error="yes";
   }
   if ((!name) && (error == "no"))
   {
      message = "Please enter  Name";
      focus="name";
```

```
      error="yes";
   }
   if ((!address_line1) && (error == "no"))
   {
      message = "Please enter  Address";
      focus="address_line1";
      error="yes";
   }
   if ((!city) && (error == "no"))
   {
      message = "Please enter  City";
      focus="city";
      error="yes";
   }
   if ((!country) && (error == "no"))
   {
      message = "Please enter  Country";
      focus="country";
      error="yes";
   }
   //PinCode Validation
   if ((!pin) && (error == "no"))
   {
      message = "Please enter  Pin Code";
      focus="pin";
      error="yes";
   }
   else
   {
   var valid ="0123456789";
   for (var i=0; i < pin.length; i++)
   {
      var temp = "" + pin.substring(i, i+1);
      if (valid.indexOf(temp) == "-1")
      {
         alert("Invalid characters in your pin code");
         focus="pin";
         error="yes";
         return false;
      }
   }
   if ((pin.length < 6) && (error == "no"))
   {
      alert("Invalid your pin code");
      focus="pin";
      error="yes";
   }
}
//validate phone number
if ((!phone_number) && (error == "no"))
{
   message = "Please enter  Phone Number";
   focus="phone_number";
   error="yes";
}
else
{
   var validph ="0123456789";
   for (var i=0; i < phone_number.length; i++)
   {
      var temp = "" + phone_number.substring(i, i+1);
      if (validph.indexOf(temp) == "-1")
      {
         alert("Invalid characters in your phone number.");
         focus="phone_number";
         error="yes";
         return false;
      }
   }
   if ((phone_number.length < 6) && (error == "no"))
   {
      alert("Invalid your phone number.");
      focus="phone_number";
      error="yes";
   }
}
if ((!email_id) && (error == "no"))
{
   message = "Please enter  Email ID";
   focus="email_id";
   error="yes";
}
//Card Number Validation
if ((!card_no) && (error == "no"))
{
   message = "Please enter  Card Number";
```

```
   focus="card_no";
   error="yes";
}
else
{
   var validcard ="0123456789";
   for (var i=0; i < card_no.length; i++)
   {
      var temp = "" + card_no.substring(i, i+1);
      if (validcard.indexOf(temp) == "-1")
      {
         alert("Invalid characters in your card number.");
         focus="card_no";
         error="yes";
         return false;
      }
   }
   if ((card_no.length < 6) && (error == "no"))
   {
      alert("Invalid your card number.");
      focus="card_no";
      error="yes";
   }
}
if ((passwd != cpasswd) && (error == "no"))
{
   message = "Check Confirm Password";
   focus="cpasswd";
   error="yes";
}
if (error == "yes")
{
   alert(message);
   var str ="document.newusr."+focus+".focus()";
   eval(str);
}
else
{
   document.newusr.submit();
}
</script>
<script language="JavaScript">
function MM_preloadImages()
{
   var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
   var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
   if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];}}
}
</script>
</HEAD>
<BODY BGCOLOR="#FFFFFF" onLoad="MM_preloadImages('/images/blank.gif')" >
<center>
<!-- begin table1-->
<TABLE BORDER="0" WIDTH="700" CELLPADDING="0" CELLSPACING="0" BGCOLOR="#FFFFFF" >
<tr>
<td>
<!-- begin table2-->
<table BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600" style="margin: 0px; padding: 0px">
<!--
<tr>
<td align="center">
<img src="/images/consbar.gif">
</td>
</tr>
-->
<tr>
<td HEIGHT="1" width="600" align="left" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1></td>
</tr>
<tr>
<td HEIGHT="1" width="600" align="left">
<table BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600">
<tr >
<td align="center" bgcolor="<?php if ($link == 1) echo $bg1; else echo $bg; ?>">
<A href="index.php?link=1"><font size="3" face="arial"
 color="#FFFFFF"><b> BOOK SHOP</b></font></A>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td HEIGHT="1" BGCOLOR="#1E237B" width="600" align="left" background="/images/line.gif">
```

```
<img SRC="/images/line.gif" ALT="-" BORDER=0 height=1 width=1></td>
</tr>
</table>
<!--end table2-->
</td>
</tr>
```

The above listing creates the top section of the home page of the online shopping cart application.

The middle section displays the options, such as searching a book, and listing the books from a category.

Listing 9-23 shows the content of the right.php file to create the middle section of the home page:

Listing 9-23: Creating the Middle Section of the Home Page

```
<td BGCOLOR="#FF0000" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="150" valign="top">
<table width="150" border="1" cellspacing="2" cellpadding="2">
<tr>
<td>
<table width="150" border="1" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<?php
$login=$_GET['login'];
if (($login == "ok") || ($sesid))
echo "<a href=\"../logout.php\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#33CC66\"><B>LOGOUT</B>";
else
echo "<a href=\"new.php\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FF0000\"><B>NEW USER</b></font></a><a
href=\"../index.php\"><font style=\"font-size:11x;\" face=\"Helvetica\"
color=\"#ffffff\"><B>LOGIN</B>";
?>
 </td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<form name="searchform" action="book_desc.php" method="GET"><BR>
<input type="hidden" name="req_from" value="search">
<input type="text" name="search_word" size="10"><BR>
<font style="font-size:12px;" face="Helvetica" color="#ffffff"><B>Search
by</B></font><BR>
<select name="search_by">
<option value="author">By Author</option>
<option value="title">By Title</option>
</select><BR><BR>
<input type="submit" value="Search">
</form>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<font style="font-size:12px;" face="Helvetica" color="#ffffff"><B>Select Book
Category</B></font></td>
</tr>
</table>
</td>
</tr>
<?php
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_cat = mysql_query("select item_type from category", $db);
print "<tr><td align=\"center\"><form name=\"form1\" method=\"get\"
action=\"book_desc.php\"><table width=\"110\" border=\"0\" cellspacing=\"1\"
cellpadding=\"1\"><tr><td align=\"center\"><SELECT NAME=\"item_type\">";
while ($row_cat=mysql_fetch_array($result_cat))
{
   print "<option value=\"$row_cat[0]\">$row_cat[0]</option>";
```

```
   //print "<tr><td bgcolor=\"#FF3300\" align=\"center\"><a
   href='book_desc.php?item_type=$row_cat[0]'><font style=\"font-size:12px;\"
   face=\"Helvetica\"
   color=\"#ffffff\">$row_cat[0]</font></a></td></tr>";
}
print "</SELECT></td></tr><tr><td align=\"center\"><input
type=\"submit\" value=\"BOOK LIST\"></td></tr>";
print "</table></form></td> </tr>";
?>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<font style="font-size:12px;" face="Helvetica" color="#ffffff"><B>Your
Cart</B></font></td>
</tr>
</table>
</td>
</tr>
<?php
$date = date('Y-m-d');
$result_cart = mysql_query("select book.title from tmp,book where tmp.user_id='$user' and
tmp.sesid='$sesid' and tmp.date='$date' and book.item_no=tmp.item_no", $db);
print "<tr><td align=\"center\"><table width=\"110\" border=\"0\"
cellspacing=\"1\" cellpadding=\"1\">";
while ($row_cart=mysql_fetch_array($result_cart))
{
   print "<tr><td align=\"center\"><font style=\"font-size:12px;\"
   face=\"Helvetica\"
   color=\"#000000\">$row_cart[0]</font></a></td></tr>";
}
print "</table></td> </tr>";
?>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<?php
if (($login == "ok") || ($sesid))
print "<a href=\"checkout.php\"><font style=\"font-size:12px;\" face=\"Helvetica\"
color=\"#ffffff\"><B>Check Out</B></font></a></td>";
?>
</tr>
</table>
</td>
</tr>
</table>
</td>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height="1" width="1">
</td>
```

The above listing creates the middle section of the home page for the Client section.

Listing 9-24 shows the bottom section of the home page created using the bottomhtml.php file:

Listing 9-24: The bottomhtml.php File

```
<tr>
<td HEIGHT="1" BGCOLOR="#1E237B" width="600" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1></td>
</tr>
</table>
<!-- end table3-->
</td>
</tr>
<tr><td>
<table border=0 width=90%>
<tr>
<td align="center" colspan="3"><font face="arial" size="1"
color="#1E237B"> </font></td>
</tr>
</table>
</td></tr>
</table>
</body>
</html>
```

The above listing creates the bottom section of the home page for the online shopping cart application.

Figure 9-14 shows the home page of the online shopping cart application:

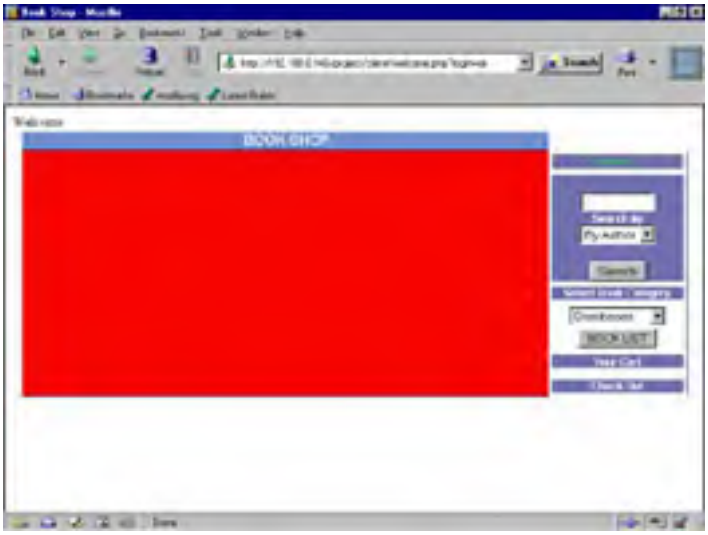


Figure 9-14: Home Page of the Client Section

Registering a New User

To initiate the registration process, end user needs to click the New User link, as shown in Figure 9-2. The new.php file is executed by the Web server that represents a Web page to accept data for a new end user.

Listing 9-25 shows the content of the new.php file:

Listing 9-25: The new.php File

```
<?php
include("tophtml.php");
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td>
<table width=100%>
<tr>
<td align="center">
<form name="newusr" action="new1.php" method="GET">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b>User Name *</b></font>
</td>
<td>
<input type="text" name="user_id" maxlength="20" value="<?php if ($user_id) echo
"$user_id"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b>Password *</b></font>
</td>
<td>
<input type="password" name="passwd" maxlength="10" value="<?php echo "$passwd";
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b>Confirm Password *</b></font>
</td>
<td>
<input type="password" name="cpasswd" maxlength="10" value="<?php if ($cpasswd) echo
"$cpasswd"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Name *</b></font>
</td>
<td>
<input type="text" name="name" maxlength="40" value="< ?php if ($name) echo "$name";
```

```
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Address1 *</b></font>
</td>
<td>
<input type="text" name="address_line1"  value="<?php if ($address_line1) echo
"$address_line1"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Address2</b></font>
</td>
<td>
<input type="text" name="address_line2" value="< ?php if ($address_line2) echo
"$address_line2";?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> City *</b></font>
</td>
<td>
<input type="text" name="city" value="<?php if ($city) echo "$city"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Country *</b></font>
</td>
<td>
<input type="text" name="country"  value="<?php if ($country) echo "$country"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Pin Code *</b></font>
</td>
<td>
<input type="text" name="pin"  value="<?php if ($pin) echo "$pin"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Email ID *</b></font>
</td>
<td>
<input type="text" name="email_id"  value="<?php if ($email_id) echo "$email_id";
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Phone Number *</b></font>
</td>
<td>
<input type="text" name="phone_number"  value="<?php if ($phone_number) echo
"$phone_number";?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Fax Number </b></font>
</td>
<td>
<input type="text" name="fax_number"  value="<?php if ($fax_number) echo "$fax_number";
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Card Number *</b></font>
</td>
<td>
<input type="text" name="card_no"  value="<?php if ($card_no) echo "$card_no"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Expiry Date</b></font>
</td>
<td>
<input type="text" name="expiry_date"  value="<?php if ($expiry_date) echo "$expiry_date";
```

```
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Card Type</b></font>
</td>
<td>
<select name="card_type">
<option value="American Express">American Express</option>
<option value="Master Card">Master Card</option>
<option value="Visa">Visa</option>
</select>
<!--
<input type="text" name="card_type" value="<?php if ($card_type) echo "$card_type";
?>">
-->
</td>
</tr>
<tr>
<td colspan="2" align="center"><input type="submit" value="Submit"></td>
</tr>
</table>
</form>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
<?php
include("bottumhtml.php");
?>
```

The above listing creates a Web page to accept information to register a new end user, as shown in Figure 9-15:

Figure 9-15: Web Page to Register New End User

When the end user clicks the Submit button, the new1.php file is processed to add the end user information to the database table.

Listing 9-26 show the content of the new1.php file:

Listing 9-26: The new1.php File

```
<?php
settype($error,"string");
settype($order_list,"string");
settype($user,"string");
settype($sesid,"string");
$name=$_GET['user'];
$user_id=$_GET['user_id'];
$passwd=$_GET['passwd'];
$address_line1=$_GET['address_line1'];
$address_line2=$_GET['address_line2'];
$city=$_GET['city'];
$country=$_GET['country'];
$pin=$_GET['pin'];
$email_id=$_GET['email_id'];
$phone_number=$_GET['phone_number'];
$card_no=$_GET['card_no'];
$expiry_date=$_GET['expiry_date'];
$card_type=$_GET['card_type'];
$fax_number=$_GET['fax_number'];
/*
echo "value ". $name;
echo $user_id;
echo $passwd;
echo $address_line1;
echo $address_line2;
echo $city;
echo $country;
echo $pin;
echo $email_id;
echo $phone_number;
echo $card_no;
echo $expiry_date;
echo $card_type;
echo $fax_number;
*/
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$query = mysql_query("insert into
user_profile(name,user_id,password,address_line1,address_line2,city,country,pin,email_id,
phone_number,card_no,expiry_date,card_type,fax_number)
values('$name','$user_id','$passwd','$address_line1','$address_line2','$city','$country',
'$pin','$email_id','$phone_number','$card_no','$expiry_date','$card_type','$fax_number')",
 $db) or ($error1=mysql_errno());
$num = mysql_affected_rows();
if ($num < 1)
$message="Agent ".$agentid." already exist.";
else
{
   $user=$user_id;
   session_save_path("/tmp");
   session_start();
   $sesid=session_id();
   session_register("sesid");
   session_register("order_list");
   session_register("user");
   $message = "You have registered successfully.";
}
include("tophtml.php");
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td>
<table width=100%>
<tr>
<td align="center">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td valign="top" align="center">
<font face="Verdana" size="2" color="#FF0000"><b><?php echo "$message"
```

```
?></b></font>
</td>
</tr>
<?php
if ($error1)
{
   print "
   <tr>
   <td valign=\"top\" align=\"center\">
   <form name=\"form1\" action=\"new.php\" method=\"post\">
   <input type=\"hidden\" name=\"name\" value=\"$name\">
   <input type=\"hidden\" name=\"user_id\" value=\"$user_id\">
   <input type=\"hidden\" name=\"address_line1\" value=\"$address_line1\">
   <input type=\"hidden\" name=\"address_line2\" value=\"$address_line2\">
   <input type=\"hidden\" name=\"city\" value=\"$city\">
   <input type=\"hidden\" name=\"country\" value=\"$country\">
   <input type=\"hidden\" name=\"pin\" value=\"$pin\"><input type=\"hidden\"
   name=\"email_id\" value=\"$email_id\">
   <input type=\"hidden\" name=\"phone_number\" value=\"$phone_number\">
   <input type=\"hidden\" name=\"fax_number\" value=\"$fax_number\">
   <input type=\"hidden\" name=\"card_no\" value=\"$card_no\">
   <input type=\"hidden\" name=\"expiry_date\" value=\"$expiry_date\">
   <input type=\"hidden\" name=\"card_type\" value=\"$card_type\">
   <input type=\"submit\" value=\"Try Again\">
   </td>
   </tr>";
}
?>
</table>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing inserts a new record in the user_profile database table. If the new end user is registered successfully, the message, You have registered successfully, is displayed to the end user; else, an error message is displayed.

Viewing Books Stored in a Category

When the end user clicks the BOOK LIST button after selecting a book category from the Select Book Category combo box, as shown in Figure 9-14, the book_desc.php file is processed. This file displays all the books present in the specified category.

Listing 9-27 shows the content of the book_desc.php file:

Listing 9-27: Displaying Books in a Specified Category

```
<?php
session_save_path("/tmp");
session_start();
include("tophtml.php");
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
```

```
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$req_from=$_GET['req_from'];
$title=$_GET['title'];
$search_by=$_GET['search_by'];
$search_word=$_GET['search_word'];
$item_type=$_GET['item_type'];
if ($req_from == "search")
{
   if ($search_by == "author")
   {
      $result1=mysql_query("select * from book where author like '%$search_word%'", $db);
   }
   elseif ($search_by == "title")
   {
      $result1=mysql_query("select * from book where title like '%$search_word%'", $db);
   }
}
else
{
   $result1=mysql_query("select * from book where item_type='$item_type'", $db);
}
$num = mysql_num_rows($result1);
if ($num > 0)
{
   while($row1=mysql_fetch_array($result1))
   {
      print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
      face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[0]</b></font></td><td
      align=\"center\"
      width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[1]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[2]</b></font></td><td
      align=\"center\"
      width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[3]</b></font></td><td
      align=\"center\"
      width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[4]</b></font></td><td
      align=\"center\"
      width=\"20%\"><form name=\"cart\" action=\"add_cart.php\"
      action=\"get\"><input
      type=\"hidden\" name=\"item_type\" value=\"$item_type\"><input type=\"hidden\"
      name=\"item_no\" value=\"$row1[0]\"><input type=\"hidden\" name=\"req_from\"
      value=\"$req_from\"><input type=\"hidden\" name=\"search_by\"
      value=\"$search_by\"><input type=\"hidden\" name=\"search_word\"
      value=\"$search_word\"><input type=\"submit\" value=\"ADD TO
      CART\"></form></td></tr>";
   }
}
else
{
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"
   colspan=\"5\"><font face=\"verdana\" size=1 color=\"#FF0000\"><b>No match for
   the $search_word found.</b></font></td></tr>";
}
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
```

The above listing retrieves the information about books for a specific category.

Figure 9-16 shows the Web page that displays the books present in the Database category:

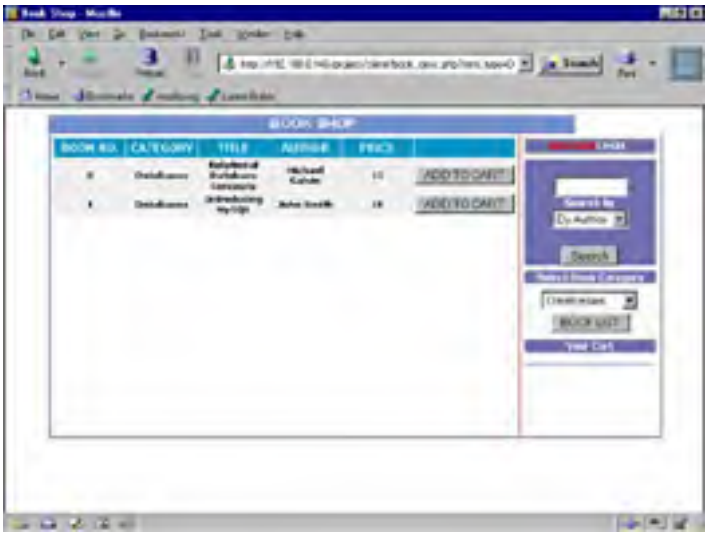


Figure 9-16: Books in the Database Category

Placing a Book in the Shopping Cart

When the end user clicks the ADD TO CART button, shown in Figure 9-16, the add_cart.php file is processed to add the selected book to the cart.

Listing 9-28 shows the content of the add_cart.php file:

Listing 9-28: The add_cart.php File

```
<?php
//include "sessioncheck.php";
session_start();
include("tophtml.php");
$date = date('Y-m-d');
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$search_by=$_GET['search_by'];
$search_word=$_GET['search_word'];
$item_type=$_GET['item_type'];
$item_no=$_GET['item_no'];
if ($user)
{
   if (!$resulr=mysql_query("insert into tmp
   values('NULL','$user','$item_no','$sesid','$date')", $db))
   {
      $e=mysql_error();
      echo "$e";
   }
}
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
```

```
color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
if ($req_from == "search")
{
   if ($search_by == "author")
   {
      $result1=mysql_query("select * from book where author like '%$search_word%'", $db);
   }
   elseif ($search_by == "title")
   {
      $result1=mysql_query("select * from book where title like '%$search_word%'", $db);
   }
}
else
{
   $result1=mysql_query("select * from book where item_type='$item_type'", $db);
}
while($row1=mysql_fetch_array($result1))
{
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
   face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[0]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[1]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[2]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[3]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[4]</b></font></td><td
   align=\"center\" width=\"20%\"><form name=\"cart\" action=\"add_cart.php\"
   action=\"post\"><input type=\"hidden\" name=\"item_type\"
   value=\"$item_type\"><input type=\"hidden\" name=\"item_no\"
   value=\"$row1[0]\"><input type=\"hidden\" name=\"req_from\"
   value=\"$req_from\"><input type=\"hidden\" name=\"search_by\"
   value=\"$search_by\"><input type=\"hidden\" name=\"search_word\"
   value=\"$search_word\"><input type=\"submit\" value=\"ADD TO
   CART\"></form></td></tr>";
}
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing adds a book to the shopping cart.

Figure 9-17 shows the output when the book, Relational Database Concepts, is added to the cart:

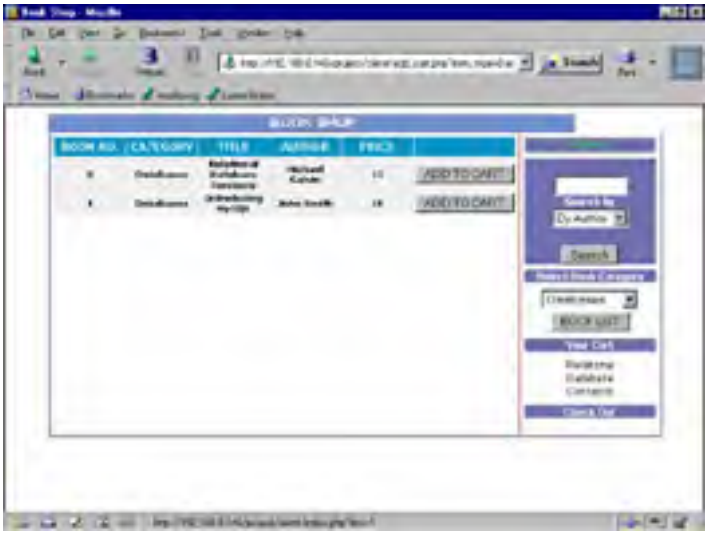


Figure 9-17: Adding a Book to the Cart

Searching for a Book

The end user can search for a book either by its title or the author name. To search a book by title, enter the title in the Search By text field, as shown in Figure 9-17, and click the Search button. The book_desc.php file is executed to retrieve the books from the database that match the specified title.

Listing 9-29 shows the content of the book_desc.php file:

Listing 9-29: Retrieving Books matching a Specified Title

```
<?php
session_save_path("/tmp");
session_start();
include("tophtml.php");
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$req_from=$_GET['req_from'];
$title=$_GET['title'];
$search_by=$_GET['search_by'];
$search_word=$_GET['search_word'];
$item_type=$_GET['item_type'];
if ($req_from == "search")
{
   if ($search_by == "author")
   {
      $result1=mysql_query("select * from book where author like '%$search_word%'", $db);
   }
   elseif ($search_by == "title")
   {
      $result1=mysql_query("select * from book where title like '%$search_word%'", $db);
```

```
    }
}
else
{
   $result1=mysql_query("select * from book where item_type='$item_type'", $db);
}
$num = mysql_num_rows($result1);
if ($num > 0)
{
   while($row1=mysql_fetch_array($result1))
   {
      print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
      face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[0]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[1]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[2]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[3]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[4]</b></font></td><td
      align=\"center\" width=\"20%\"><form name=\"cart\" action=\"add_cart.php\"
      action=\"get\"><input type=\"hidden\" name=\"item_type\"
      value=\"$item_type\"><input type=\"hidden\" name=\"item_no\"
      value=\"$row1[0]\"><input type=\"hidden\" name=\"req_from\"
      value=\"$req_from\"><input type=\"hidden\" name=\"search_by\"
      value=\"$search_by\"><input type=\"hidden\" name=\"search_word\"
      value=\"$search_word\"><input type=\"submit\" value=\"ADD TO
      CART\"></form></td></tr>";
   }
}
else
{
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"
   colspan=\"5\"><font face=\"verdana\" size=1 color=\"#FF0000\"><b>No match for
   the $search_word found.</b></font></td></tr>";
}
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

Figure 9-18 shows the output when search is performed on the basis of the author name, Galvin:

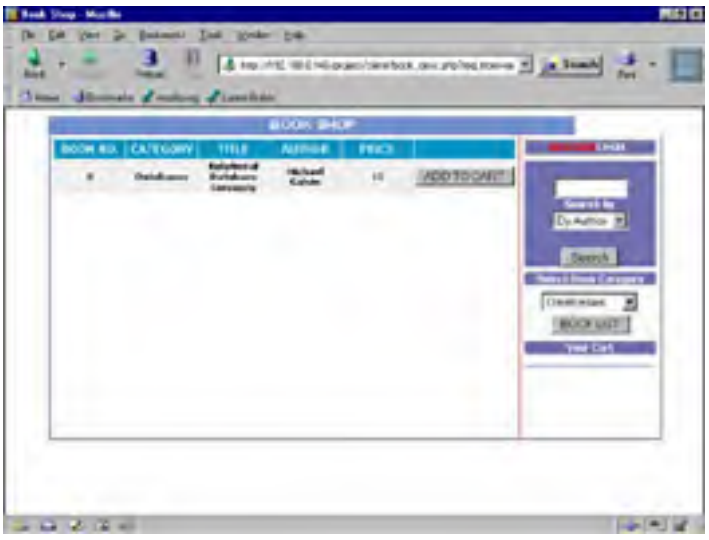


Figure 9-18: Searching Books Based on Author Name

Viewing Items Placed in a Shopping Cart

When the end user clicks the Check Out hyperlink present in the right section of the welcome.php Web page, the checkout.php file is executed. The checkout.php Web page creates a Web page that shows the books added to the shopping cart, as shown in Listing 9-30:

Listing 9-30: The checkout.php File

```
<?php
session_start();
include("tophtml.php");
$date = date('Y-m-d');
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>SNO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$sn=0;
$total=0;
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result = mysql_query("select book.*,tmp.order_no from tmp,book where tmp.user_id='$user' and
tmp.sesid='$sesid' and tmp.date='$date' and book.item_no=tmp.item_no", $db);
while($row_chack=mysql_fetch_array($result))
{
   $total=$total+$row_chack[4];
   $sn++;
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
   face=\"verdana\" size=1
   color=\"#000000\"><b>$sn</b></font></td><td align=\"center\"
   width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[0]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[1]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[2]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[3]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[4]</b></font></td><td
   align=\"center\" width=\"20%\"><form name=\"cart\" action=\"del_cart.php\"
   action=\"post\"><input type=\"submit\" value=\"DELETE\"><input type=\"hidden\"
   name=\"order_no\" value='$row_chack[5]'></form></td></tr>";
}
print "<tr bgcolor=\"#F2F2F2\"><td align=\"right\" width=\"100%\"
colspan=\"2\"><form name=\"order\" action=\"final_order.php\" method=\"post\"><input
type=\"submit\" value=\"POST YOUR ORDER\"></td><td align=\"right\" width=\"100%\"
colspan=\"2\"><font face=\"verdana\" size=1 color=\"#000000\"><b>TOTAL BOOKS:
  $sn</b></font></td><td align=\"right\" width=\"100%\"
```

```
colspan=\"2\"><font face=\"verdana\" size=1 color=\"#000000\"><b>TOTAL :
  </b></font></td><td align=\"left\" width=\"100%\"><font
face=\"verdana\" size=1
color=\"#000000\"><b>$total</b></font></td></td></tr>";
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing displays the titles of the books that are added to the shopping cart in a Web page, as shown in Figure 9-19 :

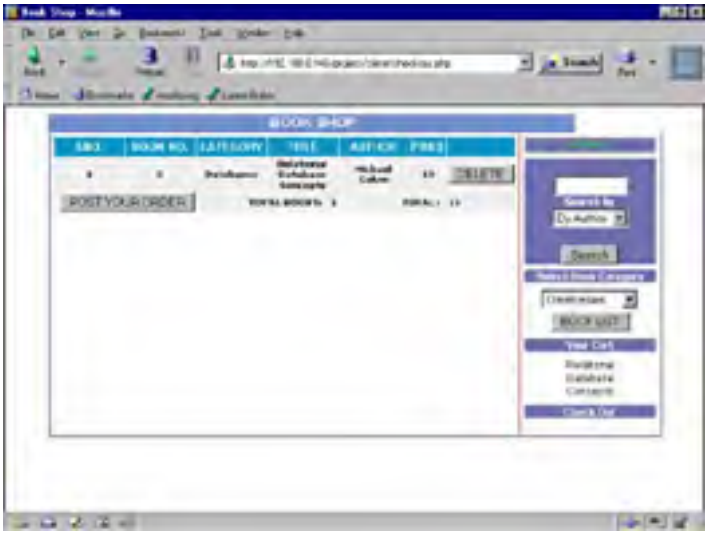


Figure 9-19: Displaying Books Added to the Shopping Cart

Removing a Book From the Shopping Cart

When the end user clicks the Delete button corresponding to a book name, as shown in Figure 9-19, the del_cart.php file is executed to remove the selected book from the shopping cart.

Listing 9-31 shows the content of the del_cart.php file:

Listing 9-31: The del_cart.php File

```
<?php
session_start();
include("tophtml.php");
$date = date('Y-m-d');
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$order_no=$_GET["order_no"];
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$resul_del=mysql_query("delete from tmp where order_no=$order_no", $db);
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
```

```
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>SNO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$sn=0;
$total=0;
$result_chack = mysql_query("select book.*,tmp.order_no from tmp,book where tmp.user_id='$user'
and tmp.sesid='$sesid' and tmp.date='$date' and book.item_no=tmp.item_no", $db);
while($row_chack=mysql_fetch_array($result_chack))
{
   $sn++;
   $total=$total+$row_chack[4];
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
   face=\"verdana\" size=1
   color=\"#000000\"><b>$sn</b></font></td><td align=\"center\"
   width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[0]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[1]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[2]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[3]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[4]</b></font></td><td
   align=\"center\" width=\"20%\"><form name=\"cart\" action=\"del_cart.php\"
   action=\"post\"><input type=\"submit\" value=\"DELETE\"><input type=\"hidden\"
   name=\"order_no\" value='$row_chack[5]'></form></td></tr>";
}
print "<tr bgcolor=\"#F2F2F2\"><td align=\"right\" width=\"100%\"
colspan=\"2\"><form name=\"order\" action=\"final_order.php\"
method=\"post\"><input type=\"submit\" value=\"POST YOUR ORDER\"></td><td
align=\"right\" width=\"100%\" colspan=\"2\"><font face=\"verdana\" size=1
color=\"#000000\"><b>TOTAL BOOKS:   $j</b></font></td><td
align=\"right\" width=\"100%\" colspan=\"2\"><font face=\"verdana\" size=1
color=\"#000000\"><b>TOTAL :   </b></font></td><td
align=\"left\" width=\"100%\"><font face=\"verdana\" size=1
color=\"#000000\"><b>$total</b></font></td></td></tr>";
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing removes the selected book from the shopping cart.

Confirming an Order

When the end user clicks the POST YOUR ORDER button, as shown in Figure 9-19, the final_order.php file is executed to confirm the order.

Listing 9-32 shows the content of the final_order.php file:

Listing 9-32: The final_order.php File

```
<?php
session_start();
include("tophtml.php");
$date = date('Y-m-d');
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$order_no=$_GET["order_no"];
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_con=mysql_query("select user_id,item_no,date from tmp where sesid='$sesid' and
user_id='$user' and date='$date'", $db);
$tt=mysql_num_rows($result_con);
if ($tt > 0)
{
   $result_ins=mysql_query("insert into transaction(order_no,user_id,date,status)
   values('NULL','$user','$date','Pending')", $db);
   $result_order=mysql_query("select max(order_no) from transaction where user_id='$user' and
   date='$date'", $db);
   $row_no = mysql_fetch_array($result_order);
   $order_no = $row_no[0];
   while($row_con=mysql_fetch_array($result_con))
   {
      $result11 = mysql_query("insert into order1
      values('NULL','$order_no','$row_con[1]','$row_con[0]')", $db);
   }
   $result_del=mysql_query("delete from tmp where sesid='$sesid' and user_id='$user' and
   date='$date'", $db);
}
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="100%"><font face="Arial" size=2 color="#000000"><b>
<?php
if ($order_no)
echo "Your order has been posted successfully. Your Order No. is $order_no";
else
echo "Please select books for order.";
?>
</b></font></td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing confirms the order for the books added to the shopping cart.

Figure 9-20 shows the Web page that displays the order confirmation message to the end user:

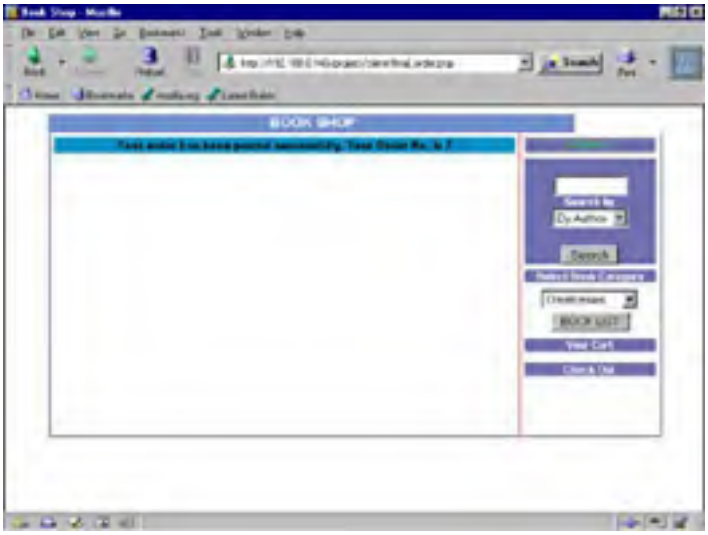


Figure 9-20: Web Page to Confirm the Order

Logging Off from the Shopping Cart Application

When the end user clicks the LOGOUT hyperlink present in the right section of welcome.php Web page, the logout.php file is executed, which logs the end user out of the application.

Listing 9-33 shows the content of the logout.php file:

Listing 9-33: The logout.php File for the End User

```
<?php
session_save_path("/tmp");
session_start();
$sesid=session_id();
session_unregister("sesid");
session_unregister("order_list");
session_unregister("user");
session_destroy();
$sesid="";
include("tophtml.php");
?>
<tr>
<td valign="top">
<table border="0" width="600" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif"
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td>
<table width=100%>
<tr>
<td align="center"><font face="arial" size=3 color="#FF0000"><b>
Logout successfully.
 </b></font></td>
</tr>
<?php
include "bottumhtml.php";
?>
```

The above listing logs out the end user from the online shopping cart application.

PREVIOUS NEXT

Appendix A: XML and XSLT Functions

This appendix describes various XML and XSLT functions.

XML Functions

Extensible Markup Language (XML) is a standard defined by the World Wide Web consortium (W3C) to represent data in a structured format.

You can use the following functions in Hypertext Preprocessor (PHP) to parse the XML documents:

- `utf8_decode()`: Transforms a string encoded in the UTF-8 format to the single-byte ISO-8859-1encoding format. The syntax of the utf8_decode() function is:

```
string utf8_decode (string data)
```

- `utf8_encode()`: Encodes a string with ISO-8859-1 encoding format to the UTF-8 format. The syntax of the utf8_encode() function is:

```
string utf8_encode (string data)
```

- `xml_error_string()`: Returns an XML parser string with a description of the error code, or returns the value, False, if no description is found. The syntax of the xml_error_string() function is:

```
string xml_error_string (int error_code)
```

- `xml_get_current_byte_index()`: Returns the current index of an XML parser. The syntax of the xml_get_current_byte_index() function is:

```
int xml_get_current_byte_index (parser)
```

In the above syntax, the xml_get_current_byte_index() function returns the value, False, if the specified parser is not a valid parser; or else, returns the current index of an XML parser.

- `xml_get_current_column_number()`: Returns the column number the parser is currently at. The syntax of the xml_get_current_column_number() function is:

```
int xml_get_current_column_number (parser)
```

- `xml_get_current_line_number()`: Returns the value of current line number the parser is currently at, in its data buffer. The syntax of the xml_get_current_line_number() function is:

```
int xml_get_current_line_number (parser)
```

- `xml_parse_into_struct()`: Parses the XML data into an array structure. The syntax of the xml_parse_into_struct() function is:

```
int xml_parse_into_struct (parser, string data, array &values [, array &index])
```

- `xml_parse()`: Parses an XML document. The syntax of the xml_parse() function is:

```
bool xml_parse (parserName, string data [, bool final])
```

- `xml_parser_create_ns()`: Creates a XML parser with XML namespace support. The syntax of the xml_parser_create_ns() function is:

```
resource xml_parser_create_ns ([string encoding [, string separator]])
```

- `xml_parser_create()`: Creates an XML parser. The syntax of the xml_parser_create() function is:

```
resource xml_parser_create ([string encoding])
```

- `xml_parser_free()`: Frees an XML parser and returns the value, False, if the parser is invalid; else, frees the specified parser and returns the value, True. The syntax of the xml_parser_free() function is:

```
bool xml_parser_free (resource parser)
```

- `xml_parser_get_option()`: Returns the value, False, if the specified parser is not a valid parser; else, returns the value of the option that is passed as argument to the xml_parser_get_option() function. The syntax of the xml_parser_get_option() function is:

```
mixed xml_parser_get_option (resource parser, int option)
```

- `xml_set_character_data_handler()`: Sets the character data handler for the specified XML parser. The syntax of the xml_set_character_data_handler() function is:

```
bool xml_set_character_data_handler (resource parser, callback handler)
```

- `xml_set_default_handler()`: Sets the default handler for the specified parser. The syntax of the xml_set_default_handler() is:

```
bool xml_set_default_handler (resource parser, callback handler)
```

- `xml_set_object()`: Lets you use the XML parser within an object. The syntax of the xml_set_object() function is:

```
void xml_set_object (resource parser, object object)
```

- `xml_set_processing_instruction_handler()`: Sets the processing instruction handler for the specified parser. The syntax of the xml_set_processing_instruction_handler() function is:

```
bool xml_set_processing_instruction_handler (resource parser, callback handler)
```

- `xml_set_start_namespace_decl_handler()`: Sets the start namespace declaration handler for the specified parser. The syntax of the xml_set_start_namespace_decl_handler() function is:

```
bool xml_set_start_namespace_decl_handler (resource parser, callback handler)
```

- `xml_set_unparsed_entity_decl_handler()`: Sets the unparsed entity declaration handler for the specified parser. The syntax of the xml_set_unparsed_entity_decl_handler() function is:

```
bool xml_set_unparsed_entity_decl_handler (resource parser, callback handler)
```

XSLT Functions

You use eXtensible Stylesheet Language Transformations (XSLT) to restructure or transform XML data into various formats, such as HTML. You can format XML data using PHP, in accordance with the template provided by XSLT.

The PHP extension provides a processor-independent API to perform XSLT transformations, and provides an interface to various XSLT processors, such as Sablotron and Xalan. PHP 4.1.1, the XSLT extension, supports only the Sablotron processor.

You can download Sablotron processor from http://www.gingerall.com/

The functions provided by the PHP extension to perform XSLT transformations are:

- `xslt_backend_info()`: Returns a string that contains information about the backend compilation settings. If no backend information is available, the xslt_backend_info() function returns an error string. The syntax of the xslt_backend_info() function is:

```
string xslt_backend_info (void)
```

- `xslt_backend_name()`: Returns the name of the backend processor. The syntax of the xslt_backend_name() function is:

```
string xslt_backend_name (void)
```

- `xslt_backend_version()`: Returns the version number of Sablotron. The syntax of the xslt_backend_version() function is:

```
string xslt_backend_version (void)
```

- `xslt_create()`: Creates a new XSLT processor. The syntax of the xslt_create() function is:

```
resource xslt_create (void)
```

- `xslt_errno()`: Returns an error number. The syntax of the xslt_errno() function is:

```
int xslt_errno (resource)
```

- `xslt_error()`: Returns an error message that describes the error generated while processing the XSLT processor. The syntax of the xslt_error() function is:

```
mixed xslt_error (resource)
```

- `xslt_process()`: Performs the XSLT transformation. The syntax of the xslt_process() function is:

```
mixed  xslt_process (resource , string xml_container, string xsl_container [, string
result_container [, array arguments [, array parameters]]])
```

In the above syntax, the containers refer to the filename containing the document to be processed. The result _container string refers to the filename for a transformed document.

- `xslt_set_base()`: Sets the base Uniform Resource Identifier (URI) for XSLT transformations. The syntax of the xslt_set_base() function is:

```
void xslt_set_base (resource x, string uri)
```

- `xslt_set_encoding()`: Sets the encoding for the parsing of XML documents. The xslt_set_encoding() function is used only when you compile the Sablotron processor as backend, with encoding support. The syntax of the xslt_set_encoding() function is:

```
void xslt_set_encoding (resource x, string encoding)
```

- `xslt_set_error_handler()`: Sets an error handler function for an XSLT processor. The syntax of the xslt_set_errror_handler () function is:

```
void xslt_set_error_handler (resource x, mixed handler)
```

- `xslt_set_log()`: Creates a file that stores the log messages. The log messages store information about the status of the XSLT processor. The syntax of the xslt_set_log() function is:

```
void xslt_set_log (resource x, mixed logparam)
```

In the above syntax, x is a variable that refers to the XSLT parameter, and logparam is a parameter that toggles logging on and off. You need to call the xslt_set_log() function with a Boolean parameter to enable message logging, as logging is disabled by default.

- `xslt_set_sax_handler()`: Sets the SAX handlers for the XSLT processor. The syntax of the xslt_set_sax_handler() function is:

```
void xslt_set_sax_handler ( resource x, array handlers)
```

- `xslt_set_scheme_handler()`: Sets the scheme handlers for an XSLT processor. The syntax of the xslt_set_scheme_handler () function is:

```
void xslt_set_scheme_handlers ( resource processor, array handlers)
```

Appendix B: Introducing eZXML

The eZXML class is an eXtensible Markup Language (XML) Document Object Model (DOM) parser written in Hypertext Preprocessor (PHP) language. The eZXML class provides an alternative approach to DOM implementation in PHP and lets you represent XML document in the form of nested objects. You do not require external libraries to support the eZXML class, because it is compatible with the libXml library.

The eZXML class generates a DOM tree after parsing the XML document, and generates an error message on encountering an incorrect XML document. The eZXML class creates a series of objects containing information about the XML nodes. Each object contains standard properties that enable you to traverse the DOM tree and access its attributes. You can develop PHP applications using the standalone libraries of the eZXML package.

This appendix describes how to create DOM tree of an XML document and how to convert an XML document into HTML using eZXML.

Working with eZXML

eZXML parses XML documents and creates a DOM tree representation from the XML document. eZXML creates objects to represent the XML document. Each object encapsulates properties that help traverse the DOM tree and access its elements and attributes. The elements of the eZXML class are defined in the eZXML.php file.

You need to include the eZXML.php file in the PHP script to parse the XML documents. eZXML.php is a procedural file that is defined in the eZXML class. The eZXML.php file defines the following members:

- NamespaceStack: Contains the namespaces.
- CurrentNameSpace: Specifies the current namespace.
- DomDocument: Specifies a reference to the DOM document.
- & eZXML::domTree ($xmlDoc, $params = array()): Returns a DOM object tree from the XML document.
- &eZXML::parseAttribute($attibuteString): Parses the attributes of the DOM tree and returns the value, False, if the string passed as argument does not contain any attributes.

Creating the DOM Tree of an XML Document using eZXML

The eZXML class contains the domTree() method that accepts XML strings as arguments and returns the DOM tree representation of an XML document. The object, which the domTree() method returns, contains information corresponding to the nodes of the XML document. You can use the information stored in the objects to create, modify, and format the XML document.

Listing B-1 shows how to create employee.xml file:

Listing B-1: Creating the employee.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<EMPLOYEEINFORMATION>
<EMPLOYEE>
<NAME ID="E001">George</NAME>
<AGE>35</AGE>
<DEPARTMENT>RESEARCH AND DEVELOPMENT</DEPARTMENT>
<DESIGNATION>BRANCH MANAGER</DESIGNATION>
</EMPLOYEE>
<EMPLOYEE>
<NAME ID="E002">John</NAME>
<AGE>45</AGE>
<DEPARTMENT>HUMAN RESOURCE</DEPARTMENT>
<DESIGNATION>MANAGER</DESIGNATION>
</EMPLOYEE>
</EMPLOYEEINFORMATION>
```

The above listing creates the employee.xml file that stores the employee information, such as name, ID, department, and designation. You can create the DOM object tree representation of the employee.xml file using eZXML, as shown in Listing B-2:

Listing B-2: Creating Object Representation of the employee.xml File

```
<?php
// include class definition
include_once( "./ezpublish-3.4.0/lib/ezutils/classes/ezdebug.php" );
include_once( "./ezpublish-3.4.0/lib/eZXML/classes/ezdomnode.php" );
include_once( "./ezpublish-3.4.0/lib/eZXML/classes/ezdomdocument.php" );
include_once( "./ezpublish-3.4.0/lib/eZXML/classes/eZXML.php" );
// XML file
$xmlFile = "employee.xml";
// parse XML file into single string
$xmlString = join("", file($xmlFile));
// create Document object
$doc = eZXML::domTree($xmlString, array("TrimWhiteSpace"=> "true"));
print_r($doc);
?>
```

The above listing creates a DOM tree using the eZXML class. The eZXML.php file is included in the script to parse the well-formed XML document, employee.xml. In the above listing, the object created by eZXML contains the node objects, which store information corresponding to the XML nodes.

Converting an XML Document into HTML using eZXML

eZXML converts XML documents into the HTML format. For example, you can create a file called customer.xml and convert into an HTML format.

Listing B-3 shows how to create the customer.xml file:

Listing B-3: Creating the customer.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<Shopping>
<customerInfo>
<custName>John</custName>
<address>1600 Pennsylvania Ave. NW, Washington, DC 20500</address>
<date>2001-09-15</date>
</customerInfo>
<items>
<item custid="3225">
<product>Air Conditioners</product>
<price>16235.00</price>
<quantity>1</quantity>
<subtotal>262235.00</subtotal>
</item>
<item custid="3226">
<product>Dryers</product>
<price>129.99</price>
<quantity>4</quantity>
<subtotal>139.96</subtotal>
</item>
<item custid="2345">
<product>Vacuum Cleaners</product>
<price>4139.99</price>
<quantity>2</quantity>
<subtotal>4139.99</subtotal>
</item>
<item custid="4576">
<product>Washers></product>
<price>349.99</price>
<quantity>2</quantity>
<subtotal>349.99</subtotal>
</item>
</items>
</Shopping>
```

The above listing creates the customer.xml file that stores billing information, such as product name, billing date, quantity, and product price.

You can obtain the HTML format of the customer.xml file after running the PHP script in your Web browser.

Listing B-4 shows how to convert the customer.xml file into the HTML format:

Listing B-4: Converting customer.xml into HTML Format

```
<?php
// Include class definition
include("eZXML.php");
// Arrays to associate XML elements with HTML output
$startTags = array
(
   'CUSTOMER'=> '<p> <b>Customer: </b>',
   'ADDRESS'=> '<p> <b>address: </b>',
   'DATE'=> '<p> <b>date: </b>',
   'ITEMS'=> '<p> <b>Details: </b> <table width="100%" border="1"
   cellspacing="0"
   cellpadding="3"><tr><td><b>Item
   description</b></td><td><b>Price </b></td>
   <td><b>Quantity</b></td><td><b>Sub-total </b></
   td></tr>',
   'ITEM'=> '<tr>',
   'PRODUCT'=> '<td>',
   'PRICE'=> '<td>',
   'QUANTITY'=> '<td>',
   'SUBTOTAL'=> '<td>',
   'TERMS'=> '<p> <b>Rules and Regulation: </b> <ul>',
   'TERM'=> '<li>'
);
$endTags = array
(
   'LINE'=> ', ',
   'ITEMS'=> '</table>',
```

```
    'ITEM'=> '</tr>',
    'PRODUCT'=> '</td>',
    'PRICE'=> '</td>',
    'QUANTITY'=> '</td>',
    'SUBTOTAL'=> '</td>',
    'TERMS'=> '</ul>',
    'TERM'=> '</li>'
);
// XML file
$xmlfile = "customer.xml";
$xmlString = join("", file($xmlFile));
// parse XML string and create object
$doc = eZXML::domTree($xmlString, array("TrimWhiteSpace" => "true"));
// start printing
print ($doc->children);
// Accepts an array of nodes recursively as argument.
// Iterates through the XML document.
// Constructs HTML markups.
// Displays the XML contents.
function print ($NODE)
{
   global $startTags, $endTags, $Totals;
   for ($t=0; $t<sizeof($NODE); $t++)
    {
      // How to handle elements
      if ($NODE[$t]->type == 1)
      {
         // Displays the opening tags
         echo $startTags[strtoupper($NODE[$t]->name)];
         // recurse
         print ($NODE[$t]->children);
         // Displays the closing tags
         echo $endTags[strtoupper($NODE[$t]->name)];
      }
      // How to handle text nodes
       if ($NODE[$t]->type == 3)
       {
         // Printing the text
         echo($NODE[$t]->content);
       }
   }
}
?>
```

The above listing converts the customer.xml file into the HTML format using the eZXML class. In the above listing, the eZXML.php file is included in the script to parse and manipulate the customer.xml file. The eZXML.php file contains functions, which represent XML document in a DOM tree format by creating nested objects.

In the above listing, the associative arrays, $startTags and $endTags, store the HTML markup, if the node is of element type. If the node is of text type, the content of the text node is displayed in the Web browser.

PREVIOUS NEXT

Appendix C: The PHP.XPath Class

PHP.XPath is a Hypertext Preprocessor (PHP) class that searches and retrieves data from an eXtensible Markup Language (XML) document using XPath. The PHP.XPath class uses the Document Object Model (DOM) Application Programming Interface (API) to modify an XML document.

This appendix introduces the PHP.XPath class, and describes the types of objects in this class. It also explains the public and private methods of the objects in the PHP.XPath class.

Introducting the PHP.XPath Class

The PHP.XPath class accepts XML data as its argument, parses the data, and creates an object that represents XML data. This object uses the methods defined in the PHP.XPath class to create XML nodesets on the basis of XPath expressions. You can add, delete, or edit the nodes from an XML document tree, and retrieve specific information from within an XML document.

If the PHP module on your server is not compiled with the XML library, you can use the PHP.XPath class to provide XML functionality. The PHP.XPath class consists of objects of three base classes, which are:

- XpathBase: Contains debugging functions.
- XpathEngine: Contains XML import and export methods. The XpathEngine base class is made up of the XPath specification.
- Xpath: Contains methods that modify an XML document.

Note You need not install the DOM XML PHP library to modify an XML document using the PHP.XPath class.

The XPathBase Class

The XPathBase class consists of public and private methods. A public method declared in a class is accessible to other classes. The public methods in the XPathBase class are:

- `XPathBase() method`: Represents the constructor of the XPathBase class.
- `reset() method`: Resets the XPathBase class object and enables the object to accept new XML data.
- `setVerbose() method`: Alters the details of error level reporting in an XML document. The syntax of the setVerbose() method is:

```
setVerbose($level=1)
```

In the above syntax, the $level parameter is set to the default value, 1. To turn off the error-level reporting, set the value of the $level parameter to 0. If the $level parameter has an integer value greater than 1, error-level reporting is turned on.

- `getLastError() method`: Returns the recent error message as a string value. This method returns a null value if there is no error message.

Note The higher the value of the $level parameter, the higher is the level of error.

A private method is not accessible outside the class definition. The private methods in the XPathBase class are:

- `_searchString() method`: Searches a string within another string in an XML document. The syntax of the _searchString() method is:

```
_searchString($term, $expression)
```

In the above syntax, the $expression parameter denotes the string to be searched; and the $term parameter indicates the string in which the _searchString() method searches the specified expression.

The _searchString() method returns an integer value. It returns the offset value at which the string is found. The _searchString() method returns –1, if the $expression parameter is not a substring of the string in the $term parameter.

- `_bracketsCheck() method`: Checks if there is an ending bracket corresponding to each starting bracket in an XPath expression. The syntax of the _bracketsCheck() method is:

```
_bracketsCheck($term)
```

In the above syntax, the $term parameter denotes the XPath string, in which the _bracketsCheck() method checks for a matching bracket.

- `_prestr() method`: Retrieves the substring positioned before a delimiter. The syntax of the _prestr() method is:

```
_prestr($string, $delimiter, $offset=0)
```

In the above syntax, the $string parameter denotes the string from which the _prestr() method retrieves a substring. The $delimiter parameter denotes the string that contains the delimiter for the _prestr() method. If the $offset parameter is set to 0, the method searches the $string string, starting from the first character, until it finds the delimiter for the first instant.

If the $offset parameter is set to a value other than 0, the _prestr() method searches the $string string from the specified offset value. The _prestr() method returns a substring that precedes the specified delimiter in the original string. An example of the _prestr() method is:

```
_prestr(Park:Town, ':', $offset=0)
```

In the above code, the _prestr() method extracts the Park substring from the Park:Town string because the Park string precedes the : delimiter.

- `_afterstr() method`: Retrieves the substring positioned after a delimiter. The syntax of the _afterstr() method is:

```
_afterstr($string, $delimiter, $offset=0)
```

In the above syntax, the $string parameter denotes the string from which the _afterstr() method retrieves a substring. The $delimiter parameter denotes the string that contains the delimiter. The _afterstr() method returns a substring that is positioned after the delimiter in the original string.

An example of the _afterstr() method is:

```
_afterstr(Park:Town, ':', $offset=0)
```

In the above code, the _afterstr() method extracts the Town substring from the Park:Town string because the Town string is positioned after the : delimiter.

- `_setLastError() method`: Generates an error message in textual form, and sets the error message to a variable. The syntax of the _setLastError() method is:

```
_setLastError($msg='', $line='-', $file='-')
```

In the above syntax, the $msg parameter represents an error message, the $line parameter represents the line number that contains the error, and the $file parameter represents the XML file name that is checked for error.

- `_displayError() method`: Displays an error message. The syntax of the _displayError() method is:

```
_displayError($msg, $line='-', $file='-', $terminate=TRUE)
```

In the above syntax, the $msg parameter represents the error message that the method displays, the $line parameter represents the line number that contains the error, and the $file parameter represents the XML file name that is checked for error. If the $terminate parameter, in the above syntax, is set to TRUE, it indicates that the file execution should stop.

The return type of the $msg and $file parameters is string, the return type of the $line parameter is an integer, and the return type of the $terminate parameter is boolean.

- `_beginDebugExecution() method`: Starts the debugging of the specified function. The syntax of the _beginDebugExecution() method is:

```
_beginDebugExecution($function)
```

In the above syntax, the $function parameter represents the name of the function that the _beginDebugExecution() method starts debugging. The return type of the _beginDebugExecution() method is an array value.

- `_closeDebugExecution() method`: Stops the debugging of the specified function. The syntax of the _closeDebugExecution() method is:

```
_closeDebugExecution($time, $return="")
```

In the above syntax, the $time parameter represents the time when the debugging starts, and the $return parameter represents the value that the debug function returns.

- `_printContext() method`: Displays an XPath context. The syntax for the _printContext() method is:

```
_printContext($context)
```

In the above syntax, the $context parameter represents the XPath context.

The XPathEngine Class

The XPathEngine class consists of import and export methods. It also handles XPath queries. The public methods of the XPathEngine class are:

- `XPathEngine() method`: Represents the constructor of the XPathEngine class. This method accepts a single optional argument that specifies the XML filename that you need to parse.

- `getProperties() method`: Returns the specified properties of the XPathEngine class. The syntax of the getProperties() method is:

```
getProperties($par=NULL)
```

In the above syntax, the $par parameter represents the property of the XPathEngine class whose value is to be retrieved. If the $par parameter is set to NULL, the getProperties() method retrieves all the properties of the XPathEngine class.

- `getNode() method`: Retrieves the nodes from an XML document tree. An XPath expression specifies the nodes that you need to retrieve. The syntax of the getNode() method is:

```
getNode($abs_XPath='')
```

In the above syntax, the $abs_XPath parameter is a string that represents the absolute path to be traversed for retrieving the specified nodes. The getNode() method returns the value, FALSE, if the specified node does not exist in the document tree.

- `exportToFile() method`: Generates an XML string that contains the content of the current document, and writes the XML string to the specified file. The syntax of the exportToFile() method is:

```
exportToFile($file, $abs_XPath='', $header=NULL)
```

In the above syntax, the $abs_XPath parameter represents the address of the node that you need to export. The $header parameter represents the string that should appear before the content in an XML file. If you specify the $header parameter as NULL, the header string is assigned the value of the XML header in the parsed XML file.

- `importFromFile() method`: Reads a Uniform Resource Locator (URL) or a file, and parses the XML data. The syntax of the importFromFile() method is:

```
importFromFile($file)
```

In the above syntax, the $file parameter represents the name of the XML file that you need to parse.

- `match() method`: Evaluates an XPath query by parsing it. The syntax of the match() method is:

```
match($query, $base='')
```

In the above syntax, the $query parameter represents the XPath expression that you need to evaluate, and the $base parameter represents the path of a document node from where the query executes.

- `equalNodes() method`: Compares two nodes to check if the nodes represent the same node in a document tree. The syntax of the equalNodes() method is:

```
equalNodes($node1, $node2)
```

In the above syntax, the $node1 and $node2 parameters represent either the absolute location paths to a specific node, or the node itself. The equalNodes() method returns TRUE if the two nodes are equal and FALSE if the nodes are not equal.

- `hasChildNodes() method`: Checks if the specified node contains child nodes. The syntax of the hasChildNodes() method is:

```
hasChildNodes($abs_XPath)
```

In the above syntax, the $abs_Xpath parameter represents the absolute path of the parent node. The hasChildNodes() method returns TRUE if the parent node exists and contains child nodes. If a parent node does not contain any child nodes, the hasChildNodes() method returns FALSE.

The private methods of the XPathEngine class are:

- `_InternalExport() method`: Exports an XML document, starting from the specified node in an XML document. The syntax of the _InternalExport() method is:

```
_InternalExport($node)
```

In the above syntax, the $node parameter represents the node in the XML document from where you need to start exporting the document.

- `_handleStartElement() method`: Handles the opening tags while parsing an XML document. The syntax of the _handleStartElement() method is:

```
_handleStartElement($parser, $node, $attributes)
```

In the above syntax, the $parser parameter represents the handler that accesses the XML parser, the $node parameter represents the opening tag in an XML document, and the $attributes parameter represents an associative array that contains a list of attributes of the opening tag.

- `_handleEndElement() method`: Handles the closing tags while parsing an XML document. The syntax of the _handleEndElement() method is:

```
_handleEndElement($parser, $node)
```

In the above syntax, the $parser parameter represents the handler for accessing the XML parser, and the $node parameter represents the closing tag in an XML document.

- `_handleCharacterData() method`: Handles character data while parsing an XML document. The syntax of the _handleCharacterData() method is:

```
_handleCharacterData($parser, $text)
```

In the above syntax, the $parser parameter represents the handler for accessing the XML parser and the $text parameter represents the character data in an XML document.

- `_checkPredicates() method`: Checks if a node matches the specified list of predicates. The syntax of the _checkPredicates() method is:

```
_checkPredicates($nodesets, $predicates)
```

In the above syntax, the $nodesets parameter represents an array that contains the location paths of the nodes that you need to match against the specified list of predicates. The $predicates parameter represents an array of predicates.

- `_getAxis() method`: Retrieves the axis name from an XPath query. The syntax of the _getAxis() method is:

```
_getAxis($step, $context)
```

In the above syntax, the $step parameter represents a string that contains the XPath query, and the $context parameter represents the context from where the method starts evaluating an XML document. The _getAxis() method returns an array that contains the axis name, and its nodetest.

- `_handleAxis_Child() method`: Handles the child axis of a document tree. The syntax of the _handleAxis_Child() method is:

```
_handleAxis_Child($axis, $context)
```

In the above syntax, the $axis parameter is an array that contains information about the axis of a node, and the $context parameter represents the path of the node of the axis that is processed.

The XPath Class

The XPath class also includes public and private methods. The public methods of the XPath class are:

- `XPath() method`: Represents the constructor of the XPath class.
- `nodeName() method`: Retrieves the names of the nodes from a document tree, along with the path of the nodes specified by the method parameter. The syntax of the nodeName() method is:

```
nodeName($query)
```

In the above syntax, the $query parameter represents the path from where you need to retrieve the nodes.

- `appendChild() method`: Adds child nodes to the existing child nodes in a document tree. The syntax of the appendChild() method is:

```
appendChild($query, $node, $afterText=FALSE, $autoReindex=TRUE)
```

In the above syntax, the $query parameter represents the path of the child node to which you append another child node. The $node parameter is either a string or an array. If the $afterText parameter is set to FALSE, the node is inserted after the text. If the $autoReindex parameter is set to TRUE, the XML document is reindexed to reflect the changes in the document. The appendChild() method returns the path of the appended child node.

The private methods of the XPath class are:

- `_title() method`: Generates a title line. The syntax of the _title() method is:

```
_title($title)
```

In the above syntax, the $title parameter represents the title that you add to an XML document.

- _xml2Document() method: Parses an XML document to a tree node. The syntax of the _xml2Document() method is:

```
_xml2Document($str)
```

In the above syntax, the $str parameter represents the string that is converted to a document node.

◄ PREVIOUS NEXT ►

Appendix D: XML-RPC for PHP

eXtensible Markup Language-Remote Procedure Calls (XML-RPC) is an XML-based protocol that lets you make remote procedure calls over the Internet. XML-RPC is encoded in XML, and uses HTTP to transfer the client requests and receive the responses from a server. You can transfer complex data over the Internet using XML-RPC.

Using XML-RPC, you can integrate and develop a Hypertext Preprocessor (PHP) Web application. XML-RPC involves the process in which the server provides procedures to call the clients. The calling program transfers a message to the remote program, which executes the procedure specified in the message and returns the result to the calling program. The XML-RPC implementation provides various basic RPC functions, such as xmlrpc_decode_request() and xmlrpc_encode_request(), which enables you to make procedure calls in HTTPS. XML-RPC also provides utility functions for the conversion of the PHP and XML-RPC data types.

This appendix describes various XML-RPC and utility functions.

XML-RPC Functions

Using XML-RPC functions, you can create XML-RPC clients and servers. The XML-RPC functions are:

- `xmlrpc_decode_request()`: Decodes XML strings into native PHP types. The syntax of the xmlrpc_decode_request () function is:

  ```
  array xmlrpc_decode_request (string xml, string method [, string encoding])
  ```

- `xmlrpc_encode_request()`: Generates the XML language for a method request. The syntax of the xmlrpc_encode_request() function is:

  ```
  string xmlrpc_encode_request (string method, mixed params)
  ```

- `xmlrpc_get_type()`: Retrieves the xmlrpc data type for a PHP value. The syntax of the xmlrpc_encode_request() function is:

  ```
  string xmlrpc_get_type (mixed value)
  ```

- `xmlrpc_parse_method_descriptions()`: Decodes the XML language into a list of method descriptions. The syntax of the xmlrpc_parse_method_descriptions function is:

  ```
  array xmlrpc_parse_method_descriptions ( string xml)
  ```

- `xmlrpc_server_call_method()`: Parses the XML requests. The syntax of the xmlrpc_server_call_method() function is:

  ```
  mixed xmlrpc_server_call_method (resource server, string xml, mixed user_data [, array out
  ```

- `xmlrpc_server_create()`: Creates an xmlrpc server. The syntax of the xmlrpc_server_create() function is:

  ```
  resource xmlrpc_server_create (void)
  ```

- `xmlrpc_server_destroy()`: Destroys the server resources. The syntax of the xmlrpc_server_destroy() function is:

  ```
  void xmlrpc_server_destroy (resource server)
  ```

- `xmlrpc_server_register_method()`: Registers a PHP function to a resource method matching the methodspecified as argument in the xmlrpc_server_register_method () function. The syntax of the xmlrpc_server_register_method() function is:

  ```
  bool xmlrpc_server_register_method (resource server, string methodName, string function)
  ```

- `xmlrpc_set_type()`: Specifies the xmlrpc data type, or datetime, for a PHP string value. The syntax of the xmlrpc_set_type() function is:

  ```
  bool xmlrpc_set_type (string value, string type)
  ```

Utility Functions

In addition to basic RPC functions, XML-RPC provides various utility functions to convert PHP data into XML-RPC compatible data types. Utility functions also provide a set of APIs to support HTTPS transactions. The utility functions are:

- `XMLRPC_prepare`: Transforms data into the appropriate data structure to be serialized into an XML-RPC message. For example, if you pass an associative array as an argument to the XMLRPC_prepare() function, the XMLRPC_prepare() function serializes it into an XML-RPC struct. The syntax of the XMLRPC_prepare() function is:

  ```
  XMLRPC_prepare($data, $type = NULL)
  ```

- `XMLRPC_request`: Connects to the site specified as argument to the XMLRPC_request() function, at the specified location, calls the method, and passes the parameters to the method. The syntax of the XMLRPC_request() function is:

  ```
  XMLRPC_request($siteName, $location, $methodName, $param = NULL, $useragent = NULL)
  ```

- `XMLRPC_response()`: Sends back an XML-RPC response to the server. The syntax of the XMLRPC_response() function is:

  ```
  XMLRPC_response($return_value, $server = NULL)
  ```

- `XMLRPC_error()`: Sends an XML-RPC error message. The syntax of the XMLRPC_error() function is:

  ```
  XMLRPC_error($fault_Code, $fault_String, $server = NULL)
  ```

- `XMLRPC_convert_timestamp_to_iso8601()`: Lets you set the formatted time string required by XML-RPC. The syntax of the XMLRPC_convert_timestamp_to_iso8601() is function

  ```
  XMLRPC_convert_timestamp_to_iso8601($timestamp)
  ```

- `count_numeric_items()`: Returns the number of numeric indices of an array. The syntax of the count_numeric_items() is function is:

  ```
  count_numeric_items($array)
  ```

- `XML_serialize()`: Accepts a data structure, and serializes it into XML format. The syntax of the XML_serialize() function is:

  ```
  & XML_serialize($data)
  ```

- `XMLRPC_debug_print`: Displays a table of debug messages and erases the debugging logs to ensure the error messages are not repeated. The syntax of the XML_serialize() function is:

  ```
  function XMLRPC_debug_print()
  ```

- `XMLRPC_debug()`: Logs the debugging messages. The syntax of the XMLRPC_debug() function is:

  ```
  XMLRPC_debug($function_name, $debug_message)
  ```

- `XMLRPC_adjustValue()`: Converts an XML-RPC data structure into a native PHP data structure. The syntax of the XMLRPC_adjustValue() function is:

  ```
  & XMLRPC_adjustValue(&$current_node)
  ```

Appendix E: Implementing SOAP using SOAPx4

Download CD Content

Simple Object Access Protocol (SOAP) is a W3C standard for exchanging information between various applications developed in different programming languages and running on various operating systems in a network. SOAPx4 is the PHP implementation of SOAP. SOAPx4 implements SOAP using the PHP server and client classes. SOAPx4 implements Web Services Description Language (WSDL) in Hypertext Pre-Processor (PHP).

This appendix explains how to implement SOAP requests and responses using SOAPx4. The appendix also explains how to create the SOAP server and SOAP client to process the requests of the clients in a distributed network.

Connecting a SOAP Client to a SOAP Server

A server accepts client requests in the form of SOAP request, and responds to the client in the form of SOAP response. The SOAP messages are transmitted in a distributed network in the form of header and body sections consisting of multiple soapval objects. You use the soapval object to specify a SOAP value. The syntax to use the soapval object is:

```
$var=new soapval(name, data_type, data)
```

The above syntax shows that the constructor of the soapval object accepts three parameters: name, data_type, and data. The name parameter indicates the name of the element, and the data_type parameter indicates the type of data to be stored in the element. The data parameter indicates the value of the element. The variable, $var, contains information stored in the soapval object.

The soapval and soapmsg objects create SOAP requests and SOAP responses. The soapmsg object of SOAPx4 contains a serialized message that is sent to the client and server of SOAP-based systems. The syntax to use the soapmsg object is:

```
$var=new soapmsg(procedure_name, array of soapval objects)
```

The above syntax shows that the constructor of the soapmsg object accepts two parameters, procedure_name and array of the soapval objects. The procedure_name parameter indicates the name of the procedure that is requested by the client, and the second parameter indicates the information to be sent to server. The $var variable indicates the complete SOAP message that is transmitted from the client to the server.

Creating a SOAP Request

You need to create a SOAP request that invokes the procedure of the SOAP server. In SOAPx4, you need to include the class.soap_client.php class to initiate the SOAP requests and invoke the required procedure.

Listing E-1 shows how to create a SOAP request using the soapval objects:

Listing E-1: Creating a SOAP Request

```
<?php
//Include the class definition of the php file that you can use to implement SOAP.
include("class.soap_client.php");
//Initialize the constructor of the soapval object that accepts three parameters.
$object=new soapval("student","string","John Williams");
//Create an object of the soapmsg object that contains data of the soapval object.
$message=new soapmsg("getStudentData", array($object));
//Create a SOAP packet that contains complete message to be transmitted using the serialize() functi
print $message->serialize();
?>
```

The above listing shows that the SOAP packet is a well-formed XML document, because it contains the envelope that indicates the encoding scheme used by the SOAP packets. The listing shows the envelope body that indicates the information to be sent.

The above listing shows that the class.soap_client.php file class is included in the SOAP request class to implement SOAP in PHP. The $object variable contains information that is passed to the procedure stored in the server class. The $message variable contains the complete SOAP information that is transmitted to the server.

> **Note** You can use the serializeval() function to serialize data into XML form. The command to display the serialized value of the soapval object is:
>
> ```
> print $object->serializeval();
> ```

The serialization process of SOAP creates SOAP packets in the XML form, which contains request that is transmitted from the client to the server.

Listing E-2 shows a generated SOAP packet:

Listing E-2: The Generated SOAP Packet

```
<xml version="1.0"?>
<SOAP-ENV: Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="httpd://www.w3.org/2001/XMLSchema"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:si=http://soapinterop.org/xsd
xmlns:ns6=http://testuri.org SOAP-ENV:
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
>
<SOAP-ENV:Body>
<ns6:getStudentData>
<student xsi:type="xsd:string">John Williams</student>
</ns6:getStudentData>
</SOAP-ENV-Body>
</SOAP-ENV:Envelope>
```

The above listing shows that the SOAP packet is a well-formed XML document, because it contains the envelope that indicates the encoding scheme used by the SOAP packets. The listing shows the envelope body that indicates the information to be sent.

Creating a SOAP Response

You need to create a SOAP response that transmits message to the SOAP request.

Listing E-3 shows how to create a SOAP response:

Listing E-3: Creating a SOAP Response

```
<?php
include("class_soap_client.php");
$object=new soapval("data","string","Age:15, Standard: 10th");
$message=new soapmsg("getStudentData", array($object));
print $msg->serialize();
?>
```

The above listing shows that the soapval object contains information stored in the getStudentData procedure. The $message variable contains the complete SOAP packet that is transmitted from the SOAP response to the SOAP request.

You can also serialize the response, sent to the client, using the serialize() function.

Listing E-4 shows a generated response in the form of SOAP packets:

Listing E-4: The Generated SOAP Response

```
<xml version="1.0"?>
<SOAP-ENV: Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="httpd://www.w3.org/2001/XMLSchema"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:si=http://soapinterop.org/xsd
xmlns:ns6=http://testuri.org SOAP-ENV:
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
>
<SOAP-ENV:Body>
<ns6:getStudentData>
<data xsi:type="xsd:string">Age:15, Standard: 10th</data>
</ns6:getStudentData>
</SOAP-ENV-Body>
</SOAP-ENV:Envelope>
```

The above listing shows that the SOAP packet is a well-formed XML document, because it contains the envelope that indicates the encoding scheme to be used by SOAP packets. The listing also shows the envelope body that indicates the information to be sent as the SOAP response to the SOAP request.

Creating a SOAP Server

You create a server class in PHP to implement the SOAP responses. PHP provides the server class to implement SOAP response in SOAPx4. You need to include the class.soap_server.php and class.soap_client.php classes to initiate a SOAP server in SOAPx4. The add_to_map() and service() functions register the required procedures with the newly created server, and send the responses to the clients.

The add_to_map() function adds the requested procedure to the list of procedures stored in the server. This function accepts three parameters: procedure_name, array_of_parameters, and return_type. The procedure_name parameter specifies the name of the procedure that is to be registered with the server. The second parameter specifies the array of the parameters passed to the procedure. The return_type parameter specifies the type of data returned by the procedure. The syntax to use the add_to_map() function is:

```
$server_object->add_to_map("procedure_name", "array_of_parameters", "return_type");
```

In the above syntax, the $server_object variable indicates the reference of the server object, which invokes the add_to_map() function to register the procedure.

The service() function processes the SOAP requests and returns the SOAP packets as a response to the client.

Listing E-5 shows how to create a SOAP server:

Listing E-5: Creating a SOAP Server

```
<?php
include("class.soap_client.php");
include("class.soap_server.php");
// Initialize a SOAP server.
$server=new soap_server;
$server->add_to_map("retrievePOP3Messages", array("SOAPStruct"), array("int"):
$server->service($HTTP_RAW_POST_DATA);
// The retrievePOP3Message function is a user-defined function that returns the number of
messages stored in the mailbox.
function retrievePOP3Messages($structure)
{
   $box=imap_open("{".$structure["pop_host"]."/pop3:110}", $structure["pop_user"],
   $structure["pop_pass"]);
   //If the connection is successfully established.
   if($box)
   {
    //Retrieve the number of messages.
    $totalMessage=imap_num_msg($box);
    imap_close($box);
    return $totalMessage;
   }
   else
   {
      // Generate an error.
      $parameter=array("errorCode"=>"75", "errorString"=>"Connection Not Established",
      "detail"=>"Not able to connect POP3 Server");
      $errMsg=new soapmsg("Errror", $parameter, http://schemas.xmlsoap.org/soap/envelope/
      );
   }
}
?>
```

In the above listing, the SOAP server establishes a connection with the mail server, POP3, and returns the number of messages stored in the mailbox. In the above code:

- The retrievePOP3Messages function is a user-defined function that takes the parameters passed by the clients.
- The imap_open() function establishes a connection with the POP3 mail server by passing three parameters: user name, password, and the host name of the mail server.
- The reference of the mail server is stored in the $box variable.
- The imap_num_msg() function retrieves the number of messages stored in the mailbox if the connection is established with the mail server.
- The imap_num_msg() function returns the value of the $totalMessage variable that contains the number of messages stored in the mailbox.
- The retrievePOP3Message() function generates an error if the connection with the mail server is not established.

Creating a SOAP Client

You need to include the class.soap_client.php class to initiate a SOAP client in SOAPx4. The SOAP client generates a SOAP request, and translates the SOAP packet returned as a response by the requested procedure.

The soapclient object lets you initiate a SOAP client with SOAPx4. The constructor of the soapclient object contains the location of the SOAP server as a parameter. The call() function of the soapclient object sends a request to the SOAP server and retrieves the SOAP packets as a response from the server.

This function accepts four parameters: procedure name, array of the soapval object, namespace, and SOAPAction parameter. The syntax to use the call() function is:

```
$client_object->call(procedure_name, array_of_soapval_object, namespace, SOAPAction)
```

In the above syntax, the procedure_name parameter indicates the name of the procedure to be invoked. The array_of_soapval_object parameter indicates the parameters passed by the SOAP client to the procedure. The urn:soapserver is the value for both namespace and SOAPAction parameters.

You can create a soap client code in any client-side script, such as Hypertext Markup Language (HTML) or Active Server Pages (ASP).

Listing E-6 shows how to code a SOAP client in HTML:

Listing E-6: Creating a SOAP Client

```
<html>
<head>
<basefont face="Times New Roman">
</head>
<body>
<?php
if(!$_POST['submit'])
{
   ?>
   <table border="0" cellspacing="5" cellpadding="5">
   <form action="<? echo $_SERVER['PHP_SELF']; ?>" method="POST">
   <tr>
   <td><b>Username:</b></td>
   <td><input type="text" name="user_name"></td>
   </tr>
   <tr>
   <td><b>Password:</b></td>
   <td><input type="password" name="user_pass"></td>
   </tr>
   <tr>
   <td><b>POP Server:</b></td>
   <td><input type="text" name="host_name"></td>
   </tr>
   <tr>
   <td colspan="2"  align="center"><input type="submit" name="submit" value="Retrieve
   Total Messages"></td>
   </tr>
   </form>
   </table>
   <?php
}
else
{
   include("class_soap_client.php");
   $server=http://mail.service/rpc/server.php;
   $parameter=array("user_name"=>$_POST['user_name'],
   "user_pass"=>$_POST['pop_pass'], "host_name"=>$POST['host_name']);
   $client=new soapclient($server);
   $object=new soapval("parameters","SOAPStruct",$parameter);
   echo $client->call("retrievePOP3Messages", array($object), "urn:soapserver",
   "urn:soapserver");
}
?>
</body>
</html>
```

The above listing shows that the HTML page accepts the user name, password, and the name of the mail server from the client. This information is passed to the retrievePOP3Messages procedure of the SOAP server, by clicking the Submit button. The SOAP server returns the number of messages stored in the mailbox.

Note You can display the client debug messages using the command:

```
$client->debug_flag=true;
```

Appendix F: PHPXML Classes

The PHPXML class is a collection of classes that process XML documents using PHP. PHPXML classes let you access XML documents using eXtensible Markup Language Path (XPath) language.

This appendix explains the packages of PHPXML classes. This appendix also explains the various classes present in the packages.

Classes in PHPXML

A package is a collection of classes that you can use to combine related classes. PHPXML consists of the following classes used for querying an XML document:

- Xml_check
- RDQL_query_document
- RDF_iterator
- RDQL_query
- RDF_document_iterator
- RDQL_db
- RDQL_query_db
- XqueryLite
- class_xindice

PHPXML classes contain the following classes used for validating an XML document:

- Path_parser
- RDDL_parser
- RSS_parser
- RDF_parser
- AbstractSAXParser
- AbstractFilter
- ExpatParser
- FilterOutput
- class_schematron
- class_xslt

The XML_check Class

The Xml_check class validates whether an XML document is well-formed or not. If the XML document is well-formed, the Xml_check class returns various information, such as the number of elements, attributes, and text section. The Xml_check class also returns the line number and column number of the XML document where an error is encountered. The XML_check class contains various functions, such as:

- `get_xml_elements()`: Provides the number of elements in an XML document.
- `get_xml_attributes()`: Provides the number of attributes in an XML document.
- `get_xml_size()`: Specifies the size of an XML document.

Resource Description Format Data Query Language

Resource Description Format (RDF) Data Query Language (RDQL) is a language that you use to query RDF documents from local file systems or URLs. RDQL is similar to Structured Query Language (SQL). You can implement RDQL using the following four classes:

- `RDQL_query_document`
- `RDF_iterator`
- `RDQL_query`
- `RDF_document_iterator`

The RDQL_query_document class implements the RDQL engine to query the RDF documents based on a URL and pathname. This class contains the rdql_query_url(string $query) function that queries the URL or string specified by the $query parameter. The rdql_query_url() function returns an associative array that contains the result of the RDQL query.

The RDQL engine uses the RDF_iterator class to query RDF documents from various sources, such as files and databases. The iterator is an object that specifies how to access and parse RDF documents. This class contains two functions, which are:

- `tuple_match(array $condition, array $row)`: Returns the value, true, if a row satisfies a condition.
- `find_tuples(array $condition, array $row)`: Returns the rows that satisfy a condition.

The RDQL_query class retrieves the RDQL rows from the RDF document using the RDQL engine. This class contains various functions, such as:

- `RDQL_query(RDF_iterator $iterator)`: Accesses the RDF rows in the RDF documents. It is a constructor of the class that does not return any value.
- `parse_query(string $query)`: Returns the results of the RDQL query.
- `tokenize(string $expression)`: Returns an array that contains various sections, such as SELECT, FROM, and WHERE, of the RDQL expression.
- `parse_select(string $expression)`: Returns an array that contains the SELECT section of the RDQL expression.

The RDF_document_iterator class contains an iterator that creates RDQL queries. This class does not contain any function.

Note RDF is a language that provides the accessed resources in the form of URLs.

The RDQL_db Class

The RDQL Database (DB) package is implemented using the class_rdql_db.php class. The RDQL DB package contains two classes, RDQL_db and RDQL_query_db. The RDQL_db class retrieves, stores, and deletes RDF documents from the MySQL database. Using the key of an RDF document, you can retrieve, store, and delete documents from the database. The primary key of a database is also called key. This class contains various functions, such as:

- `get_rdf_document(string $key)`: Returns a PHP string that is accessed by the key of the RDF document.
- `store_rdf_document(string $url, string $key)`: Returns the value, true, if the RDF document is stored in the database with the specified key. The $url variable provides a URL or file-path of the RDF document.
- `remove_rdf_document(string $key)`: Deletes an RDF document from the database, based on the key of the document, which is specified as an argument of the function.

You can query multiple documents from the database using the RDQL_query_db class, by specifying the name of documents in the FROM section of the query. You can use asterisk, *, to query all the documents stored in the database. This class contains the rdql_query_db(string $query) function that returns the result of the RDQL query.

The Path_parser Class

The Path_parser class implements the event-driven parsers, such as Expat. This class invokes the handlers when an element is encountered in an XML document. This class contains various functions to parse the XML documents, such as:

- `init()`: Parses the documents using a single object.
- `parse_file(string $url)`: Returns the value, true, if the document indicated by the $url variable is successfully parsed.
- `set_handler(string $url, string $handler)`: Processes XML elements when a specified handler is invoked.

The RDDL_parser Class

Resource Directory Description Language (RDDL) contains the RDDL_parser class that parses RDDL documents. This class contains a function that returns information pertaining to RDDL resources in an associative PHP array. Each element of an associative array contains various resources, such as role, href, type, title, and id. You can parse an RDDL document from URLs or files. The RDDL_parser class contains various functions, such as:

- `rddl_parse(string $url)`: Returns the value, true, if the RDDL document indicated by the $url variable is successfully parsed.
- `get_resources(string $rddl)`: Returns an associative array that contains resources found in an RDDL document.
- `get_error()`: Returns an error message when the rddl_parser() function returns the value, false.

Note An eXtensible Hyper Text Markup Language (XHTML) document that contains the <resource> element is known as an RDDL document.

The RSS_parser Class

Really Simple Syndication (RSS) contains the RSS_parser class that implements the RSS1.0 parser. This class contains a

function that parses RSS documents and returns an associative array. Each element of an associative array contains information about channels, such as channel data, channel_image, channel_textinput, and channel_items, in the RSS document. The RSS_parser class contains various functions, such as:

- `rss_parse(string $url)`: Returns the value, true, if the RSS document specified by the $url variable is successfully parsed.
- `get_channel_data(string $url)`: Returns an associative array that contains information about the channels, such as link, description, image, and text input.
- `get_items_data(string $url)`: Returns an associative array that contains information about the items stored in the RSS document.

Note RSS is an XML-based language that displays information about Web sites, such as hyperlinks and the content of the Web sites.

The Rdf_parser Class

Resource Description Framework (RDF) contains the Rdf_parser class that parses the RDF document with the RDF specifications. The Rdf_parser class contains various functions, such as:

- `rdf_parser_create(string $encoding_scheme)`: Returns the value, true, if a new parser is created. The $encoding_scheme parameter of the function, which specifies the encoding scheme to parse a document, is optional.
- `rdf_set_element_handler(string $start, string $end)`: Invokes the handlers when the start and end elements of the non-RDF elements are encountered in the RDF document.
- `rdf_Parse(string $str, $length, $is_final)`: Returns the value, true, if the RDF document is successfully parsed. The $str parameter indicates the chunk of data to be parsed, the $length parameter indicates the length of data to be parsed, and the $is_final parameter indicates the final data to be parsed.

Note RDF is a language that you can use to represent information on the Internet.

The XqueryLite Class

The XqueryLite class queries XML documents. This class lets you run queries in the Xquery 1.0 language, using files and PHP strings. The constructor of the XqueryLite class does not accept any parameter, and does not generate any result. The init() function initializes a query. The XqueryLite class contains various functions, such as:

- `evaluate_xqueryl(string $query)`: Runs an Xquery Lite 1.0 query.
- `_get_root_name(object $node)`: Returns the name of the root element of the XML document.
- `_parse_query(string $query)`: Returns the result of the Xquery Lite expression.

Note XqueryLite is a language that queries XML documents.

The class_sax_filters.php Class

The Simple Application Programming Interface (API) for XML (SAX) filters package contains the class_sax_filters.php class. This class contains a set of classes that implements the SAX parser filters. The Expat parser is an example of the SAX parser. Filters modify, query, update, and transform XML documents.

The class_sax_filters.php class contains four classes, which are:

- `AbstractSAXParser`
- `AbstractFilter`
- `ExpatParser`
- `FilterOutput`

The class_schematron Class

Schematron is a validation language that contains the class_schematron class to validate XML documents. You can validate XML documents using files, PHP strings and URLs. The class_schematron class contains various functions, such as:

- `compile_schematron_from_file(string $file)`: Returns an XSLT stylesheet after compiling the Schematron script.
- `validate_mem_using_file(string $xml_str, string $validation_file)`: Checks the string of the XML document specified in the $xml_str variable from the file specified in the $validation_file parameter.
- `validate_mem_using_mem(string $xml_str, string $validation_str)`: Checks an XML document using the specified PHP string in the $validation_str parameter.

Note Schematron is a structured schema language that creates a tree structure for validating XML documents. You can use this class to perform validations in PHP.

The class_xslt Class

eXtensible Style sheet Language Transformation (XSLT) contains the class_xslt class to implement the XSLT processor. This

class supports XSLT transformations and XML documents. This class contains functions that set the XSLT style sheet to be used from PHP strings and URLs. The class_xslt class contains various functions, such as:

- `getOutput()`: Provides the output of the XSLT transformation.
- `transform(string $url)`: Processes an XSLT transformation.
- `setXslString(string $xsl)`: Assigns the string to be used for the XSLT stylesheet.
- `setXsl(string $url)`: Assigns the URL to be used for the XSLT stylesheet.

The class_xindice Class

The class_xindice class accesses a Xindice 1.0 XML database using PHP script. The constructor of the class accepts two parameters, the URL of the Xindice server and the port of the server. A function of the class returns the number of documents present in the collection of databases. The class_xindice contains various functions, such as:

- `createCollection(string $base, string $collection)`: Builds a collection of databases with the name specified in the $collection variable.
- `getDocumentCount(string $collectionLocation)`: Provides the number of documents that exist in the specified location of a collection.
- `InsertDocument(string $collection, string $id, $xmldoc)`: Sets a new document, specified by the $xmldoc variable, in the collection.
- `getDocument(string $collection, string $id)`: Accesses a document from the database.
- `removeDocument(string $collection, string $id)`: Deletes a document from a collection.

Note You need to install the Xindice XML-RPC plugin because the class_xindice class uses it.

PREVIOUS NEXT

Appendix G: PHP and Extensible Stylesheet Language for Transformation

 Download CD Content

Hypertext Preprocessor (PHP) is a server-side scripting language that creates Web applications and processes eXtensible Markup Language (XML) documents using eXtensible Stylesheet Language for Transformation (XSLT). XSLT transforms XML documents into other formats, such as Hypertext Markup Language (HTML), ASCII, and Wireless Markup Language (WML). PHP uses the Sablotron XSLT processor to perform server-side transformation of XML documents.

This appendix explains how to use PHP with XSLT to transform XML documents.

Implementing XSLT with PHP

The XSLT processor combines PHP and XSLT to perform server-side XSLT transformations that convert the XML files into HTML files. PHP includes an XSLT extension that supports various XSLT processors, such as Sablotron and Xalan. The XSLT extension contains an Application Programming Interface (API) that lets you work with other processors. An API contains functions that are compatible with other XSLT engines. You can use these functions to change the XSLT processor without modifying the PHP code.

The PHP XSLT extension uses the Sablotron XSLT processor as the default processor. You need to install the Sablotron XSLT processor explicitly because it is not automatically installed with the Linux operating system. You can enable the Sablotron XSLT processor by reconfiguring and installing PHP.

To perform an XSL transformation with PHP:

1. Include the xml and xsl files in the PHP script.
2. Create an object of the XSLT processor, using the xslt_create() function of PHP, as shown in the following code:

   ```
   $xp=xslt_create();
   ```

   The above code shows that the XSLT processor is referenced by the $xp variable, and used by other XSLT functions, such as xslt_process().
3. Process the XML source document and stylesheet using the xslt_process() function of PHP. The xslt_process() function accepts three mandatory parameters: the reference of the XSLT processor, the XML document, and the stylesheet.

   Listing G-1 shows how to use the xslt_process() function:

   Listing G-1: Using the xslt_process() Function

   ```
   if($res=xslt_process($xp, $xfile, $xsltfile))
   {
      echo $result;
   }
   else
   {
      echo "Error";
   }
   ```

   In the above code:

   - The $xp parameter specifies the reference of the XSLT processor.
   - The $xfile parameter specifies the name of the XML document.
   - The $xsltfile parameter specifies the name of the stylesheet.
   - The xslt_process() function reads and validates the XML document against the rules specified in the stylesheet, and then generates a result tree.
   - The $res variable contains the result of the transformation.
   - The statement, echo $result, displays the result tree if the transformation is successful; otherwise, displays an error message.
4. Release the memory occupied by the XSLT processor using the xslt_free() function. The xslt_free() function accepts a parameter that specifies the reference of the XSLT processor. The code to use the xslt_free() function is:

   ```
   xslt_free($xp);
   ```

Note The xslt_process() function accepts six parameters, out of which, three parameters are optional and three are mandatory. The three optional parameters are:

- Name of the file that stores the result tree.
- Name of an associative array that contains buffers.

- Name of an associative array that contains XSLT parameters.

Handling Errors

PHP XSLT extension provides functions for handling and displaying errors generated at the time of the transformation of XML documents. The xslt_error() function displays the error message, and the xslt_errno() function returns the error code. You can define a user-defined function for handling errors, using the xslt_set_error_handler() function. The syntax of the xslt_error() function is:

```
xslt_error($xslt_processor);
```

In the above syntax, the $xslt_processor parameter indicates the reference of the XSLT processor. The xslt_error() function accepts only a single parameter, and generates a system-defined error message. If you do not provide a parameter to the function, it displays the last error message generated in the transformation process.

The syntax of the xslt_errno() function is:

```
xslt_errno($xslt_processor);
```

In the above syntax, the $xslt_processor parameter indicates the reference of the XSLT processor. The xslt_errno() function accepts only a single parameter, and returns the error code corresponding to the generated error. If you do not provide a parameter to the function, the function returns the error code of the last error generated in the transformation process.

The xslt_set_error_handler() function lets you define a user-defined function for handling errors. This function accepts two parameters, the reference of the XSLT processor and the name of the user-defined function to be invoked.

The syntax of the xslt_set_error_handler() function is:

```
xslt_set_error_handler($xslt_processor, "user_defined_function");
```

In the above syntax, the $xslt_processor parameter specifies the reference of the XSLT processor. The second parameter specifies the name of the function to be invoked when an error occurs in the transformation process.

The user-defined function for handling error accepts four parameters, which are:

- The reference of the XSLT processor.
- The error level.
- The error code.
- The associative array containing error information.

Logging Processor Messages

Error messages are stored in a log file, which specifies the description of errors generated in the transformation process. The xslt_set_log() file lets you initiate the logging process. You need to specify the name of the log file in the xslt_set_log() function, and invoke the xslt_set_log() function twice. The first invocation of the function specifies the initiation of the logging process, and the second invocation specifies the name of the file that stores the logging information. When you invoke the xslt_set_log() function for the first time, it accepts two parameters, which are:

- The reference of the XSLT processor.
- The Boolean value, true.

Enter the following code to invoke the xslt_set_log() function for the first time:

```
xsl_set_log($xp, true);
```

When you invoke the xslt_set_log() function for the second time, it accepts two parameters, which are:

- The reference of the XSLT processor.
- The name of a file that stores the log messages.

Enter the following code to invoke the xslt_set_log() function for the second time:

```
xsl_set_log($xp, "/xfile.log");
```

In the above code:

- The $xp parameter indicates the reference of the XSLT processor.
- The xfile.log file stores the logging information.
- The Web browser displays the logging information if you do not provide the second parameter.
- The logging information is appended at the end of the log file if the log file exists.

Using Named Buffers

Named buffers are the references that help store XML and XSLT documents in memory. You can provide the named buffers in the fifth parameter of the xslt_process() function. You can use the named buffers of the XML and XSLT strings in place of the XML and XSLT files specified in the second and third parameters of the xslt_process() function. Enter the following code to create and use the named buffers of the XML and XSLT strings:

```
$xml_str=join('',file($xfile);
$xslt_str=join('', file($xsltfile));
$parameter_buff=array("/xml"=>$xml_str, "/xslt"=>$xslt_str);
$xp=xslt_create();
$res=xslt_process($xp, "arg:/xml", "arg:/xslt", NULL, $parameter_buff);
```

In the above code:

- The file() function of PHP reads the contents of the file and stores it in an array.
- The join() function stores the text of a file in a single string.
- The $xml_str variable contains the named buffer of the XML document.
- The $xslt_str variable contains the named buffer of the XSLT document.
- The $parameter_buff parameter is an associative array that consists of two key-value pairs.
    - The /xml key specifies the named buffer of the XML document.
    - The /xslt key specifies the named buffer of the XSLT document.

If you use the Sablotron processor, you need to prefix the named buffers with the arg: text in the xslt_process() function.

PREVIOUS NEXT

Index

A-C

Index

D-E

Index

H-I

Index

J-M

Index

O-P

Index

R-S

Index

U-W

Index

X

List of Figures

Chapter 1: Introducing PHP and XML

Chapter 2: PHP and Simple Application Programming Interface for XML

Chapter 3: PHP and Document Object Model

Chapter 5: PHP and XPath

Chapter 6: PHP and XML-Remote Procedure Calls

Chapter 7: PHP and Web Distributed Data Exchange

Chapter 8: Working with Databases using PHP and XML

Chapter 9: Creating an Online Shopping Cart Application

PREVIOUS NEXT

List of Tables

List of Listings

Chapter 1: Introducing PHP and XML

Chapter 2: PHP and Simple Application Programming Interface for XML

Chapter 3: PHP and Document Object Model

CD Content

Following are select files from this book's Companion CD-ROM. These files are copyright protected by the publisher, author, and/or other third parties. Unauthorized use, reproduction, or distribution is strictly prohibited.

Click on the link(s) below to download the files to your computer:

File	Description	Size
All CD Content	Integrating PHP and XML	80,276
Chapter 1:	Introducing PHP and XML	6,991
Chapter 2:	PHP and Simple Application Programming Interface for XML	7,201
Chapter 3:	PHP and Document Object Model	7,742
Chapter 4:	Understanding Extensible Stylesheet Language for Transformation	2,511
Chapter 5:	PHP and XPath	4,726
Chapter 6:	PHP and XML-Remote Procedure Calls	4,613
Chapter 7:	PHP and Web Distributed Data Exchange	4,095
Chapter 8:	Working with Databases using PHP and XML	8,582
Chapter 9:	Creating an Online Shopping Cart Application	30,574
Appendix E:	Implementing SOAP using SOAPx4	3,161
Appendix G:	PHP and Extensible Stylesheet Language for Transformation	300

melasuce@mail.r

Integrating PHP and XML

SkillSoft Press © 2004

Learn how to use SAX, XSLT, and XPath to manipulate XML docume
protocol for accessing procedures on a remote computer, and much

Team LiB

Introduction

About InstantCode Books

The InstantCode series is designed to provide you - the developer - with code you can use for common tasks in the workplace. The goal of the InstantCode series is not to provide comprehensive information on specific technologies - this is typically well-covered in other books. Instead, the purpose of this series is to provide actual code listings that you can immediately put to use in building applications for your particular requirements.

How These Books are Structured

The underlying philosophy of the InstantCode series is to present code listings that you can download and apply to your own business needs. To support this, these books are divided into chapters, each covering an independent task.

Each chapter includes a brief description of the task, followed by an overview of the element of the book's subject technology that we will use to perform that task. Each section ends with a code listing: each of the individual code segments in the chapter is independently downloadable, as is the complete chapter code. You will be able to download source code files, as well as application files.

Who Should Read These Books

These books are written for software development professionals who have basic knowledge of the associated technology and want to develop customized technology solutions.

About the Book

PHP is a server-side scripting language used to create Web applications. XML is a markup language used to exchange data among Web applications. PHP can be integrated with XML to create Web applications. This book describes how to use SAX, XSLT, and XPath to manipulate XML documents. It also describes use of XML-RPC protocol for accessing procedures on a remote computer. In addition, the book covers WDDX, a technology used for information exchange between different programming languages.

This book describes how to create an online shopping cart application that allows an end user to search for a specific book in a database, place an order for the book, and purchase the book online.

About the Authors

Amrita Dubey holds a Bachelor's degree in Electronics and Communication Engineering. She has worked on various development projects, such as UNIX shell programming and designing function generator using IC XR-2206. She is proficient in 8085 microprocessor programming. She also has knowledge of CNCs, PLCs, circuit designing on PCBs, Software Quality Testing, RDBMS, SQL Server, and Database Design Studio Professional 2.21.1. As a technical writer, Amrita has co-authored books on Digital Electronics, Integrating Java with Oracle9i, PIC Microcontrollers, Robotics, and Integrating PHP and XML.

Poonam Sharma holds a Bachelor's degree in Commerce and a DNIIT diploma. In addition, she is pursuing Master's degree in Computer Applications. She is proficient in technologies such as Java, C++, VB, ASP, Servlets, JSP, HTML, UNIX, Linux, and SQL Server 2000. Poonam has worked on projects such as Online Banking using VB and SQL and Creating Chat Application in Java. As a technical writer, she has co-authored books on Working with Korn Shell, Administering Red Hat Linux 9, Basic Programming in C++, Unix Operating System, and Integrating PHP and XML.

Sreeparna Dasgupta holds a Master's degree in Computer Science and has earned 'A' Level certificate from DOEACC. She has worked and trained people on several technologies, such as Java, C, C++, Visual Basic, Macromedia Flash MX, Photoshop, Macromedia Director MX, Oracle, SQL Server, Perl, ASP, and JSP. In addition, Sreeparna has developed online Web applications using technologies such as ASP.

Vishi Gupta holds a degree in Electrical Engineering. She is proficient in C, C++, Linux and PHP. As a technical writer, she has co-authored several books and articles on varying technologies including PHP, XML, SOAP, Linux, Core Java, PHPLens, C and C++.

Lalit Kapoor holds a Bachelor's degree in Computer Engineering. He has completed DNIIT course from NIIT. As a technical writer, he has authored articles on Java, Oracle, SQL Server, and PHP.

Credits

I would like to thank Reena Roy, S. Sripriya, Sabari Roy, and Rajender Arora for helping me complete the book on time. I also thank the editors and the quality assurance team for their timely help.

Copyright

Integrating PHP and XML

Published by SkillSoft Corporation
20 Industrial Park Drive
Nashua, NH 03062
(603) 324-3000

information@skillsoft.com

Chapter 1: Introducing PHP and XML

Download CD Content

PHP is a server-side HTML embedded scripting language that you can use to create dynamic Web pages. It includes predefined functions that create, open, read, write, and close files stored on the server. You can create XML-based Web applications using PHP.

Some browsers such as Netscape Communicator 4.x series do not contain Some browsers such as Netscape Communicator 4.x series do not contain (XML) parser. Hypertext Preprocessor (PHP) overcomes this problem by supporting XML parsing. PHP provides a Document Object Model to access XML elements, an XML extension, and an eXtensible Stylesheet Language (XSL) processor to support XML parsing.

This chapter introduces PHP fundamentals. It describes data types, variables, operators, control structures, functions, and error handling in PHP. It also explains how to create classes in PHP and use PHP to work with files. This chapter also introduces XML fundamentals, such as Document Type Declaration (DTD), namespaces, and XML schemas.

Introducing PHP

PHP, developed using the C language, is specifically designed to work with databases. It provides a set of Application Programming Interfaces (APIs) to connect to different types of database servers, such as MySQL, SQL Server, and Oracle. PHP scripts are portable because they can run on various operating systems, such as Linux and Windows.

When a user accesses a Web page created using a PHP script, the Web server contacts the PHP engine to load, compile, and run the PHP script. Using PHP script, you can work with different data types, variables, constants, operators, functions, classes, objects, and files.

Data Types, Variables, and Constants

A variable is a container that stores variable data, such as number, character, and date.

PHP provides data types that specify type of values a variable can contain. For example, a data type defines that a variable, num, can contain only numbers. The data types that PHP supports are:

- Integer: Represents whole numbers. In PHP, integer variables can store values between -2, 147, 483, 648 and +2, 147, 483, 647. For example, in $a=5, $a is an integer variable that is assigned the value, 5.
- Float: Stores decimal values. PHP lets you store floating numbers in both normal and scientific notation. Scientific notation is a shorthand way of writing very large or very small numbers. A number expressed in scientific notation is expressed as a decimal number between 1 and 10, followed by e or E, multiplied by a power of 10. For example, in $a=5.2E2, $a is a float variable created using scientific notation and contains the value 520.
- Boolean: Stores the values, true or false. For example, in $a=true, $a is a Boolean variable that is assigned the value, true.
- String: Stores string information enclosed within double quotes. For example, in $a="Angela Jones", $a represents a string variable that contains the string, Angela Jones.
- Array: Stores elements in the form of key/value pairs. The key values start with zero.
- Object: Stores data and provides functions to process the data.
- Resource: Stores references to functions or other resources external to PHP.
- Null: Represents an undefined value.

Note In PHP, you can declare variables without specifying its data type.

Operators

PHP provides several operators, such as the arithmetic, comparison, logical, and assignment operators. You can perform arithmetic operations on variables and constants using the arithmetic operators of PHP.

Table 1-1 lists the arithmetic operators:

Table 1-1: Arithmetic Operators

Name	Symbol	Description
Addition	+	Adds two numbers.
Subtraction	-	Subtracts two numbers.
Multiplication	*	Calculates the product of two numbers.
Division	/	Divides two numbers.
Modulus	%	Calculates the remainder when one number is divided by the other.
Increment	++	Increments the number, to which it is applied, by 1.

Decrement	--	Decrements the number, to which it is applied, by 1.

PHP also supports the comparison operators to compare two or more values, to determine which value is greater, lesser, equal, or not equal to the other.

Table 1-2 lists the comparison operators in PHP:

Table 1-2: Comparison Operators

Name	Symbol	Description
Greater than	>	Returns true if the value on the left hand side of the operator is greater than the value on the right hand side.
Less than	<	Returns true if the value on the left hand side of the operator is less than the value on the right hand side.
Greater than equal to	>=	Returns true if the value on the left hand side of the operator is greater than or equal to the value on the right hand side.
Less than equal to	<=	Returns true if the value on the left hand side of the operator is less than or equal to the value on the right hand side.
Equal to	==	Returns true if the two values being compared are equal.
Not Equal to	!=	Returns true if the two values being compared are not equal.

PHP supports the logical operators that you can use to create Boolean expressions. Boolean expressions either return true or false.

Table 1-3 lists the operators that PHP supports:

Table 1-3: Logical Operators in PHP

Name	Symbol	Description
And	&&	Returns true if both the conditions placed on the left and right of the operator are true.
Or	||	Returns true if either of the conditions placed on the left and right of the operator is true.
Not	!	Returns true if the condition specified is not true.

You can use assignment operators in PHP to assign values to the variables. PHP also supports compound operators that perform the functions of the arithmetic and assignment operators in a single statement.

Table 1-4 lists the assignment and compound operators in PHP:

Table 1-4: Assignment and Compound Operators

Operator	Implementation	Description
=	a =5	Assign the value, 5, to the variable, a.
+=	a+=5	Adds the value, 5, to the original value of the variable, a, and then assigns the sum to the variable, a. This is equivalent to a=a+5.
-=	a-=5	Subtracts the value, 5, from the original value of the variable, a, and then assigns the difference to the variable, a. This is equivalent to a=a-5.
*=	a*=5	Multiplies the value, 5, with the original value of the variable, a, and then assigns the product to the variable, a. This is equivalent to a=a*5.
/=	a/=5	Divides the value in the variable, a, by 5 and then assigns the result to the variable, a. This is equivalent to a=a/5.
%=	a%=5	Divides the value in the variable, a, by 5 and then assigns the remainder to the variable, a. This is equivalent to a=a%5.

Control Structures

You can control the flow of execution of a PHP script using control structures. Two types of control structures are conditional statements and loops.

Conditional statements support unidirectional flow of control in processing a script. The conditional statement returns the true or false value after evaluating a condition, and then transfers the control to a section of the script, depending on the value returned after evaluating the condition. In PHP, two types of conditional statements are if else and switch case statement.

The if else conditional statement is divided into two parts, the if statement and the else statement. The if statement evaluates a condition and returns the value, true, or false. The if statement is followed by the PHP script enclosed in braces. This script runs if the value returned is true. The else statement contains the PHP script that runs if the condition evaluated is false. The syntax of the if else conditional statement is:

```
if(condition)
{statements to run if condition is true}
else
{statements to run if condition is false}
```

In the above syntax, the condition parameter is an expression that is evaluated to return true or false. To run a single PHP programming statement, you do not need the braces.

Listing 1-1 shows how to use the if condition to display the maximum of three numbers:

Listing 1-1: Maximum of Three Numbers

```
<?php
$x=10;
$y=20;
$z=30;
if($x > $y)
{
   if($x > $z)
   {
      echo("The maximum number is ");
      echo("$x<p>");
   }
   else
   {
      echo("The maximum number is ");
      echo("$z<p>");
   }
}
else
{
   if($y > $z)
   {
      echo("The maximum number is ");
      echo("$y<p>");
   }
   else
   {
      echo("The maximum number is ");
      echo("$z<p>");
   }
}
?>
```

The above listing shows the use of the if else condition statements to create a PHP script. The if condition compares the values of the two variables, and then controls the flow of execution of the script, depending on whether the condition returns true or false.

Create a new folder, PHP_XML, in the var/www/html directory of the Apache Web server. Save the above listing as MaxThree.php in the PHP_XML folder.

Figure 1-1 shows the output when PHP_XML is viewed in the Konqueror Web browser:

Figure 1-1: Viewing MaxThree.php

The switch case statement is another conditional statement that controls the flow of execution of a script. The switch statement compares a single variable with several values. To specify the values to which a variable is compared, you use the case statement with the switch statement. The syntax of the switch case conditional statement is:

```
switch($Variable)
{
   case 1:
   //statements to be run if case is true;
   break;
   case 2:
   //statements to be run if case is true;
   break;
   default:
   //statements to be run if all cases are false;
}
```

In the above syntax, the break statement is used within a case statement to identify the end of a case. The default statement is used in the switch case condition to specify the PHP statements that are run if all the cases are false.

Listing 1-2 shows how to use the switch case conditional statement to perform simple arithmetic operations:

Listing 1-2: Using the switch case Conditional Statement

```
<?php
//Initializing variables
$a = 10;
$b = 5;
echo("1. Addition <br>");
echo("2. Subtraction <br>");
echo("3. Multiplication <br>");
echo("4. Division <br>");
$ch=3;
switch ($ch)
{
   case 1:
   $d=$a+$b;
   echo("The sum of the two numbers is $d");
   break;
   case 2:
   $d=$a-$b;
   echo("The difference of the two numbers is $d");
   break;
   case 3:
   $d=$a*$b;
   echo("The product of the two numbers is $d");
   break;
   case 4:
   $d=$a/$b;
   echo("The quotient of the two numbers is $d");
   break;
   default:
   echo("Incorrect Option");
   break;
}
?>
```

The above listing shows how to use the switch case statement. The switch case statements:

- Display the sum of two numbers, if the variable, $ch, is assigned the value, 1.
- Display the difference of the two numbers, if the variable, $ch, is assigned the value, 2.
- Display the product of the two numbers, if the variable, $ch, is assigned the value, 3.
- Display the quotient of the two numbers, if the variable, $ch, is assigned the value, 4.
- Assign the variable, $ch, the value 3 to multiply two numbers.

Because value, 3, is assigned to the variable $ch, Listing 1-2 displays the product of the two numbers.

Save the above listing as Switch.php in the PHP_XML folder.

Figure 1-2 shows the output when the Switch.php file is viewed in the Konqueror Web browser:

Figure 1-2: Viewing Switch.php

Loops support the repetitive execution of a set of statements in the PHP script. The loop statements evaluate a Boolean condition on each iteration that returns true or false after evaluating a condition. A set of statements enclosed within a loop is processed if the condition evaluates to true. The different types of loop statements in PHP are:

- `while loop`
- `do while loop`
- `for loop`
- `foreach loop`

The while loop evaluates a condition as true or false. The variable that you use in the while condition needs to be initialized before the while loop. The value of the variable is changed as the while loop executes. The execution of the while loop terminates as soon as the specified condition becomes false. The syntax for using the while loop is:

```
$var=initialization
while(condition checking the value of the variable, $var)
{
   //statements to run;
   //increment/decrement var
}
```

In the above syntax, the variable, $var, is initialized outside the while loop. The while loop evaluates the condition as true or false and then runs the statements within the loop, if the condition is true. The increment or decrement operation within the loop modifies the value of the variable used in the condition till the condition becomes false.

Listing 1-3 shows how to use the while loop to display even numbers from 2 to 30:

Listing 1-3: Using the while Loop

```
<?php
$a=2;
echo("<p>Even Numbers from 2 to 30<p>");
while ($a <=30)
{
   echo("$a <br>");
   $a=$a+2;
}
?>
```

The above listing shows how to use the while loop to display even numbers. In the above listing:

- The variable, a, is initialized to value 2.
- The while condition compares whether the value of the variable is less than or equal to 30, and returns true or false.
- The increment operation adds 2 to the value of the variable, a, every time the loop runs.

Save the above listing as While.php in the PHP_XML folder.

Figure 1-3 shows the output when While.php is viewed in the Konqueror Web browser:

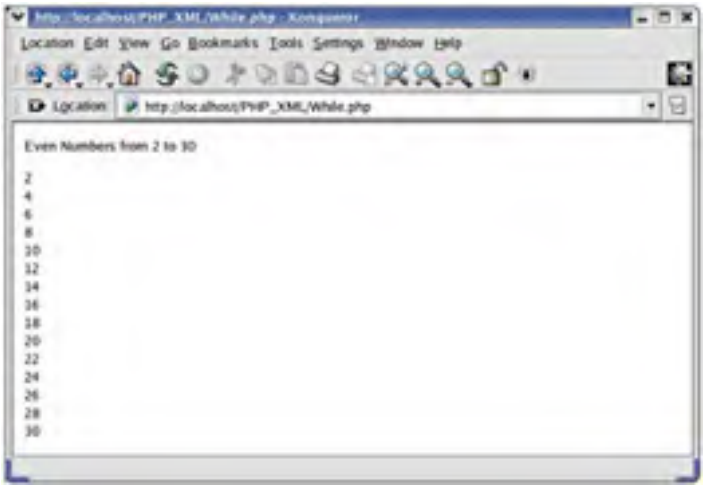


Figure 1-3: Viewing While.php

In the do-while loop, unlike in the while loop, the condition is evaluated after executing the statements once in the do-while loop. The syntax of the do-while loop is:

```
$var = initialization
do
{
   //statements to run;
   //increment/decrement var;
} while(condition);
```

In the above syntax, the variable is initialized outside the do-while loop. The do keyword indicates the beginning of the do-while loop. The while condition is specified at the end of the listing and ends with the semicolon symbol.

In a for loop statement, you can initialize a counter variable, specify the condition, and specify the increment or decrement condition in the loop statement together. The syntax of the for loop statement is:

```
for(initialization;condition;increment_decrement_operation)

{
   //Statements to run;
}
```

You can embed one for loop statement inside another.

Listing 1-4 shows how to use the for loop to display prime numbers:

Listing 1-4: Using the for Loop

```
<?php
echo("Prime Numbers between 1 and 30<p>");
//For loop from 1 to 30
for($var=1;$var<30;$var++)
{
   $ctr=0;
   for($k=2;$k<=$var/2;$k++)
   {
      $d=$var%$k;
      if ($d == 0)
      {
         $ctr=1;
         break;
      }
   }
   if($ctr==0)
   {
      echo($var);
      echo("<br>");
   }
}
?>
```

The above listing shows how to use the for loop to display prime numbers from 1 to 30. The PHP script uses nested for loops to display the value of the variable, $var. Save the above listing as For.php in the PHP_XML folder.

Figure 1-4 shows the output when For.php is viewed in the Konqueror Web browser:

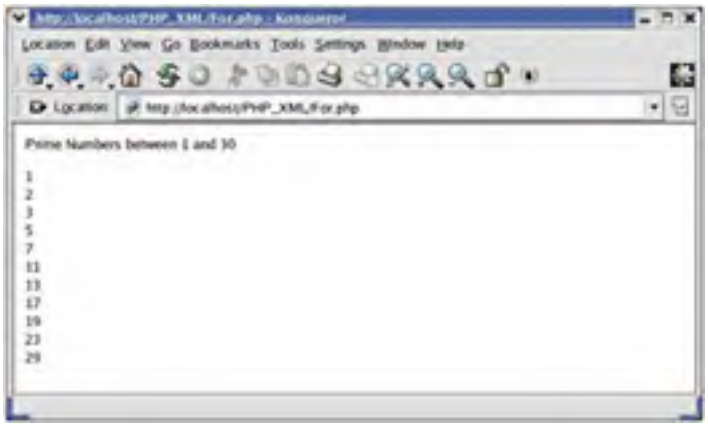


Figure 1-4: Viewing For.php

Functions

A function is a set of instructions that is grouped under a name. Instead of repeating all the instruction steps every time you use them in the script, you can call the function to run the same set of instructions again. PHP provides a set of built-in functions to use in the PHP scripts. In addition to the built-in functions, you can also define certain functions called user-defined functions. The syntax to define a function in PHP is:

```
<?php
function funcname($arg1,$arg2)
{//Statements to run;
return $val;}
?>
```

In the above syntax:

- The function keyword defines a PHP function.
- The funcname parameter defines the name of the function.
- The $arg1 and $arg2 parameters indicate the arguments that the function can receive.
- The return keyword returns a value from the function.

Listing 1-5 shows how to define and call a function in PHP:

Listing 1-5: User-Defined Function in PHP

```
<?php
echo("<center><h2>Displaying even numbers</h2></center><p><p>");
function displayeven()
{
   $ctr=0;
   echo("<font size=4>");
```

```
   for($i=2;$i<=100;$i+=2)
   {
      echo("$i    ");
      $ctr++;
      if($ctr%10==0)
      {
         echo("<p>");
      }
   }
   echo("</font>");
}
echo("<center><h2>Displaying even numbers</h2></center><p><p>");
displayeven();
?>
```

The above listing creates a function, displayeven() that displays even numbers greater than zero and less than or equal to 100.

Save the above listing as Functions.php in the PHP_XML folder.

Figure 1-5 shows the output when Function.php is viewed in the Web browser:

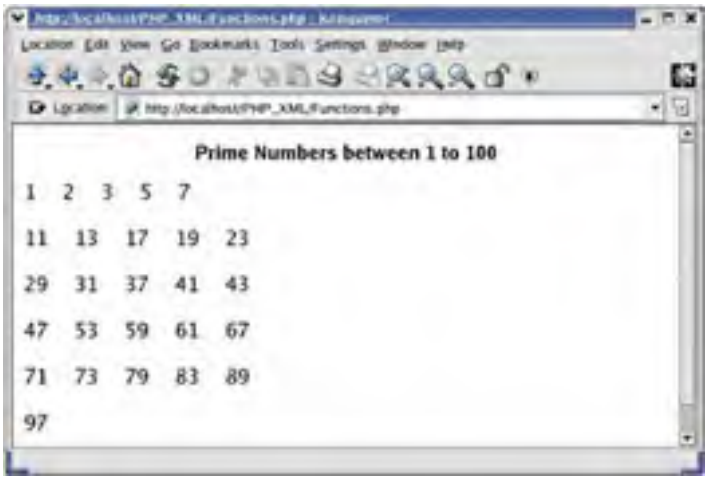


Figure 1-5: Viewing Functions.php

You can create nested functions, where you can define a function within another function. For example, in an application where you need to generate all prime numbers between two numbers entered by the end user, the outer function accepts the end user input and the inner function generates prime numbers. The inner function is a nested function, and is called only after calling the outer function. The syntax for creating a nested function is:

```
<?php
function outer_function($arg1)
{
   //Statements to run for outer_function();
   function inner_innerfunction($arg2, $arg3)
   {
      //Statements to run for inner_function()
   }
   outer_function();
   inner_function();
}
```

The above syntax shows how to create two functions, inner_function() and outer_function(). The inner_function() function is nested within the outer_function() function. The outer_function() function is called before the inner_function() function.

> **Note** In a nested function definition, calling the inner function before the outer function results in an error.

Listing 1-6 shows how to use nested functions in PHP:

Listing 1-6: Using Nested PHP Functions

```
<?php
function msg()
{
   echo("<center><h2>Displaying even
   numbers</h2></center><p><p>");
   function displayeven()
   {
      $ctr=0;
      echo("<font size=4>");
      for($i=2;$i<=100;$i+=2)
      {
         echo("$i    ");
         $ctr++;
         if($ctr%10==0)
         {
```

```
            echo("<p>");
        }
    }
    echo("</font>");
  }
}
msg();
displayeven();
?>
```

The above listing shows how to use nested functions. In the above listing:

- the displayeven() function is nested within the msg() function.
- the msg() function displays a message, Displaying even Numbers.
- the displayeven() function displays even numbers between 1 and 100.
- the displayeven() function is called only after calling the msg() function.

Save the above listing as NestedFunction.php in the PHP_XML folder.

Figure 1-6 shows the output when NestedFunction.php is viewed in the Web browser:

Figure 1-6: Viewing NestedFunction.php

Error Handling

An error represents an incorrect result, produced either because of wrong logic, such as division by zero, or improper working of other applications, such as the database server that the PHP script is using. PHP displays the error message either on the Web page, or in the error log files. The error message contains information about the nature and source of an error.

PHP has built-in support for error handling and classifies errors according to the following error types:

- E_NOTICE: Represents minor nonfatal errors that you use to identify the possible bugs within the PHP script. These errors do not stop the execution of the PHP script. E_NOTICE errors are not displayed on the Web page by default.
- E_WARNING: Represents nonfatal runtime errors that do not stop the execution of the PHP scripts. The errors generated by external applications, such as database servers, are represented by E_WARNING errors. These errors differ from E_NOTICE as the latter is associated with identifying bugs within the PHP script whereas the former is generated by external applications.
- E_ERROR: Represents fatal errors that stop the execution of the PHP script.
- E_PARSE: Represents parsing errors generated by the PHP parser, which indicate syntax error in the PHP script.

In addition to the standard errors, PHP also provides custom error types that PHP scripts can generate using the trigger_error() function. The custom error types are:

- E_USER_NOTICE: Represents the customized or user-defined version of the E_NOTICE error type.
- E_USER_WARNING: Represents the customized or user-defined version of the E_WARNING error type.
- E_USER_ERROR: Represents the customized or user-defined version of the E_ERROR error type.

PHP provides built-in error handling functions that handle errors of different types, such as E_ERROR, E_WARNING. Some error handling functions are:

- int error_reporting (int err_type): Specifies what error types should be displayed on the Web page. The parameter, err_type, takes either integers or named constants to represent the error type. Table 1-5 shows the named constants and their equivalent integer values:

Table 1-5: Named Constants and Equivalent Integer Values

Named Constant	Integer Value
E_ERROR	1
E_WARNING	2
E_PARSE	4
E_NOTICE	8
E_USER_ERROR	256
E_USER_WARNING	512
E_USER_NOTICE	1024
E_ALL	2047

- `int error_log ( string message [, int message_type [, string destination ]])`: Sends the error message to the specified location. The specified location can be the default system log file, an email address, a remote destination, or a local log file. The first parameter represents the error message that is stored in the specified destination. The second parameter represents the message type. The third parameter represents one of the four destinations where the error message needs to be stored. The second parameter can take one of the following values:
  - `0`: Stores the error messages in the default system log file.
  - `1`: Sends the error message to the destination email ID specified in the third parameter.
  - `2`: Saves the error message in a file present on a different computer in a network. The third parameter specifies the IP address and the port number of the destination computer.
  - `3`: Sends the error message to the destination file specified in the third parameter.
- `void trigger_error ( string error_msg [, int error_type])`: Triggers user-defined warnings, notices, or errors. The first parameter represents the error message and the second parameter represents the error type that needs to be triggered.

Listing 1-7 shows how to trigger errors using the trigger_error() function.

Listing 1-7: Generating Errors Using the trigger_error() Function

```
<?php
// set the error reporting level for this script
error_reporting(E_USER_ERROR);
if (assert($divisor == 0))
{
  error_log("Cannot perform division by zero", 3, "var/www/html/PHP_XML/error.log");
  trigger_error("Cannot perform division by zero", E_USER_ERROR);
}
?>
```

The above listing generates an error that stops the execution of the script if the condition passed to the assert() function is true. In the above listing:

- The assert() function checks whether the divisor is zero or not.
- The error_log() function sends the message, Cannot perform division by zero, to the log file specified by the second parameter when the divisor is zero.

Save the above listing as Trigger.php in the PHP_XML folder.

Figure 1-7 shows the output when Trigger.php is viewed in the Web browser:

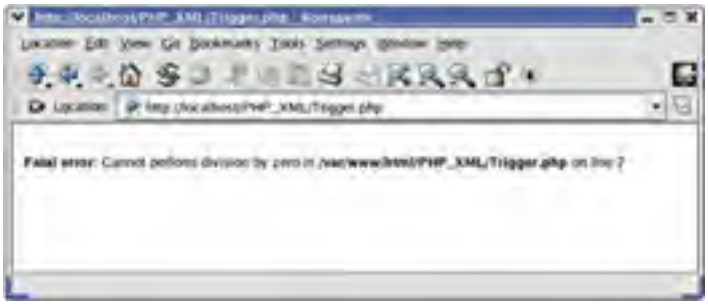


Figure 1-7: Viewing Trigger.php

You can also define your own custom error handlers. You need to define a function that acts as an error handler. The syntax to declare a custom error handling function is:

```
function func_name($errorno, $errorstr [,$errorfile, $errorline, $errorcontext])
```

The above syntax shows how to declare an error handling function. In the above code:

- The func_name represents the name of the error-handling function.
- The function must accept at least two parameters and a maximum of five parameters.
- The first two parameters represent error number and error description. The remaining parameters are optional.
- The third parameter represents the file name in which the error occurred.
- The fourth parameter represents the line number, and the fifth parameter represents the context in which the error occurred.

After defining an error handling function, you need to set it as the error handler using the set_error_handler() function. This function takes the name of the error handling function as a parameter.

Listing 1-8 shows how to use a custom error handler:

Listing 1-8: Using Custom Error Handler

```
<php
error_reporting(E_ALL);
function ErrHandler($errorno, $errorstr, $errorfile, $errorline)
{
   $display = true;
   $notify = false;
   $halt_script = false;
   $error_msg = "<br>The $errorno error is occurring at $errorline in
   $errorfile<br>";
   switch($errorno)
   {
      case E_USER_NOTICE:
      case E_NOTICE:
      $halt_script = false;
      $notify = true;
      $label = "<B>Notice</B>";
      break;
      case E_USER_WARNING:
      case E_WARNING:
      $halt_script = false;
      $notify = true;
      $label = "<b>Warning</b>";
      break;
      case E_USER_ERROR:
      case E_ERROR:
      $label = "<b>Fatal Error</b>";
      $notify=true;
      $halt_script = false;
      break;
      case E_PARSE:
      $label = "<b>Parse Error</b>";
      $notify=true;
      $halt_script = true;
      break;
      default:
      $label = "<b>Unknown Error</b>";
      break;
   }
   if($notify)
    {
      $msg = $label . $error_msg;
      echo $msg;
   }
   if($halt_script) exit -1;
}
$error_handler = set_error_handler("ErrHandler");
echo "<BR><H2>Using Custom Error Handler</h2><BR>";
trigger_error("<BR>Error caused by E_USER_NOTICE</BR>", E_USER_NOTICE);
trigger_error("<BR>Error caused by E_USER_WARNING</BR>", E_USER_WARNING);
trigger_error("<BR>Error caused by E_USER_ERROR</BR>", E_USER_ERROR);
trigger_error("<BR>Error caused by E_PARSE</BR>", E_PARSE);
?>
```

The above listing defines a custom error handler function, ErrHandler(), which handles the error generated by the trigger_error() function. In the above listing:

- The $display variable is a configuration flag that indicates whether to display the error on the Web page or not. The configuration flags are used for the proper functioning of PHP scripts.
- The $notify variable is another configuration flag that specifies whether the error should be notified to the end user or not. The default value for the $notify variable is false.
- The $halt_script configuration variable specifies whether to stop the execution of the PHP script when a fatal error occurs. The default value of the $halt_script variable is true.

- The $error_msg variable represents the error message to be displayed when an error occurs.
- The switch statement execute statements for any error type based on the value of the $error variable.
- The $notify variable displays the error message.
- The ErrHandler() function is set as the error handler using the set_error_handler() function.
- The trigger_error() function triggers the error of each of the error types.

Note The trigger_error() function can only trigger error of the E_USER types. It cannot trigger error for standard error types. Save the above listing as Trigger2.php.

Figure 1-8 shows the output when Trigger2.php is viewed in the Konqueror Web browser:

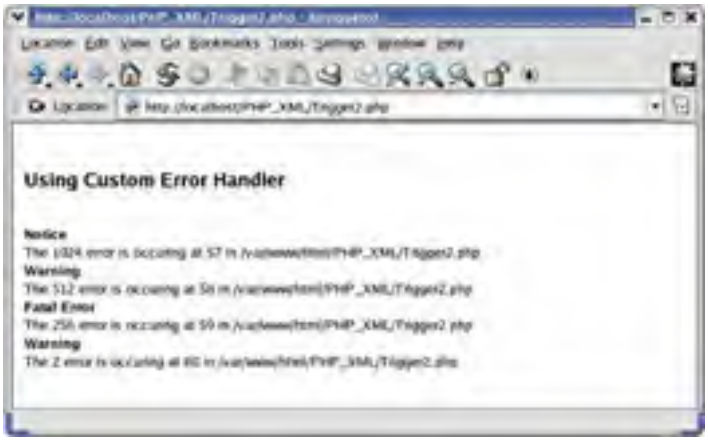


Figure 1-8: Viewing Trigger2.php

Classes and Objects

A class defines the properties and methods of an object. The properties are represented by the variables defined in the class and a method represents a function associated with an object. If student is a class, the variables in the student class represent the attributes of the student object, such as name and age of a student, and a method can be defined to calculate the grade. You can create classes in PHP using the keyword, class. The syntax to define a class is:

Defining a Class

```
<?php
class Class_name
{
   var $variables;
   function class_function($arg1, $arg2)
   {
      // Statements to run within the class function
      $this variables=$arg1;
   }
}
?>
```

The above syntax shows the use of the keyword, class, for creating a class. In the above syntax:

- The Class_name parameter indicates the name of the class that you are creating.
- The $variables parameter indicates the class variable in the class.
- The class_function parameter indicates the name of the class function.
- The keyword, `this`, access the current object properties and methods. Using the this variable, you can access properties to assign values, such as $arg1, to it.

You create objects of a class to use the methods defined in the class. An object is an instance of the class that you use to call the class functions and assign values to the class variables. The syntax to create objects of a class is:

```
$object= new Class_name;
$object->class_function();
```

The above syntax shows the use of the keyword, new, to create an object, $object. The Class_name parameter indicates the name of the class for which you are creating the object. The keyword, this, is used to call the class_function() method of the current class object.

Listing 1-9 shows the process of creating classes and objects in PHP:

Listing 1-9: Using PHP Classes and Objects

```
<?php
class emp
{
   var $name;
   var $address;
   var $dept;
   function assign_info($n,$a,$d)
   {
      $this->name=$n;
      $this->state=$a;
      $this->dept=$d;
   }
   function display_info()
   {
      echo("<p>Employee Name : $this->name");
      echo("<p>State : $this->state");
      echo("<p>Department : $this->dept");
   }
}
$empobj = new emp;
$empobj->assign_info("Angela Jone","California","Accounts");
echo("<center><h2>Displaying Employee Information</h2></center>");
echo("<font size=4>");
$empobj->display_info();
echo("</font>");
?>
```

The above listing defines a class, emp, with the class variables, $name, $state, and $dept. In the above listing:

- The emp class also defines the method, assign_info() and display_info().
- The assign_info() method assigns values to the class variables.
- The display_info() method displays the values assigned to the class variables.

Save the above listing as Classes.php in the PHP_XML folder.

Figure 1-9 shows the output when Classes.php is viewed in the Web browser:

Figure 1-9: Viewing Classes.php

A constructor is a special function in a class that is executed at the time of object creation. You can create a constructor in PHP by defining a function with the same name as the class name.

Listing 1-10 shows how to define a constructor for a class:

Listing 1-10: Creating Constructors with Classes

```
<?php
class student
{
   var $name;
   var $age;
   var $grade;
   function student($n,$a,$g)
   {
      $this->name=$n;
      $this->age=$a;
      $this->grade=$g;
   }
   function display_info()
   {
      echo("<p>Name : $this->name");
      echo("<p>Age : $this->age");
      echo("<p>Grade : $this->grade");
```

```
    }
}
$sobj = new student("Rick Carter","15","A");
echo("<center><h2>Displaying Student Information</h2></center>");
echo("<font size=4>");
$sobj->display_info();
echo("</font>");
?>
```

The above listing defines the constructor for the class, student. In the above listing:

- The constructor, student, is a parameterized constructor because it accepts three arguments, $n, $a, and $g, and assigns them to the variables of the object created of this class.
- The values are passed to the constructor and assigned to the class variables at the time of object initialization.

Save the above listing as Constructor.php in the PHP_XML folder.

Figure 1-10 shows the output when Constructor.php is viewed in the Web browser:

Figure 1-10: Viewing Constructor.php

Working with Files

You can perform the read, write, and append operations on PHP files. You need to open a file before you perform the read and write operations on it. You can open existing files using the PHP function, fopen(). The syntax to use the fopen() function is:

```
$fobj=fopen("FileURL","Mode");
```

In the above syntax:

- The parameter, FileURL, indicates the name of the file that is opened using the fopen() function.
- The parameter, Mode, defines the mode in which you open a file, such as the read, write, and append modes.

Table 1-6 lists the various modes in which you can open a file:

Table 1-6: File Open Modes

Symbol	Mode
Are	Represents read-only mode.
r+	Represents read and write mode.
W	Represents write mode.
w+	Represents read and write mode. Creates a new file if the file does not exist.
A	Represents append mode.
a+	Represents append mode. Creates a new file if the file does not exist.

You can read a file after you have opened it. PHP provides the fread() function to read the content of a file. The syntax of the fread() function is:

```
$text=fread(FileObject, filesize(FileURL));
```

In the above syntax, the parameter, FileObject, represents the name of the file to be read. The filesize() function calculates the size of the file passed to it as parameter, FileURL.

You can write data to a file after you open it in the write or append modes. PHP provides the fputs() function to write data to the file. The syntax to use the fputs() function is:

```
fputs(FileObject, $text);
```

In the above syntax:

- The FileObject parameter represents the file in which data needs to be written.
- The $text parameter indicates the text that is written to the file.

Listing 1-11 shows how to use the file handling functions to display the content of a text file:

Listing 1-11: File Handling Functions

```
<?php
$file='info.txt';
$fobj=fopen($file,"r");
$text=fread($fobj, filesize($file));
echo("<h2><font color='blue'>Displaying Content of a Text File</font><h2>");
echo("<br>");
echo("<font size=3>");
echo($text);
echo("</font>");
fclose($fobj);
?>
```

The above listing shows how to display the content of a text file. In the above listing:

- The fopen() function opens the info.txt file in the read-only mode.
- The fread() function reads the content of the info.txt file.
- The fclose() function closes the file object after the information of the text file is displayed on the Web page.

Save the above listing as File.php in the PHP_XML folder.

Create the info.txt file in the PHP_XML folder. The following code shows the content of info.txt file:

Hello World. This is a test file.

Figure 1-11 shows the output when File.php is viewed in the Web browser:

Figure 1-11: Viewing File.php

PREVIOUS NEXT

Introducing XML

XML is a markup language using which you can store data in a structured format in plain text files, which can be read by many applications. You can reuse data in different types of applications, such as database applications. XML works in combination with HTML to separate data from its presentation. For example, in XML, you can define a name as a combination of first and last name; HTML describes how data should be displayed on the Web page. The documents created in XML are stored with the extension .xml.

In HTML, you cannot use tags other than those that are already defined. Using XML, you can create your own tags.

Structure of XML Document

An XML document consists of user-defined tags. The XML document starts with a processing instruction, <?xml version="1.0">.

The tags that you create in the XML documents are called elements. All XML documents contain a root element, which is the outermost tag in the document. It is mandatory to close all tags that you create in an XML document. XML is case sensitive, and the closing tag must match the opening tag.

Listing 1-12 shows the structure of an XML document:

Listing 1-12: Structure of an XML Document

```
<?xml version="1.0"?>
<Books>
<Book>
<Name>Programming in VC++</Name>
<Author>Stephen Miller</Author>
<Category>Programming</Category>
<Price>50</Price>
</Book>
<Book>
<Name>Designing Websites in Dreamweaver MX</Name>
<Author>Angela Jones</Author>
<Category>Web Design</Category>
<Price>20</Price>
</Book>
</Books>
```

The above listing shows an XML document that stores information on several books. Save the above listing as Books.xml. The <Books> tag is the root element, and the starting and ending element of the document. Other user-defined tags in the listing are:

- `<Title>` tag: Indicates the name of the book.
- `<Author>` tag: Indicates the name of the author.
- `<Category>` tag: Indicates the type of the book.
- `<Price>` tag: Indicates the price of the book.

Figure 1-12 shows the output when you view Books.xml in the Mozilla Web browser:

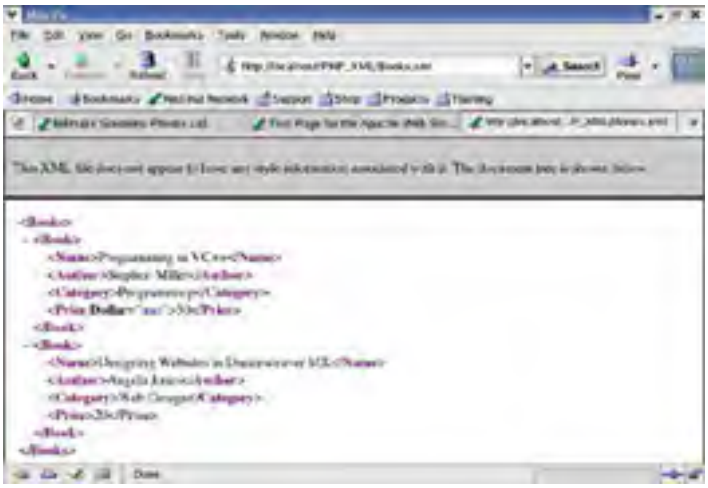


Figure 1-12: Viewing Books.xml

You can also use attributes with XML elements. An attribute provides additional information about the element. For example, you can indicate that the price of the books is in dollars, using an attribute, currency. The following code shows an example of using an attribute in an element:

```
<Price currency="dollar">20</Price>
```

You can also define empty elements in an XML document. An empty element does not contain any text. The following code shows an example of an empty element:

```
<Currency type="dollar"/>
```

You can also insert comments within XML documents. The XML parser does not parse the statements embedded in the comment tags. The following code shows how to insert comments in an XML document:

```
<!--Text for comments-->
```

The comment text is enclosed within the <!-- and --> symbols.

DTD

The rules that an XML document should follow can be defined using DTD. An XML document is valid if it conforms to the rules defined in a DTD document. The declarations in a DTD document include:

- `Element declarations`: Represent the rules for the tags in an XML document.
- `Attribute declarations`: Represent the rules for the attributes in the tags of XML documents.
- `Content declarations`: Represent the rules for text contained in the elements.

You can create two types of DTDs, which are:

- `Internal DTD`: Defines the rules for validating the structure of an XML file, within the XML file.
- `External DTD`: Creates the rules for validating the structure of an XML file in a separate document and stores the file with the extension .dtd.

A DTD begins with the syntax, <!DOCTYPE>, and the rules defining the structure of a document are present within the <!DOCTYPE> tag. The instruction tag, <!ELEMENT>, defines the rules for an element. An example of <!ELEMENT> is:

```
<!ELEMENT Book(Name, Author, Category, Price)>
```

In the above example, the Book element contains four elements, Name, Author, Category, and Price. The <ELEMENT> tag indicates that the <Book> tag must contain the elements, Name, Author, Category, and Price, in an XML document, in the given order.

You can also define the type of content within an element. The XML elements can contain two types of data: Parsed Character Data (PCDATA) and Character Data (CDATA). The XML parser parses PCDATA but does not parses CDATA. You can define the content in an element using the following syntax:

```
<!ELEMENT Name (#PCDATA)>
```

In the above example, the content within the Name element is of PCDATA type.

You declare attribute using the <!ATTLIST> instruction. You use the <!ATTLIST> instruction using the following syntax:

```
<!ATTLIST Price Currency CDATA #REQUIRED>
```

In the above syntax:

- Price is the name of the element for which you define the attribute rules.
- Currency is the name of the attribute for the Price element.
- The CDATA parameter specifies that the attribute value is of character data type.
- The #REQUIRED parameter specifies that it is mandatory to use the Currency attribute with the Price element.

Listing 1-13 shows an internal DTD for an XML document:

Listing 1-13: Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE Books [
<!ELEMENT Books (Book)+>
<!ELEMENT Book (Name, Author, Category, Price)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Price Currency CDATA #REQUIRED> ]>
<Books>
<Book>
<Name>Web Designing in XHTML</Name>
<Author>Angela Jones</Author>
<Category>Web Designing</Category>
<Price Currency="dollar">20</Price>
</Book>
<Book>
<Name>Programming in VC++</Name>
<Author>Stephen Miller</Author>
<Category>Programming</Category>
<Price Currency="dollar">50</Price>
</Book>
</Books>
```

The above listing shows the internal DTD created in an XML file. In the above listing:

- The XML document conforms to the rules defined in the internal DTD.
- The root element is defined as Books, and contains more than one Book element.
- The Book element contains the Name, Author, Category, and Price elements that contain PCDATA.
- The Price element defines the mandatory Currency attribute.

Save the above listing as InternalDTDBooks.xml.

Figure 1-13 shows the output when InternalDTDBooks.xml is viewed in the Mozilla Web browser:

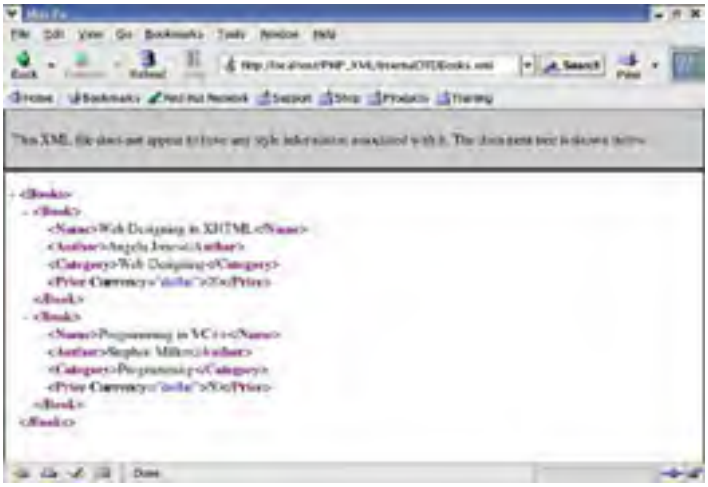


Figure 1-13: Viewing InternalDTDBooks.xml

You can also define an external DTD, where the rules to validate the structure of an XML document are specified outside the XML document. In an external DTD, the rules are defined in a .dtd file, and the reference to the file is provided in the XML file that conforms to the rules of the external DTD.

Listing 1-14 shows an external DTD:

Listing 1-14: The External DTD MyDTD.dtd

```
<!ELEMENT Book (Name, Author, Category, Price)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Category EMPTY>
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Category Ctype CDATA #REQUIRED>
```

The above listing shows an external DTD file. The Book element contains the Name, Author, Category, and Price elements. The Category element is empty and defines an attribute, Ctype, which is mandatory with the Category element.

The <!DOCTYPE> instruction provides a reference to the external DTD document in an XML file.

Listing 1-15 shows the XML file with a reference to the external DTD:

Listing 1-15: Referencing an External DTD

```
<?xml version="1.0"?>
<!DOCTYPE Book "MyDTD.dtd">
<Book>
<Name>Programming in C++</Name>
<Author>Stephen Wright</Author>
<Category Ctype="Programming"/>
<Price>$35</Price>
</Book>
```

The above listing shows how to use the <!DOCTYPE> instruction to provide a reference to the external DTD file, MyDTD.dtd. The structure of the XML document is checked against the rules defined in MyDTD.dtd.

Namespaces

Namespaces are naming conventions that you use to prevent inconsistency between element names. An inconsistency between element names occurs if you use the same name to identify two different elements. You can prevent this inconsistency by assigning a prefix to the name of the elements.

Listing 1-16 shows how to assigning a prefix to an element:

Listing 1-16: Assigning Prefix

```
<doc>
<keyframes>12</keyframes>
<framerate>24</framerate>
<my:frame>
<my:keyframes>100</my:keyframes>
<my:framerate>100</my:framerate>
</my:frame>
</doc>
```

In the above code, you use the my prefix with the frame tag to avoid conflict with the existing HTML tags, <keyframe> and <framerate>. The <my:keyframes> and <my:framerate> tags are encapsulated in </my:frame> tags.

The xmlns attribute is used with a user-defined element, such as <my:frame>, to assign the namespace URI to the element. The syntax of using the xmlns attribute is:

```
<prefix:MyElement xmlns:prefix="NamespaceURI">
```

In the above syntax:

- The prefix parameter represents the name of the prefix that is used with the user-defined element to avoid name conflicts.
- The MyElement parameter represents the name of the user-defined XML element.
- The xmlns:prefix represents the namespace URI. In the xml:prefix attribute, the :prefix parameter represents the prefix that is being used to access the namespace.
- The NamespaceURI parameter is the namespace URI.

Listing 1-17 shows the use of namespaces in an XML file:

Listing 1-17: Namespaces in XML

```
<doc xmlns:document="namespace1"
xmlns:my="namespace2">
<document:keyframes>12</document:keyframes>
<document:framerate>24</document:framerate>
<my:frame>
<my:animationno>1</my:animationno>
<my:keyframes>100</my:keyframes>
<my:framerate>20</my:framerate>
</my:frame>
</doc>
```

The above listing shows how to create a file that contains user-defined XML elements with namespace prefixes applied on the elements.

XML Schemas

An XML schema defines the structure of an XML document and is an alternative to DTD. An XML Schema defines the following:

- Elements that can be used in an XML document.
- Attributes and text that an element can contain.
- Child elements that a parent element can contain, and the order of the child elements in which they appear within a parent element.

The <schema> element represents the root of an XML schema. The syntax to use the <schema> element is:

```
<?xml version="1.0"?>
<xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema"
xmlns="namespace1"
elementFormDefault="qualified">
...
</xsi:schema>
```

In the above syntax:

- The prefix, xsi:, represents the XML namespace.
- Data types, such as schema, element, string, and Boolean, used in the XML schema, are present in the www.w3.org/2001/XMLSchema name space.
- The xmlns attribute specifies the default namespace, which is namespace1.
- The elementFormDefault attribute specifies that the namespace qualifier, xsi, should prefix every element declared in the XML document.

XML schemas are of two types, external and internal. An external schema is saved in a separate file with the extension, .xsd. An internal schema is defined within the same XML file that contains the XML elements. After defining an XML schema namespace prefix, you can use the schemaLocation attribute of the <schema> element to refer to an external schema. The schemaLocation attribute takes two values. The first parameter represents the namespace and the second parameter represents the location of the external schema. For example, if the name of the external schema is external.xsd, then you can refer to it in your XML document using the following code:

```
xsi:schemaLocation=" http://www.w3.org/2001/XMLSchema external.xsd"
```

After beginning the <schema> element, you can define various elements that it can contain. The syntax to define a schema element is:

```
<prefix:element name="xxx" type="yyy" default="value" fixed="value"/>
```

The above syntax shows how to define a schema element. In the above syntax:

- Prefix represents the namespace prefix, such as xsi.
- The name attribute represents the name of the element, and the type attribute represents the data type of the element.
- The default attribute specifies the default value for the element.
- The fixed attribute indicates that the element cannot have any other value except the value specified by the fixed attribute.

Some XML schema data types are: xsi:string, xsi:decimal, xsi:integer, xsi:Boolean, xsi:date, and xsi:time.

Listing 1-18 shows how to use the internal XML schema to define the structure of an XML document:

Listing 1-18: Using Internal XML Schema

```
<?xml version="1.0"?>
<xsi:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema"
targetNamespace="namespace1"
xmlns="namespace2"
elementFormDefault="qualified">
<xsi:element name="Employee">
<xsi:complexType>
<xsi:sequence>
<xsi:element name="id" type="xsi:integer"/>
<xsi:element name="name" type="xsi:string"/>
<xsi:element name="address" type="xsi:string"/>
</xsi:sequence>
</xsi:complexType>
</xsi:element>
<Employee>
<id>1</id>
<name>John Smith</name>
<address>Washington, D.C.</address>
</Employee>
</xsi:schema>
```

The above listing defines an XML schema for the <Employee> element that contains three attributes: id, name, and address. In the above listing:

- The <xsi:complexType> element specifies that the <Employee> element contains other XML elements, such as <Id>, <name>, and <address>.
- The <xsi:sequence> element defines various elements that an <Employee> element contains in the order in which they appear in the XML document.

Save the above listing as EmployeeSchema.xml.

Figure 1-14 shows the output when EmployeeSchema.xml is viewed in the Mozilla Web browser:

Figure 1-14: Viewing EmployeeSchema.xml

Chapter 2: PHP and Simple Application Programming Interface for XML

 Download CD Content

Hypertext Preprocessor (PHP) is a server-side scripting language that runs on various platforms, such as Linux and Windows. You use PHP to read and parse an eXtensible Markup Language (XML) document using the Simple Application Programming Interface (API) for XML (SAX) parser. PHP displays the parsed XML document on a Web browser.

This chapter introduces the parsers, and explains how to implement the SAX parser to parse XML documents. It also describes the object-oriented framework of parsers.

Parsing an XML Document

Parsing validates the structure of an XML document by examining its syntax. A parser validates the structure of an XML document using Document Type Definition (DTD), which specifies the valid format for the syntax of the elements contained in an XML document. The XML parser checks the opening and closing tags within the code of an XML document, against the rules specified in DTD.

Various Web browsers, such as Mozilla, Konqueror, and Internet Explorer, contain built-in parsing capabilities to parse XML documents. To provide better readability and presentation of data, you can use PHP to parse the XML documents. PHP 3.0 or latest version supports XML parsing. To parse the XML document, you need to initialize the XML parser in the PHP code. You can use the data of the parsed XML document in other applications.

Introducing Parser

XML parser is a program that checks whether the XML document is a well-formed document. A document that satisfies the syntax and standard rules specified by the World Wide Web Consortium (W3C) for XML is called a well-formed document. There are two types of XML parsers, which are:

- Non-validating parser: Checks if an XML document is well-formed according to the XML syntax rules.
- Validating parser: Checks if the XML document is well-formed and valid according to the rules specified by DTD.

The parser translates the data of an XML document into platform-specific objects. The non-validating parser adopts the event-driven approach of parsing, and the validating parser adopts the tree-based approach of parsing.

In the event-driven parsing approach, the parser processes XML data and XML tags sequentially in the memory, one at a time. The XML parser generates an event that does parsing, when it finds any XML element or data within the elements of an XML document. An example of the event-driven parser is the SAX parser.

In the tree-based parsing approach, the parser processes and organizes an XML document in a hierarchical form, all at one time. It supports Document Object Model (DOM), which is a platform and language independent model. An example of the tree-based parser is the DOM parser. The DOM parser reads an XML document and divides it into various objects, such as elements, attributes, and comments. DOM creates a tree structure for each element of the XML document and stores the structure in the memory. As a result, the DOM parser performs fast searching of any element in the XML document, as compared to the SAX parser.

Creating a Parser in PHP

You can implement an XML parser by including the xml_parser_create() function in the PHP code. The syntax to create an XML parser is:

```
resource xml_parser_create(string encoding)
```

The above code shows that the xml_parser_create() function creates an XML parser, and specifies that the return type of the function is resource. The resource type returns the resources handled by the XML document, which are used by other functions of the document.

The parameter of the xml_parser_create() function is optional, and provides the character encoding schemes in which the XML document is parsed. The character encoding schemes supported by the XML parsers are: ISO-8859-1, UTF-8, and US-ASCII. UTF stands for Universal Character Set (UCS) Transformation Format. The default character encoding format is ISO-8859-1.

PHP 4.0 or later versions also support the creation of an XML parser with namespace support. The syntax to create an XML parser with namespace support is:

```
resource xml_parser_create_ns(string encoding, string name, string namespace, string delimiter)
```

The above code shows that the xml_parser_create_ns() function creates an XML parser that supports the XML namespaces. The function returns the resources handled by the XML document, which are used by other functions of the document. The xml_parser_create_ns() function consists of two optional parameters, name and namespace of the tag. The string delimiter, provided in the xml_parser_create_ns() function, separates the parameters. The default string delimiter is colon.

The xml_parse_free() function in PHP frees the resources handled by an XML parser, and releases the reference of the XML parser. You should include the function before the PHP script ends. If you do not use the xml_parse_free() function for releasing the resources, either the connection with the Web server closes or the segmentation fault error is displayed. The syntax to release the reference of an XML parser is:

```
bool xml_parser_free(resource parser)
```

In the above code:

- The parser parameter specifies the reference of the XML parser to be released.
- The function returns the Boolean values, true or false. The true value indicates the successful execution of the function, which means that the function frees the reference of an XML parser. The false value indicates the unsuccessful execution of the function, if the provided parameter, parser, is invalid.

Working with Well-formed XML Documents

An XML document starts with the XML declaration statement, which is called Processing Instruction (PI). PI specifies the encoding scheme to process an XML document. The code to use the PI statement in an XML document is:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The above code shows the XML version that you need to use in an XML document. The encoding attribute indicates the encoding scheme that you use for creating the XML document. You use the UTF-8 encoding scheme to create Web pages in English. The UTF-8 encoding scheme uses 8 bits, which are compatible with ASCII, to represent a character. You use the UTF-16 encoding scheme when an application uses languages other than English, such as Japanese, Katana, and Cyrillic. UTF-16 uses 16 bits to represent a character.

A well-formed XML document meets the standards and rules for XML provided by W3C. Rules for creating a well-formed XML document are:

- Ensuring that every element in the XML document must have a start and an end tag. For example you need to provide the starting and the ending tags of an element, as shown:

```
<LI>First</LI>
<LI>Second</LI>
```

- Closing an empty tag with a slash mark (/). Empty tags do not have closing tags because they do not contain data. The <IMG> and
 tags are examples of empty tags in XML, as shown:

```
<IMG src="image.gif" /> <BR />
```

The above code shows that the <IMG> and
 tags are empty tags that contain the slash mark preceding the closing angle bracket. The <IMG> tag contains the src attribute that indicates the source of the image file. The
 tag inserts a new line.

- Using quotation marks to provide attribute values, as shown:

```
<P align="right">
```

The above code shows that the <P> tag contains the align attribute with the value right. The value of the align attribute is enclosed within quotation marks.

- Closing the innermost tag before the outermost tag. For example,

```
<NAME> Michael <AGE> 25 </AGE></NAME>
```

The above code shows that the innermost tag, <AGE> is closed before the outermost tag, <NAME>.

- Matching the case of the starting and the ending tags. XML tags are case-sensitive. The mismatch of the cases of the starting and ending tags generates an error, as shown:

```
<NAME> Michael </name>
```

The above code generates an error because the cases of the starting and ending tags are mismatched. The correct code to use the tags is:

```
<NAME> Michael </NAME>
```

The above code shows that both the starting and ending tags are in the uppercase. You can also specify the starting and ending tags in lowercase.

- Including all other elements within the root element. Listing 2-1 shows that the root element contains all other elements of an XML document:

Listing 2-1: Using the Root Element of an XML Document

```
<EMPLOYEEDETAIL>
<EMPLOYEE>
<NAME>Michael</NAME>
<AGE>25</AGE>
<ADDRESS>New York</ADDRESS>
</EMPLOYEE>
<EMPLOYEE>
<NAME>John</NAME>
<AGE>26</AGE>
<ADDRESS>New Jersey</ADDRESS>
</EMPLOYEE>
</EMPLOYEEDETAIL>
```

The above listing shows that <EMPLOYEEDETAIL> is the root element, which contains other elements of the XML document. The <EMPLOYEEDETAIL> element contains two <EMPLOYEE> elements, which also contain other elements, <NAME>, <AGE>, and <ADDRESS>.

- Providing unique names to the attributes of the elements. You cannot create two attributes with the same name. For example:

```
<TITLE heading="Employee Details" heading="Employee Information">
```

The above code shows that the <TITLE> element contains two attributes with the same name, heading. As a result, the <TITLE> element is not valid in a well-formed XML document.

A document is well-formed if it adheres to all the above rules.

Listing 2-2 shows a well-formed XML document:

Listing 2-2: Creating Well-Formed XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<STUDENTDETAIL>
<STUDENT>
<NAME ID="S001">George</NAME>
<AGE>15</AGE>
<ADDRESS>New York</ADDRESS>
<STANDARD>10</STANDARD>
</STUDENT>
<STUDENT>
<NAME ID="S001">John</NAME>
<AGE>15</AGE>
<ADDRESS>New York</ADDRESS>
<STANDARD>10</STANDARD>
</STUDENT>
</STUDENTDETAIL>
```

In the above listing:

- The XML document is well-formed because it adheres to all the rules of a well-formed document.
- The XML document contains the processing instruction at the starting of the document.
- The root element, <STUDENTDETAIL>, contains other elements.
- All tags are closed and are written in the same case.
- The value of the ID attribute of the <NAME> tag is enclosed within quotation marks.

Introduction to SAX

The SAX parser is an event-based, non-validating parser that reads data from the XML document. The current version of SAX is SAX 2. SAX2 processes documents in a sequential manner. It reads a part of the XML document and generates events when it finds an XML tag. It then reads the next part of the XML document. You can use the SAX parser to modify, query, and write an XML document.

Architecture of the SAX Parser

The SAX parser checks the validity of the structure of an XML document. The SAX parser consists of various handlers that are invoked for each XML tag. The handlers are user-defined functions, which are also called callback functions.

Figure 2-1 shows the architecture of the SAX parser:

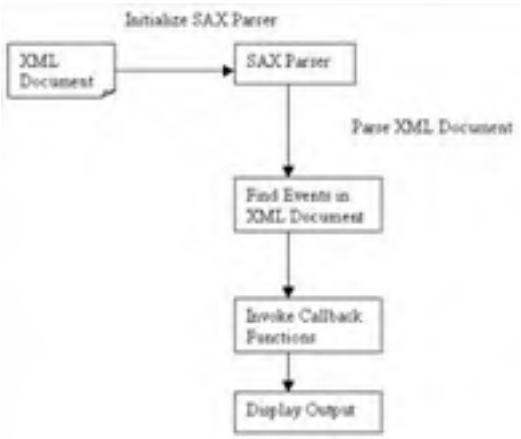


Figure 2-1: Architecture of the SAX Parser

Working with the SAX Parser in PHP

The SAX parser invokes handlers for each opening and closing tag in an XML document. It also invokes handlers for character data and processing instructions of the XML document. To use the SAX parser:

1. Initialize the SAX parser using the PHP function, xml_parser_create(). The code to initialize the SAX parser is:

   ```
   $xparser=xml_parser_create();
   ```

   The above code initializes the SAX parser and creates the xparser variable, which provides a reference to the SAX parser.

2. Identify the events, and set the callback functions to be invoked for the events. The code to identify the events and set the callback functions is:

   ```
   xml_set_element_handler($xparser, "startingHandler", "endingHandler");
   xml_set_character_data_handler($xparser, "cdataHandler");
   ```

   The above code shows that the xml_set_element_handler() function is the built-in function of PHP. In the above code:

   - The xml_set_element_handler() function invokes the callback functions for the opening and closing tags of an XML document.
   - The startingHandler function is the callback function that is invoked when the SAX parser finds an opening tag.
   - The endingHandler function is the callback function that is invoked when the SAX parser finds a closing tag.
   - The xml_set_character_data_handler() function is a built-in function in PHP. The function specifies the callback functions to be invoked for character data within the tags of the XML document.
   - The cdataHandler function is the callback function that is invoked when the SAX parser finds character data within the XML document.

3. Provide the code for the callback functions, startingHandler(), endingHandler(), and cdataHandler(), in the PHP script.

4. Open the XML document using the fopen() function, as shown in the following code:

   ```
   if(!($fp=fopen("student.xml","r")))
   {
      die ("File does not exist");
   }
   ```

   The above code creates the fp variable, which refers to the student.xml file. In the above code:

- The fopen() function opens the student.xml file in the read mode. If the fp variable does not contain the pointer to the XML file, the code displays the error message, File does not exist.
- The die() function is the built-in function of PHP that terminates the execution of the script and displays the message specified as an argument.

5. Parse the XML document using the xml_parse() function, as shown in Listing 2-3:

Listing 2-3: Parsing the XML Document

```
while($data=fread($fp, 4096))
{
 if(!xml_parse($xparser,$data,feof($fp)))
   {
      die("XML parse error: xml_error_string(xml_get_error_code($xparser))");
   }
}
```

In the above code:

- The SAX parser reads the content of the XML document in chunks of 4KB.
- The xml_parse() function parses the XML document until it reaches the end of the XML document.
- The feof() function returns the Boolean value, true, if the end of the document is reached, and notifies the parser to terminate the processing.
- The die() function terminates the execution when an error occurs in parsing the XML document.
- The xml_get_error_code() function returns the error code and the xml_error_string() function returns the error description corresponding to the error code.

6. Release the resources of the XML parser using the xml_parser_free() function of PHP, as shown in the following code:

```
xml_parser_free($xml_parser);
```

The above code releases the XML parser when the execution of the script ends.

Note To initialize the parser with other encoding schemes, use the following code:

```
$xparser=xml_parser_create("UTF-16");
```

Implementing the SAX Parser

The SAX parser consists of various functions, known as handlers. Each handler is invoked when the SAX parser finds certain events, such as opening tag, closing tag, character data, processing instructions, and comments.

For parsing an XML file, you need to provide the XML data file to the SAX parser, as shown:

```
$xfile="student.xml";
```

The above code creates the xfile variable that contains the name of the XML document to be parsed by the SAX parser. You can refer to the student.xml file using the $xfile variable within the PHP script.

To implement the SAX parser, you need to create callback functions for handling all events. PHP passes three parameters to the startingHandler() callback function, which are:

- Reference to SAX parser
- Element name
- Element attributes

Listing 2-4 shows the startingHandler() callback function:

Listing 2-4: Handling the Opening Tag Event

```
function startingHandler($xparser, $element_name, $attributes)
{
   echo "Opening Tag:<b>$element_name</b><br>";
   while (list($key,$value)=each($attributes))
   {
      echo "Attribute:<b><i>$key=$value</i></b><br>";
   }
}
```

In the above listing:

- The parser invokes the startingHandler() function when it finds the opening tag.
- The parser displays the name of opening tags in bold.
- The PHP functions, list() and each(), access the array variables.

- The parser also displays the name and value of the attributes of the tag elements in italics and bold.

Unlike the start tag handler, PHP passes two parameters to the endingHandler() callback function, because it does not contain attributes. The endingHandler() callback function is the end tag handler, which is invoked when the parser finds an end tag. The parameters passed to the end tag handler include the reference to the SAX parser and the element name.

The code to define the endingHandler() callback function, is:

```
function endingHandler($xparser, $element_name)
{
   echo "Closing Tag:<b>$element_name</b><br>";
}
```

The above code shows that the parser invokes the endingHandler() function when it finds the closing tag. It displays the names of the closing tags of the XML document in bold.

PHP passes two parameters to the character data handler. The parameters passed to the character data handler include the reference to the SAX parser and the character data.

The code to define the character data callback function, cdataHandler, is:

```
function cdataHandler($xparser, $cdata)
{
   echo "CDATA: <i><u>$cdata</u></i><br>";
}
```

The above code shows that the cdataHandler() function is invoked when the parser finds text between the opening and closing tags. The cdataHandler() function displays the text between the opening and closing tags in underlined and italicized format.

You can implement the SAX parser in a PHP script, as shown in Listing 2-5:

Listing 2-5: Implementing the SAX Parser

```
<html><head>
<basefont face="Times New Roman">
</head>
<body>
<?php
function startingHandler($xparser, $element_name, $attributes)
{
   echo "Opening Tag:<b>$element_name</b><br>";
   while (list($key,$value)=each($attributes))
   {
      echo "Attribute:<b><i>$key=$value</i></b><br>";
   }
}
function endingHandler($xparser, $element_name)
{
   echo "Closing Tag:<b>$element_name</b><br>";
}
function cdataHandler($xparser, $cdata)
{
   echo "CDATA: <i><u>$cdata</u></i><br>";
}
$xfile="student.xml";
$xparser=xml_parser_create();
xml_set_element_handler($xparser, "startingHandler","endingHandler");
xml_set_character_data_handler($xparser,"cdataHandler");
if(!($fp=fopen($xfile,"r")))
{
   die("File Input/Output error: $xfile");
}
while($data=fread($fp,4096))
{
   if(!xml_parse($xparser,$data,feof($fp)))
   {
      die("XML parser error: xml_error_string(xml_get_error_code($xparser))");
   }
}
xml_parser_free($xparser);
?>
</body>
</html>
```

The above listing shows that the SAX parser of PHP parses the XML document, student.xml. In the above code:

- The $xparser variable indicates the reference to the SAX parser.
- The xml_set_element_handler() function contains the functions, startingHandler() and endingHandler(), for handling the opening and closing tags respectively.
- The xml_set_character_data_handler() function contains the cdataHandler() function for handling the text between the opening and closing tags.

The content of the student.xml file is, as shown:

```
<?xml version="1.0"?>
<studentdata><student><name
id="s001">George</name><age>15</age><address>New
York</address><standard>10</standard></student></studentdata>
```

Note The cdataHandler() function also accepts any white space in the XML file as its parameter.

Figure 2-2 shows the output of Listing 2-5:

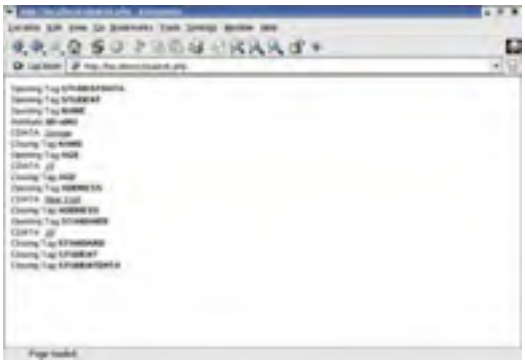


Figure 2-2: Output of Listing 2-5

Using the Expat Parser

The Expat parser is a SAX parser that supports the event-driven approach of parsing a document. The Expat parser is the default parser for the PHP scripting language. This parser contains wrapper classes and filters, which you use to perform advanced processing on XML documents, such as transforming, updating, and querying XML documents.

The PHPXML class is an API that contains wrapper classes, which implement filters in the SAX parser. The class_sax_filters.php file of the SAX parser is an example of a wrapper class.

Note You can download the PHPXML class from http://phpxmlclasses.sourceforge.net/

The AbstractSAXParser class implements the SAX parser. This class checks the XML document and invokes events using the objects of the AbstractFilter class, called listener objects. The methods used by the AbstractSAXParser class are:

- `AbstractSAXParser()`: Creates a parser by passing the XML document as an argument. It is a constructor of the AbstractSAXParser class.
- `Parse()`: Parses the XML document and invokes the StartElementHandler($xml_file, $element_name, $attributes), EndElementHandler($xml_file, $element_name), and CharacterDataHandler($cdata) functions.
- `SetListener()`: Assigns the listeners to a parser object.

The ExpatParser class implements the AbstractSAXParser class to parse an XML document. The constructor of the ExpatParser class accepts the XML document as an argument. The code to implement the ExpatParser class is:

```
$xml_parser=new ExpatParser("file.xml");
$f1=new FilterAddStudent();
$xml_parser->SetListener(f1);
$xml_parser->parse();
```

In the above code:

- file.xml is the name of the XML document, and the xml_parser variable refers to the Expat parser.
- FilterAddStudent is the user-defined class, which acts as a filter that is implemented by the AbstractSAXParser class.
- The SetListener() method accepts the object of the filter class as an argument and sets the filter class as listener of the events generated by the SAX parser.

Filters are user-defined classes that accept SAX events from a parser, process it, and provide the result to other filters or Web browsers. A filter class implements the methods that are defined in the AbstractFilter class. You need to extend the AbstractFilter class to create a user-defined filter. The handlers implemented by a filter class are:

- `StartElementHandler($element_name, $attributes)`: Accepts two arguments, name and attributes, of the element of the starting tag of an XML document. This handler is invoked when a parser finds the starting tag of an element of the XML document.
- `EndElementHandler($element_name)`: Accepts only one argument, because the ending tag of an XML document does not contain attributes. This handler is invoked when a parser finds the ending tag.
- `CharacterDataHandler($cdata)`: Is invoked when a parser finds text within the starting and ending tags.
- `SetListener($object)`: Sets the listener object of the filter. It accepts the object of the filter class as an argument. You need not implement the SetListener() function in the filter class, because the AbstractFilter abstract class already contains the functionality of the SetListener() function.

Filters that do not invoke other events, but display the output on the Web browser, are called finalizers. The FilterOutput() method is a finalizer, which displays the output of the XML document on the Web browser.

Listing 2-6 shows how to implement the AbstractFilter abstract class:

Listing 2-6: Implementing the AbstractFilter Class

```
<?php
include_once("/class_sax_filters.php");
public class FilterAddStudent extends AbstractFilter
{
   function StartElementHandler($element_name, $attributes)
   {
      // Implementation of the start tag handler
   }
   function EndElementHandler($element_name)
   {
      // Implementation of the end tag handler
   }
   function CharacterDataHandler($cdata)
   {
      // Implementation of the character data handler
   }
}
$xml_parser=new ExpatParser("file.xml");
$f1=new FilterAddStudent();
$f2=new FilterOutput();
$f1->SetListener($f2);
$xml_parser->SetListener(f1);
$xml_parser->parse();
?>
```

The above listing shows that the AbstractFilter class implements the FilterAddStudent filter class. The f2 variable refers to the object of the FilterOutput class that displays the output of the XML document on the Web browser.

◄ PREVIOUS NEXT ►

Using SAX to Parse XML Documents

You can parse the XML document using the two methods, SAX and DOM. When you parse the XML document using DOM, it occupies greater memory space to represent data in a tree structure, which results in slow performance. SAX overcomes the drawbacks of the DOM parser, and provides fast and efficient use of memory by reading the document in chunks. The Expat parser implements the SAX parser.

Using SAX to Modify a Document

You can update XML documents by adding elements and attributes, removing elements, and changing the text of the XML document. For example, an XML document contains information pertaining to the students of a school. You need to add information about the students who have enrolled in a particular semester, and remove information about the students who have left the school.

To modify the information in the XML document, you need to create a filter by implementing the AbstractFilter class.

Listing 2-7 shows how to add information pertaining to a student in the XML document, using SAX:

Listing 2-7: Adding Data in XML Document Using SAX

```
<?php
include_once("/class_sax_filters.php");
class FilterAddStudent extends AbstractFilter
{
   var $studentDetail=Array();
   function AddStudent($id, $name, $age, $address, $standard)
   {
      $detail=Array();
      $detail["id"]=$id;
      $detail["name"]=$name;
      $detail["age"]=$age;
      $detail["address"]=$address;
      $detail["standard"]=$standard;
      $this->studentDetail[]=$detail;
   }
   function StartElementHandler($element_name, $attributes)
   {
      $this->listener->StartElementHandler($element_name, $attributes);
      if($element_name=="STUDENTDETAIL")
      {
         foreach($this->studentDetail as $student)
         {
            $this->listener->StartElementHandler("STUDENT", Array("id"
            =>$student["id"]));
            //For student name
            $this->listener->StartElementHandler("NAME", Array());
            $this->listener->CharacterDataHandler($student["name"]);
            $this->listener->EndElementHandler("NAME");
            //For student age
            $this->listener->StartElementHandler("AGE", Array());
            $this->listener->CharacterDataHandler($student["age"]);
            $this->listener->EndElementHandler("AGE");
            //For student address
            $this->listener->StartElementHandler("ADDRESS", Array());
            $this->listener->CharacterDataHandler($student["address"]);
            $this->listener->EndElementHandler("ADDRESS");
            //For student standard
            $this->listener->StartElementHandler("STANDARD", Array());
            $this->listener->CharacterDataHandler($student["standard"]);
            $this->listener->EndElementHandler("STANDARD");
            $this->listener->EndElementHandler("STUDENT");
         }
      }
   }
   function EndElementHandler($element_name)
   {
      $this->listener->EndElementHandler($element_name);
   }
   function CharacterDataHandler($data)
   {
      $this->listener->CharacterDataHandler($data);
   }
}
$f1 = new ExpatParser("student.xml");
$f1->ParserSetOption(XML_OPTION_CASE_FOLDING, 0);
$f2 = new FilterAddStudent();
$f2->AddStudent("S003", "Jack", "14","New York","9");
$f2->AddStudent("S004", "Jim", "14","New York","9");
$f3 = new FilterOutput();
$f2->SetListener($f3);
$f1->SetListener($f2);
$f1->Parse();
?>
```

In the above listing:

- The FilterAddStudent filter is implemented using the AbstractFilter class.
- The FilterAddStudent filter contains the AddStudent() function that adds records of the students to the XML document.
- The AddStudent() function stores the student details in the StudentDetail array, which is accessed by the event handlers to add new student records in the existing XML document.

Listing 2-8 shows the original XML document:

Listing 2-8: Contents of the XML Document

```
<?xml version="1.0" ?>
<studentdata>
<student>
<name id="s001">George</name>
<age>15</age>
<address>New York</address>
<standard>10</standard>
</student>
<student>
<name id="s001">John</name>
<age>15</age>
<address>New York</address>
<standard>10</standard>
</student>
</studentdata>
```

Figure 2-3 shows the output of Listing 2-7:

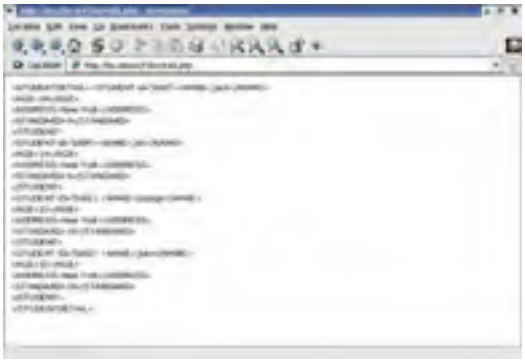


Figure 2-3: Adding Data

Using SAX, you can remove data from the XML document by creating a filter that is implemented by the AbstractFilter class, as shown in Listing 2-9:

Listing 2-9: Removing Data in XML Document Using SAX

```
<?
include_once("/class_sax_filters.php");
class FilterRemoveStudent extends AbstractFilter
{
   var $studentDetail = Array();
   var $flag = 0;
   function RemoveStudent($id)
   {
      $this->studentDetail[] = $id;
   }
   function StartElementHandler($element_name, $attributes)
   {
      if($element_name == "STUDENT")
      {
        if(in_array($attributes["ID"],$this->studentDetail))
         {
            $this->flag = 1;
         }
      }
      if(!$this->flag)
      {
         $this->listener->StartElementHandler($element_name, $attributes);
      }
   }
   function EndElementHandler($element_name)
   {
      if (!$this->flag)
      {
```

```
            $this->listener->EndElementHandler($element_name);
        }
        else
        {
            if($element_name=="STUDENT")
            {
                $this->flag=0;
            }
        }
    }
    function CharacterDataHandler($data)
    {
        if(!$this->flag)
        {
            $this->listener->CharacterDataHandler($data);
        }
    }
}
$fep = new ExpatParser("student.xml");
$frs = new FilterRemoveStudent();
$frs->RemoveStudent("S001");
$fou = new FilterOutput();
$frs->SetListener($fou);
$fep->SetListener($frs);
$fep->Parse();
?>
```

In the above listing:

- The FilterRemoveStudent filter is implemented using the AbstractFilter class.
- The FilterRemoveStudent filter contains the RemoveStudent() function that removes the specified student's records from the XML document.
- The RemoveStudent() function accepts the student id attribute as an argument. The RemoveStudent() function stores student ID in the StudentDetail variable, which is accessed by the event handlers to remove a student record from the existing XML document.

Figure 2-4 shows the output of Listing 2-9:

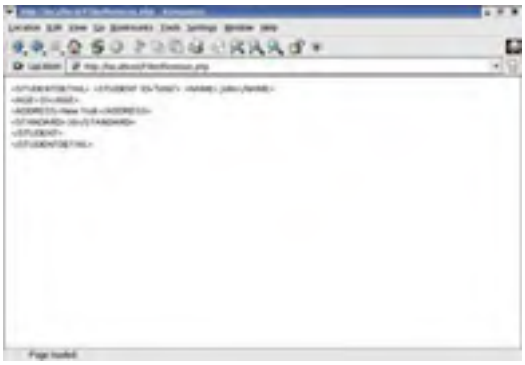


Figure 2-4: Removing Data

Using SAX to Query a Document

Querying a document refers to the process of traversing the XML document and searching for specific information. For example, an XML document contains information pertaining to the students of a school. You can query the XML document to search for the students who live in New York City.

Using SAX, you can query an XML document by creating a filter that is implemented by AbstractFilter, as shown in Listing 2-10:

Listing 2-10: Querying an XML Document Using SAX

```
<?php
include_once("/class_sax_filters.php");
class FilterQueryStudent extends AbstractFilter
{
    var $detail = Array();
    var $stud = Array();
    var $st;
    function RetrieveDetails()
    {
        return $this->detail;
    }
    function StartElementHandler($element_name, $attributes)
    {
        $this->st='';
        if ($element_name <> "STUDENTDETAIL" && $element_name <> "STUDENT")
    {
        $this->st=$element_name;
```

```
      $this->stud[$this->st]='';
   }
}
function EndElementHandler($element_name)
{
   if($element_name == "STUDENT")
   {
      if(((trim($this->stud["ADDRESS"]))=="New York") &&
      (trim($this->stud["AGE"])>=15))
      {
         $this->detail[]=$this->stud;
      }
      $this->stud = Array();
      }
   }
   function CharacterDataHandler($data)
   {
      if($this->st)
      {
         if(!empty($data))
         {
            $this->stud[$this->st] .= $data;
         }
      }
   }
}
$tests=Array();
$f1 = new ExpatParser("student.xml");
$f2 = new FilterQueryStudent();
$f1->SetListener($f2);
$f1->Parse();
$tests=$f1->listener->RetrieveDetails();
$num=count($tests);
for($i=0;$i<$num;$i++)
{
   foreach($tests[$i] as $key => $value)
   {
      print "<b>$key=$value</b><br/>";
   }
}
?>
```

In the above listing:

- The AbstractFilter class implements the FilterQueryStudent filter.
- The FilterQueryStudent filter contains the RetrieveDetails() function, which retrieves the records of students that satisfy the specified condition.
- The EndElementHandler() function checks if the address of the student is New York and the age of the student is greater than or equal to 15.

Figure 2-5 shows the output of Listing 2-10:

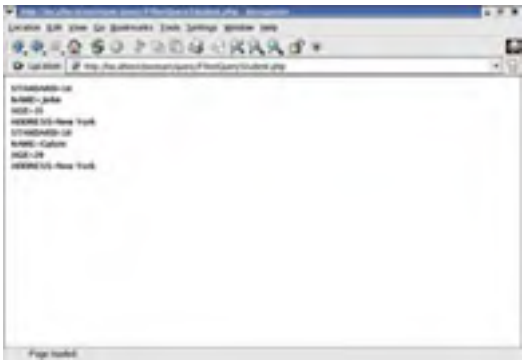


Figure 2-5: Output of Querying a Document

Using SAX to Generate an XML Document

Using SAX, you can generate an XML document from various sources, such as database, text file, and objects. You need to create a parser by implementing the AbstractSAXFilter class for generating the XML document.

For example, you can use the content of the student.txt text file to write an XML document. The student.txt file is shown below:

```
George, English
John, French
Jack, Physics
Jim, French
Jack, English
Jim, Physics
George, Physics
John, Physics
```

Convert the student.txt text file into an XML document, using SAX, as shown in Listing 2-11:

Listing 2-11: Generating XML Data from a Text File

```
<?php
include_once("/class_sax_filters.php");
class StudentParser extends AbstractSAXParser
{
   var $studentDetail = Array();
   function StudentParser($textfile)
   {
      $xml_parser=new ParseStudent();
      $xml_parser->ParseTextFile($textfile);
      $this->studentDetail=$xml_parser->GetStudents();
   }
   function Parse()
   {
      $this->StartElementHandler($this,"STUDENTDETAIL",Array());
      foreach($this->studentDetail as $student => $subjects)
      {
         $this->StartElementHandler($this, "STUDENT",Array());
         $this->StartElementHandler($this, "NAME",Array());
         $this->CharacterDataHandler($this, XmlEntities($student));
         $this->EndElementHandler($this,"NAME");
         $this->StartElementHandler($this, "SUBJECTS",Array());
         foreach ($subjects as $subject)
         {
            $this->StartElementHandler($this, "SUBJECT",Array());
            $this->CharacterDataHandler($this, XmlEntities($subject));
            $this->EndElementHandler($this,"SUBJECT");
         }
         $this->EndElementHandler($this, "SUBJECTS");
         $this->EndElementHandler($this, "STUDENT");
      }
      $this->EndElementHandler($this,"STUDENTDETAIL");
   }
}
// Class that writes the XML data manually.
class ParseStudent
{
   var $studentDetail = Array();
   function ParseTextFile($textfile)
   {
      $fp=fopen($textfile,"r");
      if (!$fp)
      {
         return 0;
      }
      $text=fread($fp, filesize($textfile));
      return $this->ParseText($text);
   }
   function GetStudents()
   {
      return $this->studentDetail;
   }
   function ParseText($text)
   {
      $entries = Array();
      $entry = Array();
      $entries = explode("\n",$text);
      foreach( $entries as $entry)
      {
         $entry = chop($entry);
         $entry = explode(",", $entry);
         if (!isset($this->studentDetail[$entry[0]]))
         {
            $this->studentDetail[$entry[0]]=Array();
         }
         $this->studentDetail[$entry[0]][]=$entry[1];
      }
      return 1;
   }
}
// Function that overwrites the special character with the XML entities.
function XmlEntities($data)
{
   $pos=0;
   $len=strlen($data);
```

```
    $escdata ="";
    for(;$pos<$len;)
    {
       $char=substr($data,$pos,1);
       $num = Ord($char);
       switch($num)
       {
          case 34:
          $char= "";
          break;
          case 38:
          $char = "&";
          break;
          case 39:
          $char = "&apos;";
          break;
          case 60:
          $char = "<";
          break;
          case 62:
          $char = ">";
          break;
          default:
          if ($num < 32)
          $char = ("&#".strval($num).";");
          break;
      }
       $escdata .= $char;
       $pos++;
    }
    return $escdata;
}
$f1=new StudentParser("student.txt");
$f2=new FilterOutput();
$f1->SetListener($f2);
$f1->Parse();
?>
```

In the above listing:

- The parser, StudentParser, parses the student.txt text file, using the ParseStudent parsing class.
- The parsing class reads the text file and stores the content according to the number of lines in the text file.
- The chop() function stores the content of text file in the entry array, and the explode() function splits the text file using a string delimiter.
- The comma delimiter divides each line into two words, and stores these words in the studentDetail array.

Figure 2-6 shows the output of Listing 2-11:

Figure 2-6: Output of Listing 2-11

Note You can create a well-formed XML document by encoding the required functions.

Using SAX to Create PHP Objects from XML

Using SAX, you can create PHP objects from an XML document. The SAX parser creates objects corresponding to each element in the XML document. These objects are represented as PHP objects using the SAX filters, and their references are stored in an array. For example, if an XML document contains information pertaining to students; it can be represented as the student object.

Listing 2-12 shows the content of the XML file that contains information pertaining to students:

Listing 2-12: Content of the XML File

```
<?xml version="1.0"?>
<studentdetail>
<student>
<name>George</name>
<marks>75</marks>
</student>
<student>
<name>John</name>
<marks>85</marks>
</student>
</studentdetail>
```

The above listing shows the elements of the XML document that are converted to the PHP objects.

The parser parses the XML document, finds the <student> tag, creates the student object, and stores its reference in an array. When the parser finds the <name> and <marks> tags, it assigns the value to the name and marks property of the student objects respectively.

Listing 2-13 shows how to create PHP objects from the XML document:

Listing 2-13: Creating PHP Objects from XML Document

```
<?php
include_once("/class_sax_filters.php");
class Student
{
   var $name;
   var $marks;
   function GetName()
   {
      return $this->name;
   }
   function GetMarks()
   {
      return $this->marks;
   }
   function SetName($name)
   {
      $this->name=$name;
   }
   function SetMarks($marks)
   {
      $this->marks=$marks;
   }
}

class FilterNull extends AbstractFilter
{
}
class FilterStudentObject extends AbstractFilter
{
   var $count=0;
   var $student;
   var $sub_element=0;
   var $studentDetail=Array();
   function GetStudents()
   {
      return $this->studentDetail;
   }
   function StartElementHandler($element_name, $attributes)
   {
      if($element_name=="STUDENT")
      {
      $this->studentDetail[]=new Student();
      $this->count=1;
      }
      else
      {
         if($this->count)
         {
            $this->sub_element=$element_name;
         }
      }
      $this->listener->StartElementHandler($element_name, $attributes);
   }
   function EndElementHandler($element_name)
   {
      $this->sub_element=0;
      if($element_name == "STUDENT")
      {
         $this->count=0;
      }
      $this->listener->EndElementHandler($element_name);
   }
```

```
    function CharacterDataHandler($cdata)
    {
       if($this->count && $this->sub_element)
       {
        $getMethod="get".strtoupper(substr($this->sub_element,0,1)).substr
        ($this->sub_element,1);($this->sub_element,0,1)).substr($this->sub_element,1);
        $sub=$this->studentDetail[count($this->studentDetail)-1]->$getMethod();
        $this->studentDetail[count($this->studentDetail)-1]->$setMethod($sub.$cdata);
       }
       $this->listener->CharacterDataHandler($cdata);
    }
}
$f1=new ExpatParser("student.xml");
$f2=new FilterStudentObject();
$f3=new FilterNull();
$f2->SetListener($f3);
$f1->SetListener($f2);
$f1->Parse();
$studentDetail=$f1->listener->GetStudents();
print_r($studentDetail);
$num=count($studentDetail);
print "<br/>";
for($i=0;$i<$num;$i++)
{
   foreach($studentDetail[$i] as $key => $value)
   {
      print "<b>$key=$value</b></br/>";
   }
}
?>
```

In the above listing:

- The parser creates the objects of the student element and stores them in the studentDetail array.
- The Student class creates the functions for retrieving the XML data. The StudentCreateObject filter parses the XML document and creates the objects corresponding to each element.
- The FilterNull class is an empty user-defined class that does not contain any function. This class specifies that the output of the StudentCreateObject filter is not displayed in the XML format.

Figure 2-7 shows the output of Listing 2-13:

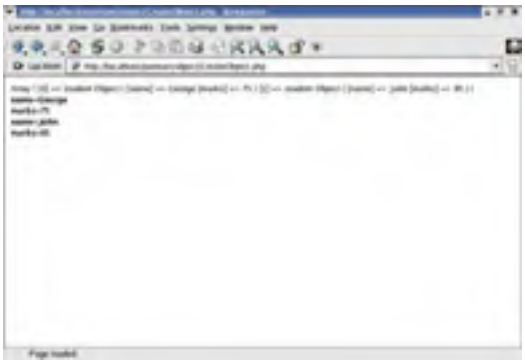


Figure 2-7: Creating PHP Objects

Note The print_r() function displays how the array elements are stored in the array.

◄ PREVIOUS NEXT ►

Using Object-Oriented Frameworks

You can create Web applications using the object-oriented framework in PHP. PHP supports object-oriented programming using classes and objects. In the object-oriented approach, you can use inheritance to change the functionality of the existing class. There are two types of object-oriented frameworks that PHP script can use, extremePHP and SAXParser.

The extremePHP Framework

eXtremePHP (xpl) is a set of object-oriented libraries that helps you create dynamic Web applications. eXtremePHP provides object-oriented framework to implement SAXParser. eXtremePHP supports all features of object-oriented programming, and provides functionality, such as:

- Input/output streams and socket connections
- xpl framework that simplifies the use of HTML tags
- HTML tag classes, form handling, and layout managers
- Utility classes, such as Strings, Dates, Vectors, and Iterators
- Database access
- XML parsing with SAX and DOM processor
- Portable Document Format (PDF) library that supports XML

The minimum requirement for installing the eXtremePHP framework is a Web server running on PHP4. You can download the current version of the framework from the Web site, http://sourceforge.net/projects/extremephp/. The Web site contains the zip version and tar ball of the software.

To implement the eXtremePHP framework in the PHP script, you need to install the eXtremePHP framework in a Linux environment. To install the eXtremePHP framework:

1. Copy the tar file in the document root directory of the Apache Web server using the command:
   ```
   cp eXtremePHP-0.15a.tar.gz /var/www/html
   ```
   The default document root directory of the Apache Web server is /var/www/html.
2. Change the current directory to the document root directory using the command:
   ```
   cd /var/www/html
   ```
3. Untar the tar file of the eXtremePHP framework using the command:
   ```
   gunzip eXtremePHP-0.15a.tar.gz
   tar xvf eXtremePHP-0.15a.tar
   ```
   The above command uncompresses the tar file and creates the xpl directory under the document root directory of the Web server. The xpl directory stores all the class libraries and frameworks of eXtremePHP.
4. Include the /xpl/common/common.inc.php file in the /etc/php.ini file by entering the following command:
   ```
   auto_prepend_file="/var/www/html/common/common.inc.php"
   ```
   The above command automatically includes the common.inc.php file in a PHP document.
5. Create an .htaccess file in the /var/www/html directory, and store the following command in this file:
   ```
   php_value auto_prepend_file /www/var/html/xpl/common/common.inc.php
   ```

The SaxParser Framework

The SAXParser framework helps you create an object-oriented parser to read an XML document and display them in Web browser when the eXtremePHP framework is installed. The SAXParser class of the SAXParser framework provides three functions, StartingTagHandler, EndingTagHandler, and CharacterDataHandler, to handle an XML document. The SAXParser class invokes these functions while parsing the document.

Figure 2-8 shows the SAXParser framework:

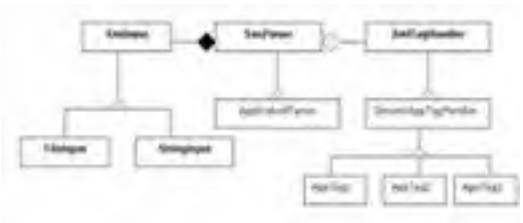


Figure 2-8: The SAXParser Framework

To create a SAX parser, you need to define two new classes, the ApplicationParser class and the GenericAppTagHandler class. The ApplicationParser class inherits the SAXParser class and the GenericAppTagHandler class inherits the XmlTagHandler class. The GenericAppTagHandler class adds tag handlers to the parser, and provides interfaces to global variables required by the tag handlers.

The SAXParser class is based on the white box model and should not be used as a standalone class. The SAXParser class determines the object to be used when handling parsed data. This object-oriented approach offers the advantage of grouping tag-specific logic in a single component. You need to create a base tag handler that is inherited by other application element handlers for grouping them in a single component.

Note The white box model deals with the internal working and the functionality of the SAXParser class.

The XmlTagHandler class contains the common functionality between various handler classes, such as GenericAppTagHandler class and its subclasses. For example, all the element handlers provide HTML tags as the output during the begin and end element handlers. The constructor of the SAX parser accepts a reference of the XMLInput class that specifies the name of the file to be parsed. The XMLInput class specifies the source of the input, which can be a file or string.

You can override the constructor of the SAXParser class by defining a new functionality to the constructor. You need to pass the object of the XMLInput class as an argument to the constructor for defining the new functionality. The code to create the constructor is:

```
function StudentParser($file)
{
   SAXParser::SAXParser(new FileInput($file));
}
```

The above code shows that the constructor of the base class is overridden in the constructor of the derived class.

You can define the handlers using the hook up methods, which the parser invokes automatically. The addHandlers() method is the hook up method of the SAXParser class. A derived class of the SAXParser class implements the hook up method.

If you create a new handler, it must inherit the XmlTagHandler class defined in the API of the SAXParser framework.

PREVIOUS NEXT

Chapter 3: PHP and Document Object Model

Download CD Content

Using the tree-based Document Object Model (DOM) approach to parse XML data, PHP loads the complete XML document in the system memory and provides standard classes to navigate and manipulate the XML document. The DOM implementation in PHP is based on the libxml library, which contains various functions that provide parsing capabilities to modify the XML document.

This chapter explains how to implement DOM in PHP to parse XML documents. This chapter also provides a comparison between the event-based, Simple API for XML (SAX) approach, and the tree-based DOM approach to parse XML documents. Finally, this chapter explains the DOM architecture, standard DOM classes, and the uses of the DOM approach to modify and query XML documents.

Introducing DOM

DOM is a programming interface that you can use to create and edit XML documents. It specifies the logical structure of the document that you can use to add, delete, or modify XML documents.

DOM parses the XML document by creating a hierarchical tree structure of standard objects. Each object encapsulates the methods and properties that you can use to manipulate and navigate the DOM tree. For example, the DomAttribute object encapsulates attribute properties, such as name and value. The basic interface in every DOM object is a node. The tree structure contains multiple nodes that are related to each other in a parent-child relationship.

DOM is platform-independent, and can interact with both HTML and XML documents. You can implement the DOM approach with languages, such as PHP, Java, Python, Visual Basic, Perl, and Delphi.

The W3C has defined DOM specifications in multiple levels. The various levels of DOM specifications are:

- DOM Level 1: Contains the core features of DOM developed by the W3C in October 1998.
- DOM Level 2: Contains DOM features, such as core functions, document traversal, and event handling.
- DOM Level 3: Incorporates features, such as XPath and abstract schemas.

A DOM parser reads the entire XML document into memory, converts it into a hierarchical tree structure, and provides an API to access tree nodes and the contents attached to each node. For example, the DOM parser can represent the contents of emp.xml that stores employee information, such as company name and department name in a hierarchical structure.

Listing 3-1 shows how to create the emp.xml document:

Listing 3-1: Creating an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<Company name="Unique Systems">
<Department>Marketing</Department>
<Level>Middle-Level</Level>
<EmployeeInformation>
<Name first="John" middle="E" last="Williams"/>
<DateOfHiring>10/01/1982</DateOfHiring>
</EmployeeInformation>
</Company>
```

The above listing creates an XML document, emp.xml, which stores employee information, such as employee names, company names, department names, and hiring dates.

The DOM tree contains various nodes interlinked to each other with the parent-child relationship. For example, in the emp.xml document, the Marketing node is the child node of the Department node, and Name is the child node of the Employee Information node. You can parse this XML document into a hierarchical structure using the DOM parser.

Figure 3-1 shows a representation of the emp.xml file as a DOM tree structure generated by the DOM parser:

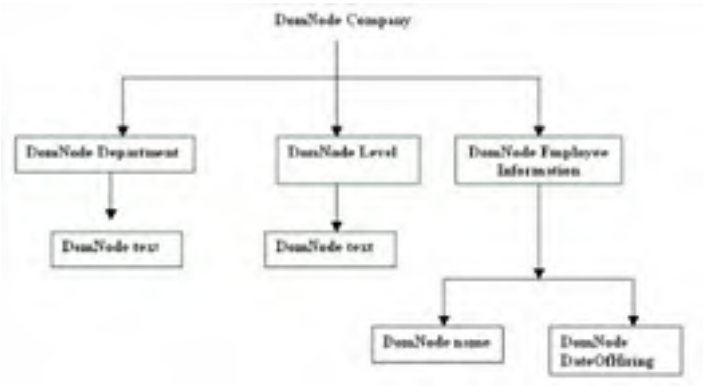


Figure 3-1: DOM Tree Representation of emp.xml

Comparing DOM with SAX

You can parse an XML document using both the DOM and SAX parsers. Both the DOM and SAX approaches have their advantages and disadvantages when working with XML documents.

Table 3-1 lists comparisons between the SAX and DOM parsing approaches:

Table 3-1: Comparison between DOM and SAX

Mode of Comparison	DOM	SAX
Approach	Uses a tree-based approach to parse XML documents. The DOM approach loads the whole XML document in the memory as a tree structure that contains various nodes.	Uses an event-based approach that processes the XML files in a linear manner. The SAX approach parses the XML documents in chunks without creating a DOM tree.
Memory	Consumes system memory as the DOM parser loads the whole XML document in the memory.	Consumes less system memory in comparison to DOM, as it does not load the whole XML structure in memory and parses the XML document in chunks.
Efficiency	Lets you parse XML documents in a non-sequential manner and manipulate complex XML documents, as the DOM parser stores information about how the nodes of a tree are related to each other.	Faster than the DOM approach in parsing simple XML documents, as it processes the XML documents in a sequential manner

The DOM Architecture

The DOM architecture contains modules, where each module defines various DOM API domains, such as XML, HTML, tree events, and Cascading Style Sheets (CSS).

Figure 3-2 shows the block diagram of the DOM architecture:

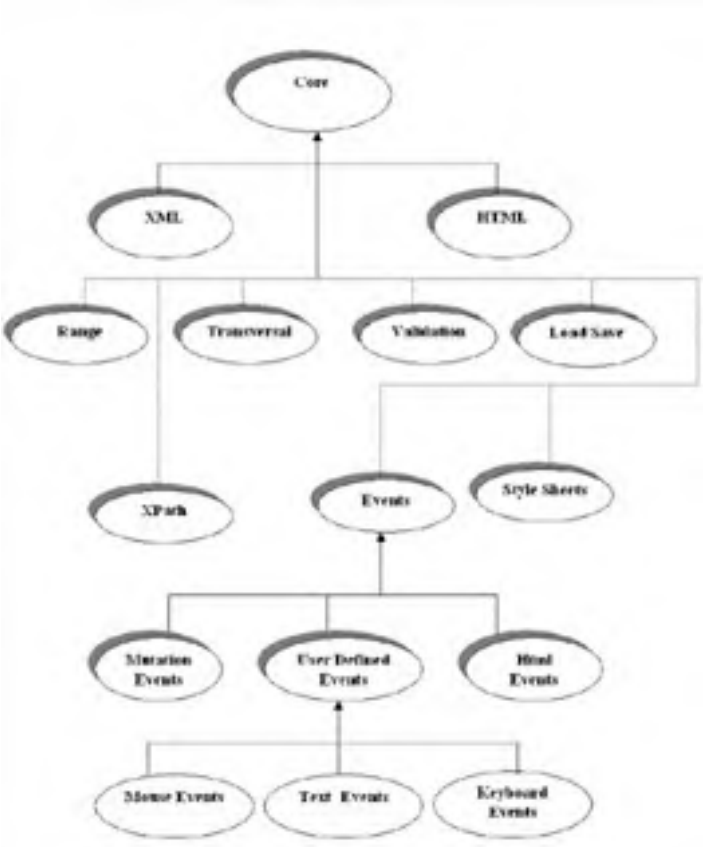


Figure 3-2: The DOM Architecture

The various modules in the DOM architecture are:

- Core: Represents the internal tree-like structure of the document and enables you to move through the hierarchy of the DOM tree elements.

- XML: Provides interface-processing instructions to the DOM parser, such as entities and CDATA.
- HTML: Manipulates HTML documents.
- Events: Defines events that perform XML-tree manipulation.
- Cascading Style Sheets: Manipulates CSS style sheets.
- Load and Save: Provides various parameters that control the load and save operations of XML documents. The load module lets you load the XML document into the DOM tree and save the DOM tree into an XML document.
- Validation: Provides various methods, such as cansetAttribute and canInsertAttribute, which provide validation checks on the DOM documents.
- Xpath: Provides methods to query a DOM tree.

◄ PREVIOUS NEXT ►

Using DOM Classes

DOM provides various built-in classes that let you parse the XML documents in PHP. The PHP DOM parser represents the XML document by creating standard objects that are instances of the following classes:

- DomDocument Class
- DomNode Class
- DomElement Class
- DomText Class
- DomAttribute Class

You can add, delete, or modify the structured content of the XML document by using various properties and methods. The DomDocument object stores the properties of an XML document, such as its name and version.

Listing 3-2 shows the contents of the DOM.xml file:

Listing 3-2: Properties of an XML File

```
DomDocument Object
(
    [name] => #document
    [url] => DOM.xml
    [version] => 1.0
    [encoding] => UTF-8
    [standalone] => -1
    [type] => 9
    [compression] => -1
    [charset] => 1
    [0] => 1
    [1] => 136451608
    [doc] => Resource id #0
)
```

The above listing specifies the values of various properties of the DOM.xml file. You can display the values of the properties of the XML document using the print_r() function. The listing specifies values, such as the version, node type, and character set of the XML document. It also specifies whether the DOM.xml file is compressed or not.

The DomDocument Class

The DomDocument object is the object created after an XML document is created by the DOM parser.

Listing 3-3 shows the structure of the DomDocument class:

Listing 3-3: Structure of the DomDocument Class

```
class DomDocument
{
   Properties:
   version
   encoding
   standalone
   type
   Methods:
   root()
   children()
   add_root( $node )
   dtd()
   dumpmem()
}
```

The above listing shows the structure of the DomDocument class, which contains various properties and methods defined by the DomDocument class.

The properties defined in the DomDocument class specify the version of the XML document, the text encoding, and the type of the document. The standalone property of the DomDocument object specifies the Boolean value whether or not the document is a standalone document. The methods provided by the DomDocument class are:

- `root()`: Returns the root element.
- `children()`: Returns the child node of the document.
- `addroot()`: Creates a new document element and returns the DomElement object.
- `dtd()`: Returns the Document Type Definition (DTD) object. The DTD object contains the basic information of the document and encapsulates properties, such as systemId and name. The systemId property contains the file name of the DTD document.

- `dumpmem()`: Stores the XML structure in a string variable.

Listing 3-4 shows how to parse an XML string using the xmldoc() function:

Listing 3-4: Parsing the XML string

```
<?php
$xml_str = "<?xml version='1.0'?><name>John</name>";
if (!$doc=xmldoc($xml_str))
{
  die("Error in XML");
}
else if ($doc->version > 1.0)
{
   die("Unsupported XML version");
}
else
{
   echo "No Error";
}
?>
```

The above listing parses the string, John, using the xmldoc() function. The xmldoc() function accepts the XML string as an argument and generates the DOM object that represents the XML data.

You can also create the DomDocument object of an XML file. The syntax to create the DomDocument object using the xmldocfile() function is:

```
$doc = xmldocfile("<filename>");
```

In the above code, the xmldocfile() function creates an object of the DomDocument class that represents the XML file.

Listing 3-5 shows how to parse an XML file using the xmldocfile() function:

Listing 3-5: Parsing XML File

```
<?php
$xml_str = "/var/www/html/vishi/test.xml";
if (!$doc = xmldocfile($xml_str))
{
   die("Error in XML");
}
else if ($doc->version > 1.0)
{
   die("Unsupported XML version");
}
else
{
   echo "No Error in Parsing the XML file";
}
?>
```

The above listing parses the xml file, test.xml, using the xmldocfile() function.

The DomNode Class

The DomNode class contains properties, such as name, content, and type of the node. The name property specifies the tag name of the node. The content property specifies the contents stored in the node. The type property represents the integer that refers to the object type. The DomNode class also encapsulates various methods, such as the lastchild(), children(), and parent() methods.

Listing 3-6 shows the structure of the DomNode class:

Listing 3-6: Structure of the DomNode Class

```
class DomNode
{
   properties:
   name
   content
   type
   methods:
   lastchild()
   children()
   parent()
   new_child( $name,$content )
   getattr( $name )
   setattr( $name,$value )
   attributes()
}
```

In the above listing, the structure of the DomNode class is defined. The listing defines various properties and methods that are encapsulated in the DomNode class. The methods defined in the DomNode class are:

- `lastchild()`: Returns the last child node for the node.

- `parent()`: Returns the parent node.
- `children()`: Returns an array of child nodes.
- `new_child()`: Adds a new DomNode to its children.
- `attributes()`: Returns the array of the DomAttribute objects.
- `getattr()`: Retrieves the attributes from the XML document.
- `setattr()`: Sets the attribute value.

The DomElement Class

The DomElement class defines elements of the XML document. A DomElement object represents the name and type of an element of the XML document. The methods contained in the DomElement class are:

- `tagname()`: Returns the element name. The syntax is:
  ```
  string DomElementName->tagname(void);
  ```
- `get_attribute()`: Returns the name and value of the attribute in the element. If no attribute is specified with the method, it returns an empty string. The syntax of the get_attribute() method is:
  ```
  objectDomElement ->get_attribute(string);
  ```
- `set_attribute()`: Adds a new attribute to the node. The syntax of the set_attribute() method is:
  ```
  bool DomElement->set_attribute ( string name, string value)
  ```
- `remove_attribute()`: Removes an attribute from the element structure. The syntax of the remove_attribute() method is:
  ```
  bool DomElement->remove_attribute (string);
  ```
- `children()`: Returns an array of the DomElement object containing the child nodes of the element.
- `parent()`: Returns a DomElement object that is the parent of a node.
- `last_child()`: Returns the last entry of a node child. If no last child is found in the DOM tree, a NULL value is returned.
- `attributes()`: Returns an array of the DomAttribute objects that represent the attributes of a node.

You can distinguish the node type using the type property that contains specific integer values for each node.

Table 3-2 lists various predefined node types that identify the nodes:

Table 3-2: DOM Node Types

Node Type	Specification	Integer Value
XML_ELEMENT_NODE	Specifies an element.	1
XML_ATTRIBUTE_NODE	Specifies an attribute.	2
XML_TEXT_NODE	Specifies text.	3
XML_ENTITY_REF_NODE	Represents an entity reference.	5
XML_PI_NODE	Represents a processing instruction.	7
XML_COMMENT_NODE	Represents a comment.	8
XML_DOCUMENT_NODE	Represents the XML document.	9
XML_NOTATION_NODE	Represents the notation node.	12
XML_CDATA_SECTION_NODE	Represents the cdata section.	4

You can identify the elements of the XML documents using the methods of the DomElement class. You can also count the number of nodes or elements contained in an XML document.

Listing 3-7 shows how to create an XML file that stores employee information:

Listing 3-7: Creating an XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<EMPLOYEEINFORMATION>
<EMPLOYEE>
<NAME ID="E001">George</NAME>
<AGE>35</AGE>
<DEPARTMENT>RESEARCH AND DEVELOPMENT</DEPARTMENT>
<DESIGNATION>BRANCH MANAGER</DESIGNATION>
</EMPLOYEE>
<EMPLOYEE>
<NAME ID="E002">John</NAME>
<AGE>45</AGE>
<DEPARTMENT>HUMAN RESOURCE</DEPARTMENT>
<DESIGNATION>MANAGER</DESIGNATION>
</EMPLOYEE>
```

```
</EMPLOYEEINFORMATION>
```

The above listing shows how to create an XML file containing employee information. Employee information, such as employee name, employee ID, and employee designation, is stored in the emp.xml file. You can generate the DOM tree using the DOM parser that represents the structure of the XML document.

Listing 3-8 shows how to use the classElement objects to generate the structure of the emp.xml file:

Listing 3-8: Structuring an XML Document

```
<?php
$xmlfile = "emp.xml";
if (!$doc = xmldocfile($xmlfile))
{
   die("Error in XML document");
}
$root = $doc->root();
$children = getChildren($root);
$count = 1;
printTree($children);
function printTree($nodeCollection)
{
   global $count;
   echo "<ul>";
   for ($t=0; $t<sizeof($nodeCollection); $t++)
   {
      $count++;
      echo "<li>". $nodeCollection[$t]->tagname;
      $nextCollection = getChildren($nodeCollection[$t]);
      printTree($nextCollection);
   }
   echo "</ul>";
}
function getChildren($node)
{
   $temp1 = $node->children();
   $store = array();
   for ($t=0; $t<sizeof($temp1); $t++)
   {
      if ($temp1[$t]->type == XML_ELEMENT_NODE)
      {
         $store[] = $temp1[$t];
      }
   }
   return $store;
}
echo "Total number of elements in document: $count";
?>
```

In the above listing, the DOM tree of the emp.xml file is generated. The root node is obtained using the root() method. After retrieving the root node of the document, the listing generates other node elements and attributes. The total number of elements in the XML document is also calculated in the above listing. Figure 3-3 shows the output of Listing 3-8:

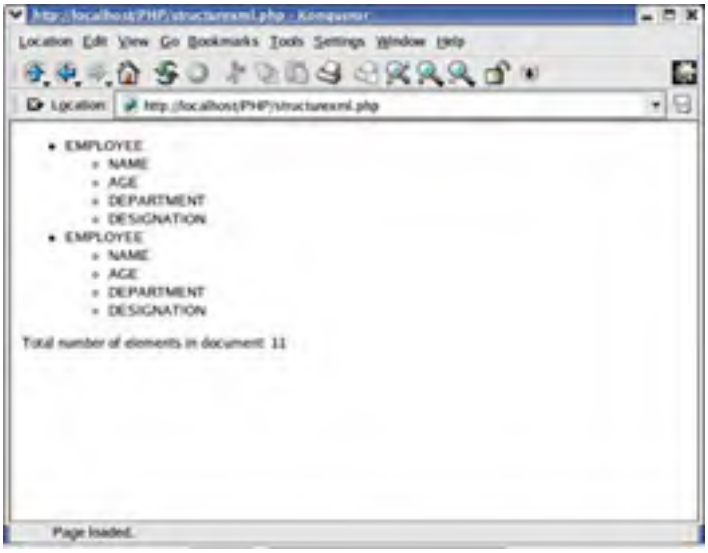


Figure 3-3: The DOM Tree

The DomText Class

You can represent the character data in an XML document using the DomText class. The syntax of the DomText class is:

```
DomText Object
{
   [type]
   [content];
}
```

In the above syntax, the type property specifies the node type, such as XML_TEXT_NODE. You can specify either the node number or the node name with the type. The content property stores the character data. The DOM parser identifies the whitespaces as character data and creates DomText objects for the whitespaces.

Listing 3-9 shows how to retrieve the character data from an XML document using the DomText object:

Listing 3-9: Using the DomText Object

```
<?php
$xml_str = "<?xml version='1.0'?>
<STUDENT>
<NAME>John</NAME>
<NAME>Ronan</NAME>
<ID>OO2</ID>
<NAME>Daniel</NAME>
<ID>003</ID>
<NAME>Mark</NAME>
<ID>004</ID>
<NAME>William</NAME>
<ID>005</ID>
</STUDENT>
";
$data = array();
if(!$doc = xmldoc($xml_str))
{
 die("Error parsing XML");
}
$root = $doc->root();
$nodes = $root->children();
foreach ($nodes as $t)
{
   $text = $t->children();
   if($text[0]->type ==XML_TEXT_NODE)
   {
       if ($text[0]->content != "")
       {
         $data[] = $text[0]->content;
       }
   }
}
print_r($data);
?>
```

The above listing retrieves the text data from an XML document. The DOM parser creates an array to hold the name and the ID of the students. The foreach loop iterates the <STUDENT> elements and adds the character data found while traversing through the DOM tree in the, $data array. Figure 3-4 shows the output of Listing 3-9:

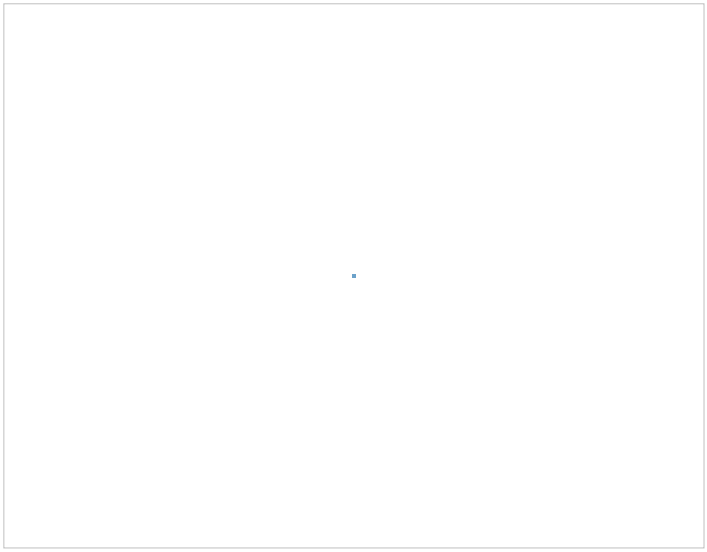


Figure 3-4: The Retrieved Character Data

The DomAttribute Class

The DomAttribute class creates the attr object that returns the attribute of an element.

Listing 3-10 shows the structure of the DomAttribute class:

Listing 3-10: Structure of the DomAttribute Class

```
class DomAttribute
{
   Properties
   name
   value
   Method
   name()
}
```

The above listing specifies the method and properties of the DomAttribute class. The name property sets the attribute name. The value property specifies the value stored in the attribute node.

Listing 3-11 explains the DomAttribute class:

Listing 3-11: Using the DomAttribute Class

```
<?php
$xml_string = "<?xml version='1.0'?>
<EmployeeInfo>
<NAME ID='E001'>George</NAME>,
<NAME ID='E002'>John</NAME>,
<NAME ID='E003'>Angel</NAME>
</EmployeeInfo>";
if (!$doc = xmldoc($xml_string))
{
   die("Error in XML document");
}
// get the root node
$root = $doc->root();
// get its children
$children = $root->children();
// Iterate through child list
for ($x=0; $x<sizeof($children); $x++)
{
   if($children[$x]->type == XML_ELEMENT_NODE)
   {
      // Retrieving the text
      $text = $children[$x]->children();
      $cdata = $text[0]->content;
      if($children[$x]->get_attribute("ID"))
      {
         echo "<br><Employee=" . $children[$x]->get_attribute("ID").">" .
         $cdata .
         "</Employee></br>";
      }
      else
      {
         echo $cdata;
      }
   }
   // if text node
   else if ($children[$x]->type == XML_TEXT_NODE)
   {
      // simply print the content
      echo $children[$x]->content;
   }
}
?>
```

The above listing shows how to implement the DomAttribute class. In the above listing the get_attribute() function retrieves the attribute value of the attribute nodes of the DOM tree. The above listing parses the XML document and then identifies the root and child nodes of the XML document. The DOM parser iterates through the child nodes and verifies the presence of the element node in the DOM tree. The listing retrieves the text node and the contents stored in it. Figure 3-5 shows the output of Listing 3-11:

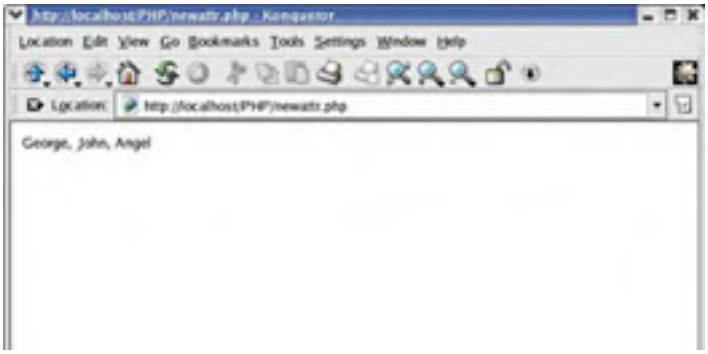


Figure 3-5: Output of using the DomAttribute Class

PREVIOUS NEXT

Using DOM to Manage XML Documents

Using DOM you can manage an XML document by creating a DOM tree of the XML document containing standard objects. You can modify and traverse the DOM tree, and access the attributes, data, and elements of the DOM tree. You can modify the XML document by modifying the properties and methods of the objects. The DOM approach also lets you query XML documents to retrieve specific information. In addition to parsing the XML documents, the DOM approach lets you create XML documents from scratch. Methods such as add_root() and new_child() let you create nodes of DOM tree.

Using DOM to Modify a Document

You can modify the XML document using the DOM approach. Modifying a document includes various tasks, such as adding a new element, modifying the attribute value, or removing an element from the document tree. The DOM approach occupies a large amount of memory to parse documents. As a result, you can modify an XML file which is smaller in size using the DOM approach and the large XML documents using the SAX approach.

To understand how to modify XML documents using DOM, consider an XML file that stores information about a bookstore-shopping cart application, as shown in Listing 3-12:

Listing 3-12: Creating the bookcart.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<Bookcart>
<customer CID="354">
<name>John Anderson</name>
</customer>
<BookInfo>
<book ID="1001">
<bookName>XML Bible</bookName>
<author>Elliotte Rusty Harold</author>
<quantity>2</quantity>
<list_price>33</list_price>
</book>
<book ID="1002">
<bookName>XML in a Nutshell</bookName>
<author>Elliotte Rusty Harold</author>
<quantity>3</quantity>
<list_price>37</list_price>
</book>
<book ID="1003">
<bookName>Advanced PHP Programming</bookName>
<author>George Schlossnagle</author>
<quantity>5</quantity>
<list_price>40</list_price>
</book>
</BookInfo>
</Bookcart>
```

The above listing creates an XML file that stores information about the books that are available at an online bookstore. The XML file stores information, such as book titles, book IDs, author names, quantities available, and list prices.

Listing 3-13 shows how to add information on books available in the bookcart.xml file using the DOM approach:

Listing 3-13: Adding Book to the Shopping Cart

```
<?php
$xmlfile = "bookcart.xml";
$modfile = AddProduct($xmlfile, "1004", "XML and PHP", "William", "4", "30");
print($modfile);
function AddProduct($xml, $ID, $bookName, $author, $quantity, $list_price)
{
   $doc = xmldocfile($xml);
   $root = $doc->root();
   $children = $root->children();
   foreach ($children as $child)
   {
      if ($child->node_type() == XML_ELEMENT_NODE)
      {
         if ($child->tagname() == "BookInfo") {
         $newbook = $doc-create_element("book");
         $newbook->set_attribute("bID", $ID);
         $nname = $doc-create_element("bookName");
         $nametext = $doc-create_text_node($bookName);
         $nname->add_child($nametext);
         $nprice = $doc-create_element("list_price");
         $pricetext = $doc-create_text_node($list_price);
         $nprice->add_child($pricetext);
         $nauthor = $doc-create_element("author");
         $authortext = $doc-create_text_node($author);
         $nauthor->add_child($authortext);
         $nquantity = $doc-create_element("quantity");
         $quantitytext = $doc-create_text_node($quantity);
         $nquantity->add_child($quantitytext);
```

```
            $newbook->add_child($nname);
            $newbook->add_child($nprice);
            $newbook->add_child($nauthor);
            $newbook->add_child($nquantity);
            $child->add_child($newbook);
         }
      }
}
$xml = $doc->dump_mem();
$fp = fopen("/var/www/html/add.xml", "w+");
fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
return $xml;
}
?>
```

The above listing lets you add a new book in the online shopping cart.

You need to pass book details, such as book titles, authors, prices, and quantities as arguments to the AddProduct() function. The cart.xml file is parsed using the xmldocfile() function, and the root and child nodes are retrieved from the document. The AddProduct() function verifies the tag name property of the element node and uses the add_child() function to add new child nodes and attributes to the DOM tree. In the above listing, the dumpmem() method dumps the modified XML document into a string. Figure 3-6 shows the output of Listing 3-13:

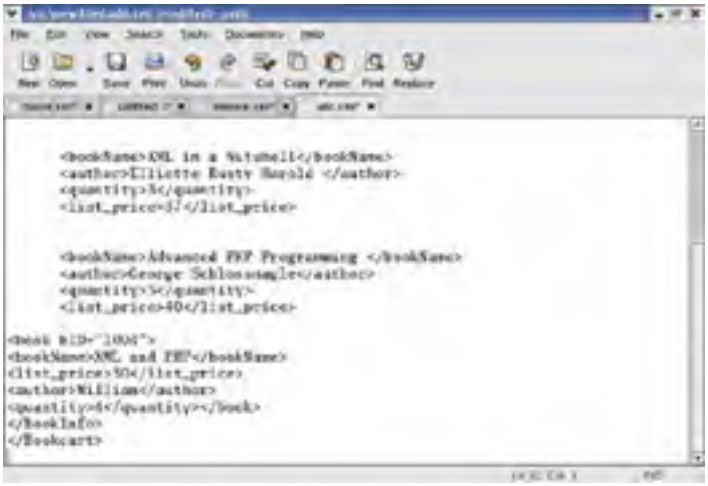


Figure 3-6: Output of Adding Information about Books Available in the DOM Tree

You can also delete elements from the tree structure using the DOM parser. For example, you can delete the information about a book sold from the online shopping cart.

Listing 3-14 shows how to delete elements from the DOM tree.

Listing 3-14: Removing Tree Elements

```
<?php
$xml_file = "bookcart.xml";
$tt=RemoveProduct($xml_file, "1003");
print($tt);
function RemoveProduct($xml, $id)
{
   $doc = xmldocfile($xml);
   $root = $doc->root();
   $children = $root->child_nodes();
   foreach ($children as $child)
   {
      if ($child->node_type() == XML_ELEMENT_NODE)
         {
            if ($child->tagname() == "BookInfo")
            {
               $BookInfo = $child->child_nodes();
               foreach ($BookInfo as $book)
               {
                  if ($book->node_type() == XML_ELEMENT_NODE)
                  {
                     $book_children = $book->child_nodes();
                     $ts = $book->get_attribute("ID");
                     if ($ts == $id)
                     {
                        $book->unlink_node();
                     }
                  }
               }
            }
         }
    }
```

```
    $xml = $doc->dumpmem();
    $fp = fopen("/var/www/html/remove.xml", "w+");
    fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
    return $xml;
}
?>
```

The above listing lets you remove the instances of a particular book from the online shopping cart. In the above listing, all the instances of the book with the book ID 1003 are removed from the cart using the unlink_node() function. Figure 3-7 shows the output of Listing 3-14:

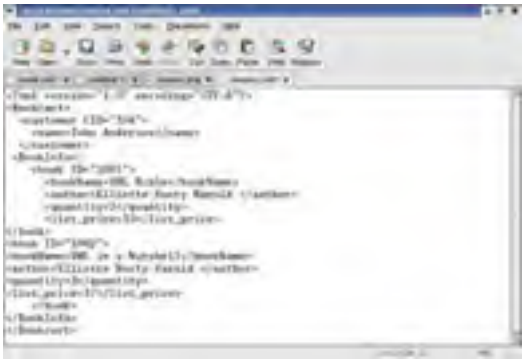


Figure 3-7: Output of Removing Tree Elements

Using DOM to Query a Document

In addition to adding and deleting elements from the XML document tree, the DOM parser also lets you retrieve specific information from an XML document. Querying the XML documents includes the process of traversing through the XML document to search for particular elements or information. For example, you have created an XML document that stores books information, such as author names, number of pages, prices, and titles of the books available at an online bookstore, and you want to retrieve information on a specific book from the XML document. You can create a query to retrieve information from the XML document using the DOM parser.

Listing 3-15 shows how to create an XML document that stores information on books:

Listing 3-15: Creating the book.xml File

```
<?xml version="1.0"?>
<books>
<book>
<title>XML in a Nutshell</title>
<author>Elliotte Rusty Harold</author>
<price>121</price>
<pages>300</pages>
</book>
<book>
<title>Advanced PHP Programming</title>
<author>George</author>
<price>161</price>
<pages>150</pages>
</book>
<book>
<title>Learning XML</title>
<author>John William</author>
<price>231</price>
<pages>200</pages>
</book>
</books>
```

The above listing creates the XML file, book.xml, which stores information about books available for online shopping. The above listing lets you retrieve information, such as author names and book prices from the book.xml file, for a book titled Learning XML, with more than 100 pages.

Listing 3-16 shows how to create the query to retrieve the specified book information:

Listing 3-16: Querying the XML Document using DOM

```
<?php
function get($domnode)
{
   $content = '';
   foreach ($domnode->child_nodes() as $child)
   {
      if ($child->node_type() == XML_TEXT_NODE)
      {
      $content .= $child->content;
```

```
        }
    }
    return $content;
    }
    $doc = xmldocfile("book.xml");
    $root = $doc->document_element();
    foreach ($root->child_nodes() as $element)
    {
       if ($element->node_type() == XML_ELEMENT_NODE and
       $element->tagname() == "book")
       {
          $c1 = false;
          $c2 = false;
          foreach ($element->child_nodes() as $book_element)
          {
             if ($book_element->node_type() == XML_ELEMENT_NODE and
             $book_element->tagname() == "title")
             {
             if (($title = get($book_element)) == "Learning XML")
             {
                $c1 = true;
             }
          }
          if ($book_element->node_type() == XML_ELEMENT_NODE and
          $book_element->tagname() == "pages")
          {
             if (($pages = get($book_element))>100)
             {
                $c2 = true;
             }
          }
          if ($book_element->node_type() == XML_ELEMENT_NODE and
          $book_element->tagname() == "author")
          {
             $author = get($book_element);
          }
          if ($book_element->node_type() == XML_ELEMENT_NODE and
          $book_element->tagname() == "price")
          {
             $price = get($book_element);
          }
       }
       if ($c1&& $c2)
       {
          print ("Author: $author Price:$price<br/>");
       }
    }
}
?>
```

The above listing shows how to query for specific information stored in an XML document using the DOM approach.

The DOM parser traverses through the entire DOM tree to search for the nodes that fulfill the specific conditions. The listing searches for the nodes that are of the element type and then verifies the tag name and the text in the nodes. The listing displays the author name and the price of the book with the title, Learning XML with more than 100 pages.

Figure 3-8 shows the output of Listing 3-16:

Figure 3-8: Output of Querying the XML Document

Using DOM to Write XML Data

In addition to modifying or traversing the XML document, DOM lets you create a new XML document. You can create a root node and add children to the root node to construct the DOM tree. The process of creating an XML document using the DOM parser can be used in the following instances:

- Creating an XML file from a text file.
- Constructing an XML document from various databases such as MySQL and MSXS.
- Combining more than one XML document into an XML document.
- Representing the PHP object properties in the form of an XML document.

The DOM approach provides you with various methods, such as add_root() and new_child(), which lets you create the root and text nodes of a DOM tree. For example, the add_root () method lets you add a root node to the DOM tree, and the set_attribute () method lets you set the attributes for the root node. The new_xmldoc() method lets you create a new XML document using the DOM parser.

Listing 3-17 shows how to create an XML document using the DOM parser:

Listing 3-17: Creating XML File using DOM

```
<?php
$doc = new_xmldoc("1.0");
// Adding root node
$root = $doc->add_root("Book");
// Setting attribute for the root node
$root->set_attribute("Title","XML and PHP");
// Adding children to the root node
$title = $root->new_child("Author","John Williams");
$author = $root->new_child("Price","145");
$date = $root->new_child("PublishingDate", date("d-M-Y", mktime()));
echo $doc->dumpmem();
print "$new_xmldoc";
$fp = fopen("/var/www/html/add.xml", "w+");
fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
return $xml;
?>
```

The above listing creates a DomDocument object, new_xmldoc, generating the first version of an XML document. The root node, Book, and the child nodes, such as Author, Price, and PublishingDate, are added to the DomDocument object. The dumpmem () method dumps the created XML tree into a string.

Figure 3-9 shows the output of Listing 3-17:

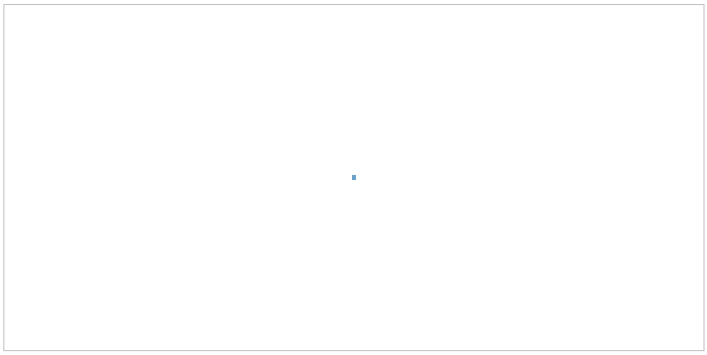


Figure 3-9: Contents of the File Created using DOM

Chapter 4: Understanding Extensible Stylesheet Language for Transformation

 Download CD Content

Extensible Stylesheet Language for Transformation (XSLT) is a language using which you can transform an eXtended Markup Language (XML) document into any other format, such as HTML. XSLT stylesheets change the structure and type of an XML document

This chapter introduces XSLT, and explains its data model, expression, and templates. It also explains the variables and parameters used in XSLT.

Introducing XSLT

XSLT is an application of XML that specifies how data is sent over the Internet using XML. You can use the XML Path (XPath) language to retrieve elements and attributes of an XML document to perform transformation on XML data. During the transformation process, you can sort, add, and delete elements from an XML document using XSLT. You can also change the order of the elements in the output of an XML document using the expressions and templates in XSLT. The XML document that is transformed is called a source tree or the source document and the transformed document is called the result tree or result document.

The XSLT processor parses the source tree, by creating an XSLT tree from the stylesheet that contains the specification of the source tree. The XSLT processor uses the source tree to generate the result tree. Various elements in XSLT, such as stylesheet, value-of, for-each, sort, and text help to create data model of XSLT, which specifies the structure of the source tree.

Data Model

The data model of XSLT specifies the structure of the source tree, which consists of elements and attributes of an XML document. You can also refer to elements in an XML document as nodes. XSLT validates an XML document by creating a tree structure of elements and attributes present in the document. A tree of an XML document consists of various nodes, such as root, element, text, attribute, namespace, processing instruction, and comment.

The root node refers to the root element of the tree structure. An element of an XML document refers to the child node of the root element. The root node contains one or more child nodes but does not contain any text. The root node also contains processing instructions and comment nodes as its child nodes.

The data model of an XSLT consists of various elements, such as:

- `<xsl:stylesheet>`
- `<xsl:value-of>`
- `<xsl:for-each>`
- `<xsl:sort>`
- `<xs:text>`

Each XSLT stylesheet starts with the stylesheet element. The syntax to declare the stylesheet element is:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

In the above syntax, xmlns is the attribute of the stylesheet element that provides a Uniform Resource Identifier (URI) of XSLT. You need to specify the URI, http://www.w3.org/1999/XSL/Transform, as a value of the xmlns attribute, to check that the elements and attributes are specified according to the specifications defined by W3C.

The value-of element displays the value of an element or attribute. The syntax to use the value-of element is:

```
<xsl:value-of select="element_name/@attribute_name"/>
```

In the above code:

- The select attribute of the value-of element indicates the name of an element or attribute that displays its value on a Web browser.
- The attribute name of an element must be prefixed with the @ sign.
- The value-of element is an empty tag that includes the slash mark, /, before the right angle bracket of the tag.

Enter the following code in the stylesheet to use the value-of element:

```
<xsl:value-of select="STUDENT"/>
```

The above code displays the value of the STUDENT element of an XML document. Enter the following code to display the value of an attribute:

```
<xsl:value-of select="@id"/>
```

The above code displays the value of the id attribute.

The for-each element of XSLT processes data each time a specified pattern occurs in the XML document. This element lets you use looping construct in XSLT as in the C or C++ language. The syntax to use the for-each element is:

```
<xsl:for-each select="pattern">
<!-- Data to be processed -->
</xsl:for-each>
```

In the above syntax, the select element can contain three patterns:

- The element pattern: The XSLT processor processes the data for each instance of a given element.
- The root/sub-element pattern: The XSLT processor processes the data each time a specified sub-element is encountered under the root element.
- The ancestor//sub-element pattern: The XSLT processor processes the data each time a specified sub-element is encountered under the ancestor element.

Enter the code in Listing 4-1 to use the for-each element with the root/sub_element pattern:

Listing 4-1: Using the for-each Element

```
<xsl:for-each select="STUDENTDATA/STUDENT">
<font color="red" size=20><b><xsl:value-of select="NAME" /></b></font>
<xsl:value-of select="@id"/>
</xsl:for-each>
```

In the above code:

- The root/sub_element pattern is used, which processes the data when the element encounters the STUDENT sub element of the STUDENTDATA root element.
- The <font> element of the HTML document specifies the font size and color for the value of the NAME element.
- The <b> element of the HTML document displays the value of the NAME element in bold. The value of the id attribute is displayed in normal text.

The sort element of XSLT arranges data in the ascending or descending order. The sort element has four attributes, select, order, case-order, and data-type. The syntax to use the sort-element is:

```
<xsl:sort select="element_name/expression"
order="ascending/descending"
case-order="upper-first/lower-first"
data-type="text/number"/>
```

In the above syntax:

- The select attribute: Indicates the expression on which sorting is performed.
- `The order attribute`: Indicates the order of sorting.
- `The case-order attribute`: Indicates the order of sorting for the uppercase and lowercase characters.
- `The data-type attribute`: Specifies the data type of the expression that is to be sorted.

Listing 4-2 shows the code to use the sort element:

Listing 4-2: Using the sort Element

```
<xsl:for-each select="STUDENT">
<xsl:sort select="AGE" data-type="number" order="ascending"/>
<xsl:value-of select="NAME"/>
<xsl:value-of select="AGE"/>
</xsl:for-each>
```

The above code shows that the for-each element traverses every sub element of the STUDENT element. The code displays the value of the NAME and AGE elements in the ascending order of the age element.

The default value for the order attribute is ascending, and the default value for the data-type is text.

Templates

XSLT contains a collection of template rules that provide specifications to convert an XML document and display the results on a Web browser. A template rule has two parts, which contain information about the transformation process. The first section specifies the part of an XML document to be converted, and the second specifies the format of the output.

The XSLT processor accepts the XML document and templates, processes the data, and generates an output.

The template and apply-templates elements define the template rules in XSLT. The syntax to use the template element is shown in Listing 4-3:

Listing 4-3: Using the syntax Element

```
<xsl:template match="expression">
<!--Data to be processed-->
</xsl:template>
```

In the above syntax, the value of the match attribute indicates an element of an XML document that checks against the template rules.

Various expressions that you can use in the match attribute are:

- `/`: Indicates the root element that matches all the elements, such as processing instruction, comments, and the root element of the XML document. It is also known as document root.
- `element`: Indicates an element of an XML document, which is processed when the XSLT processor encounters it.
- `*`: Indicates a wildcard character that matches any element in an XML document.
- `element1| element2`: Indicates two elements, which are processed when the element1 and element2 are encountered in an XML document.
- element [@attribute]: Processes data when an attribute of an element is encountered.
- `element [@attribute='value']`: Matches all the elements that contain the specified value of the attribute defined in the @attribute attribute.
- `root_element/sub_element`: Processes data when the sub elements of the root element are encountered.
- `ancestor//sub_element`: Processes data when the sub element under the ancestor element is encountered.

The apply-template element enables the XSLT processor to apply the templates, specified by the template element, to the selected XML elements. The syntax to use the apply-template is:

```
<xsl:apply-template [select="expression"]>
```

The above code shows that the template rules apply on the expression specified in the select attribute of the apply-template element. In the syntax, the select attribute is optional.

The default value of the apply-template element is node(), which specifies that the template matches the sub elements of the current node. The expression specified in the select attribute of the apply-template element matches the expression specified in the match attribute of the template element. Each node contains a base URI that resolves the value of the attributes. The URI of the document entity refers to the base URI of the document root node.

Listing 4-4 shows how to use the template and apply-template elements:

Listing 4-4: Implementing Template Rule

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:template match="/">
<xsl:apply-templates/>
</xsl:template>
```

In the above listing:

- The <xsl:template match="/"> element indicates the template rule.
- The value of the match attribute specifies the document root of an XML document.
- The apply-template element indicates that the template affects all the elements of an XML document.
- The listing does not contain a format style, so the output displays the result with the default font size, face, and color.

You can define your own template rules using the template and apply-templates elements, as shown in Listing 4-5:

Listing 4-5: Creating Template Rules

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  version="1.0">
<xsl:template match="/">
<xsl:apply-templates select="STUDENTDATA/STUDENT"/>
</xsl:template>
<xsl:templalte match="STUDENTDATA/STUDENT">
<font color="red">
<LI><xsl:apply-templates/></LI>
</font>
</xsl:template>
</xsl:stylesheet>
```

The above listing shows that the template, STUDENTDATA/STUDENT, is created for the document root. The XSLT processor searches for the STUDENTDATA/STUDENT template. It then sets the font color of the value of all the child elements of the STUDENT element to red, and displays them in bulleted list.

Expression

In addition to the templates, XSLT also contains expressions, which correspond to the nodes of an XML document being transformed. The main functions of the XSLT expressions are:

- Identify the node to be transformed.
- Specify the condition to process the node.
- Insert character data in the result tree.

The XPath expression defines the format of the expression for XSLT. It retrieves data based on specified conditions, and performs calculations on numeric data. The XPath expression can contain various characters, which are:

- `\n`: Indicates a newline character.
- `.`: Indicates a single character.
- `\d`: Indicates a numeric character.
- `\s`: Indicates the whitespace characters, such as horizontal tab, line feed, and carriage return.
- `[a-d]`: Indicates the characters from a to d.
- `*`: Indicates zero or more occurrences of the whitespace characters.
- `+`: Indicates one or more occurrences of the whitespace characters.
- `?`: Indicates zero or one occurrence of the whitespace characters.
- `\?`: Searches for the question mark character, and overwrites the special meaning of the question mark expression.

You can combine the characters in the XPath expression to denote specific meaning. For example, the expression character, \s*, used in the XPath expression, specifies zero or more occurrences of the whitespace characters. The expression character, \s+, specifies at least one or more occurrences of the whitespace characters.

An expression in XSLT uses the following functions:

- `tokenize(string, delimiter)`: Returns the number of strings that are separated by the delimiter.
- `matches(string1, string2)`: Returns the value, true, which specifies that the value of the first parameter is same as the value of the second parameter.
- `replace(string1, string2, string3)`: Searches the second string within the first string and replaces the searched string with the third string specified in the function.

You can specify expressions within a function to change the format of an XML document.

Listing 4-6 shows the use of the expression characters and functions:

Listing 4-6: Using the XSLT Expressions and Functions

```
<xsl:template match="price">
<xsl:match>
<xsl:attribute name="pattern">
<xsl:value-of select='matches(.,".*\$\d\.\d.*")' />
</xsl:attribute>
<xsl:value-of select='replace(., "\$\d\.\d", "\$#.#")' />
</xsl:match>
</xsl:template>
```

The above listing shows that the template changes the price element by adding the pattern attribute. The pattern attribute stores the value, true or false, depending on the result of the match() function. The content of the price element changes according to the result of the replace() function. In the above code:

- The first parameter of the matches() function indicates the entire text of the document.
- The second parameter of the matches() function indicates the expression that specifies any text before and after the dollar sign, followed by a digit, period, and another digit.
- The second parameter of the replace() function indicates the expression that specifies the dollar sign followed by a digit, period, and another digit.
- The XSLT processor replaces the text of the pattern element with the third parameter, $#.#, of the replace() function.

Listing 4-7 implements the expressions and functions of XSLT on the following XML document:

Listing 4-7: Implementing the Expressions and Functions on XML Document

```
<product>
<price>Price of Pen: $2.8</price>
<price>Price of Pencil: $1</price>
</product>
```

In the above code:

- The first instance of the price element matches the expression. As a result, the pattern attribute of the price element stores the value, true.
- The second instance of the price element does not match the expression, because it does not contain a period followed by a digit.

The template rule specified in Listing 4-3 changes the XML document, as shown in the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<product>
<price pattern="true"> Price of Pen: $#.#</price>
<price pattern="false">Price of Pencil: $1</price>
```

</product>

The above code shows that the pattern attribute of the second instance of the price element contains the false value because it does not match the expression. As a result, the text, $1, of the price element is not replaced with the text, $#.#.

◄ PREVIOUS NEXT ►

Using Variables and Parameters in XSLT

A variable is a memory location that stores data, and parameter is a variable that is passed to a function. The <xsl:variable> element helps define variables in XSLT, and the <xsl:param> element helps define XSLT parameters. The <xsl:variable> and <xsl:param> elements contain the name attribute that specifies the name of the variable and parameter.

You create local variables and parameters by defining them in the template rule. You can also create global variables and parameters in XSLT, by creating them as child elements of the <xsl:stylesheet> element. The global variable is accessible throughout the stylesheet.

You cannot change the value of an XSLT variable. You need to create a new variable if you want to change the value of an existing variable. Enter the following code to create an XSLT variable:

```
<xsl:variable name="StudName" select="/StudentDetail/Student>
</xsl:variable>
```

In the above code, the select attribute specifies the XPATH expression and the result of the expression specifies the value of the XSLT variable. The StudName variable indicates an element node, which refers to the Student element of the source document.

Result Tree Fragments

The <xsl:variable> or <xsl:param> element creates a result tree fragment when you refer to the value of a specific element in the result tree. The result tree fragment is a part of the result tree obtained from an XSLT transformation, and is created by the XSLT processor. A result tree fragment should be balanced, which means that each start tag must correspond to an end tag. You can perform a valid string operation on a result tree fragment. For example, the following code shows a part of an XSLT stylesheet:

```
<xsl:variable item="Computer">
<ch>chapter number</ch> books
</xsl:variable>
```

The item variable, in the above code, creates the following result tree fragment:

```
<ch>chapter number</ch> books
```

The result tree fragment, in the above code, consists of the <ch> element, and the trailing string, books, outside the <ch> element.

> **Note** You cannot use the /, //, and [] operators on result tree fragments.

The <xsl:copy-of> element inserts a result tree fragment into the result tree without converting it to a string value. The syntax to use the <xsl:copy-of> element is:

```
<xsl:copy-of select=expression/>
```

The above syntax shows that the select attribute contains an expression that is evaluated. When the evaluation of the specified expression returns a result tree fragment, the <xsl:copy-of> element copies the tree fragment to the result tree.

An XSLT stylesheet consists of one or more template rules, which identify a pattern that you need to match in a source tree, and describe the structure of the result tree. Each template rule is applicable to a specific set of nodes, and executes a set of instructions to create the structure of the nodes in the result tree.

Top-level Variables and Parameters

A top-level variable is a sub-element of the <xsl:stylesheet> element. A top-level variable creates a global variable and parameter. You cannot change the value of a top-level variable. The <xsl:variable> and <xsl:param> elements are the examples of top-level elements. The <xsl:param> element only creates a parameter of the stylesheet, but does not pass the parameter to the stylesheet. You can assign a value to the parameter of the stylesheet using various forms, such as the top-level parameter, the <xsl:with-param> element, or a default value, as shown in Listing 4-8:

Listing 4-8: Assigning the Value to a Parameter

```
<xsl:template match="/">
<xsl:call-template name="Temp'>
<xsl:with-param name="paramName" select="'any-value'"/>
</xsl:call-template>
</xsl:template>
<xsl:template name="Temp">
<xsl:param name="paramName" select='default-value'"/>
<xsl:value-of select="$param"/>
</xsl:template>
```

The above listing shows that the <xsl:call-template> element invokes the Temp template, which accepts the paramName parameter. The Temp template assigns a default value to the parameter.

Variables and Parameters in a Template

XSLT contains the <xsl:variable> element that creates variables in stylesheets. The syntax to declare an XSLT variable is:

```
<xsl:variable name="var" select="0"/>
```

In the above syntax, you declare a variable using an XPath expression. The var variable, declared in the above syntax, stores the numeric value, 0.

Another syntax for declaring an XSLT variable is:

```
<xsl:variable name="var">0</xsl:variable>
```

In the above syntax, the value of the var variable is specified within the starting and ending tags.

You can use a variable either as a top-level element or as a template-level element. A template-level element is defined within the template, and all the elements positioned after the variable declaration can use this element. You can change the value of a variable by declaring it inside the for-each loop, where its value changes on every iteration. The template-level elements remain in effect only within their scope, which is represented by the following-sibling nodes and the descendent nodes of the elements.

The parameter element in XSLT is similar to the variable element. Both these elements share the same namespace and syntax to assign names and values to an element. The keyword, param, denotes the parameter element. The param element consists of a name attribute and a select attribute. The value of the select attribute in a parameter element specifies the default value of the element. This means that if you pass a new value to the select attribute, then the default value is replaced with the new value. You can assign value to a parameter element either from the process that invokes the stylesheet, or from the <xsl:with-param> element.

To pass the value of a parameter to another template, you pass a parameter using the <xsl:with-param> element. The process of passing a parameter to a template calls a specific template using the <xsl:call-template> element.

◄ PREVIOUS NEXT ►

Chapter 5: PHP and XPath

Download CD Content

Extensible Markup Language (XML) Path (XPath) language is a query language that you can use to retrieve selected information from an XML document. It also provides functions to manipulate strings, numbers, booleans, and nodeset values in an XML document. Using XPath, you can create XPath expressions that specify the path of an XML element to be retrieved from an XML document.

Hypertext Preprocessor (PHP) consists of XPath classes that you can use to retrieve XML elements matching specific criteria. You can use XPath expressions with the Document Object Model (DOM) and Simple Application Programming Interface (API) for XML (SAX) parsers implemented in PHP language to determine the location of the XML elements in an XML document. In addition, you can process XML documents using XPath expressions in Extensible Stylesheet Language for Transformation (XSLT) templates.

This chapter describes XPath and how to use XPath expressions in an XML document. It also describes how to use location paths to retrieve specific elements from an XML document. In addition, it explains the method to access XML elements using DOM and SAX parsers and XSLT templates implemented in PHP.

Introducing XPath

An XPath expression specifies the part of an XML document that you need to select and retrieve.

XPath selects a portion of data from an XML document by navigating the hierarchical structure of that document. This query language follows a path to point to a specific node in the XML document hierarchy. XPath searches an XML document from the starting point of the document called the context node, progresses in a specific direction, and arrives at the required point by navigating through the path described by a location path. Location steps in the location path determine the path to navigate an XML document and locate a particular XML element.

Location steps consist of three parts: an axis, node test, and predicates. The axis informs the XPath processor about the process of navigating around an XML document. The node test determines the type of node selected by the location step, along with its name. The predicate filters the set of nodes selected by the axis and the node test. If there are more than one location step in a location path, each location step uses the node in the previous nodeset as its starting point.

Creating Nodes Using XPath

An XML document consists of a succession of elements, each with a start and end tag. Various elements are nested within each other in the document. XPath uses a data model to represent all elements within an XML document as a hierarchy of nodes.

For example, you want to represent the data model in XPath for the customer.xml XML document.

Listing 5-1 shows the contents of customer.xml:

Listing 5-1: The customer.xml Document

```
<?xml version="1.0"?>
<customer>
<companyname>IBG, Inc.</companyname>
<item>Computer</item>
<itemcode>c001</itemcode>
<item_quantity>20</item_quantity>
<price>20000</price>
</customer>
```

The above listing shows the content of an XML document, customer.xml. The document contains information about an item that a company wants to purchase. This information includes the company name, item name, item code, item quantity, and the price.

Figure 5-1 shows the corresponding data model in XPath for the customer.xml document:

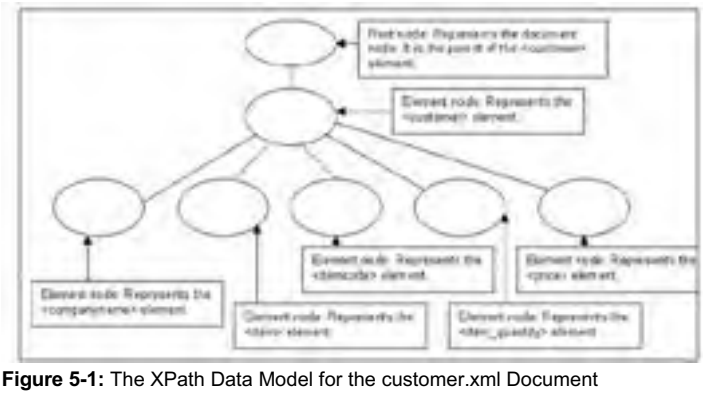


Figure 5-1: The XPath Data Model for the customer.xml Document

This model shows a root node, which is the parent of the element root. The element root is represented by the element node, which represents the <customer> element. The element node that represents the <customer> element contains five child element nodes that represent the <companyname>, <item>, <itemcode>, <item_quantity>, and <price> elements.

In XPath, a node refers to a specific part, such as an element or attribute, of an XML document. Each XML document functions as a tree and consists of a root node, which is the parent of the element node. XPath includes seven types of nodes. Of these seven nodes, only some nodes consist of an expanded name. The expanded name consists of two parts, local part and namespace Uniform Resource Identifier (URI). Local part is a string and namespace URI is either a string or a NULL value.

The namespace URI is a URI reference. An XML document that does not contain a namespace declaration contains a NULL value for the namespace URI. In these cases, the expanded name of an element is same as the element name.

The seven types of nodes available in XPath are:

- Root: Represents the root of the XPath tree. There is only one root node for each XPath tree structure. It does not have an expanded name.
- Element: Represents each element in the XML source document. It contains an expanded name that results from expanding the Qualified Name (QName) in the element tag. The QName is an element name with a prefix added to it. If the QName does not contain a prefix and default namespace, the value of the namespace URI for the expanded name of the element is NULL.
- Attribute: Represents the attributes of an element in the XML source document. It can be empty. The element node is the parent of the attribute node. However, an attribute node is not a child of the parent node.
- Namespace: Associated with an element node. A namespace node exists for each attribute on the element whose name starts with the prefix, xmlns:. A namespace node exists for each attribute on an ancestor element with names starting from xmlns:.
- Processing instruction: Represents the processing instructions in the XML source document. It consists of an expanded name with the local part representing the application to which the processing instruction applies.
- Comment: Represents the comments in the XML source document. It does not have an expanded name.
- Text: Represents the character data in the XML document. Text nodes are not produced for character values that exist inside comments, attributes, and processing instructions. It does not have an expanded name.

Note If a location path starts with child, the resultant nodeset contains neither attribute nor namespace nodes.

Every node other than the root node in the data model consists of one parent. For example, the XML document, customer.xml, consists of all nodes included in it.

Listing 5-2 shows the customer.xml document that contains different types of nodes:

Listing 5-2: The customer.xml Document with Different Nodes

```
<?xml version="1.0"?>
<customer><!—Denotes the element node-->
<companyname company="IBG, Inc."><!—company is the child attribute node.-->
The name of the company is IBG, Inc.!<!—The string denotes the child text node.-->
<customer xmlns=http://www.xmml.com/><!—Denotes the namespace
node-->
</companyname>
<item>Computer</item>
<?customer SELECT * FROM item?><!—Denotes the processing instruction node.-->
</customer>
```

The above listing shows the content of the customer.xml document with a root node, which is the parent of the comment node, <companyname> element node, <item> element node, and the processing instruction node. The root node is not visible in the document. The <companyname> element node consists of a child attribute node, which is the company with the value, "IBG, Inc.". The <companyname> element node also contains a child text node with the value, "The name of the company is IBG, Inc.". The comment node starts with <!—and ends with -->.

Note Two expanded names are equal if they have the same local part, regardless of the value of the namespace URI.

Implementing the XPath Syntax

XPath does not use the XML-based syntax. You cannot include XPath expressions in the URIs or attributes of XML elements. The XPath syntax is the same as file system addressing and enables querying the data in an XML document. XPath uses a pattern expression to:

- Identify the nodes within an XML document.
- Select unknown elements.
- Select the branches of an element.
- Select several paths along the axis.
- Select attributes.

An XPath pattern expression describes a path that contains a list of child element names separated by a slash. The pattern selects those elements from an XML document that matches the path.

Listing 5-3 shows a sample XML document:

Listing 5-3: The sample XML Document

```
<?xml version="1.0"?>
<customer>
<item itemcode="01">
<itemprice price="15.00"></itemprice>
<quantity value="20"></quantity>
</item>
<item itemcode="02">
<itemprice price="30.00"></itemprice>
<quantity value="10"></quantity>
</item>
</customer>
```

The above XML document contains information about items purchased by a customer. The document contains item information, such as the item code, item price, and the quantity.

The following pattern expression selects all <itemprice> element nodes of all <item> element nodes of the <customer> element node:

```
/customer/item/itemprice
```

In the above pattern expression, <itemprice> element is the child element node of the <item> element node, which is the child element of the <customer> element.

> **Note** If a path expression starts with a double slash (//), the resultant nodeset contains all elements described by the path expression.

For example, the path expression //customer selects all customer elements in the customer.xml document.

XPath uses wild card characters, such as *, to select unknown elements in an XML document. The * wild card character selects all elements defined by the preceding path. For example, the expression to select all child element nodes of all <item> element nodes of the <customer> element node is:

```
 /customer/item/*
```

Similarly, the expression to select all <itemprice> elements that are grandchild elements of the <customer> element is:

```
/customer/*/itemprice
```

The pattern expression to select all elements that have two ancestors is:

```
/*/*/itemprice
```

The pattern expression to select all elements in an XML document is:

```
//*
```

You can specify branches on a location step using additional patterns. As a result, these additional patterns further describe an element. For example, the expression that selects the first node representing the <item> element child of the <customer> element node is:

```
/customer/item[1]
```

In the above syntax, the numeric value in the square brackets represents the position of the element in the selected element set. The expression to select the last <item> element belonging to the <customer> element is:

/customer/item[last()]

The expression that selects all <itemprice> element nodes of the <customer> element node with the <price> attribute value 15.00 is:

```
/customer/itemprice[price=15.00]
```

You can select several paths in a document using the pipe (|) operator in XPath. For example, the expression to select all <quantity> and <itemprice> elements of the <item> element of the <customer> element is:

```
/customer/item/quantity|/customer/item/itemprice
```

Similarly, the expression to select all <quantity> and <itemprice> elements in the XML document is:

```
//quantity|//itemprice
```

Using XPath, you can also select attributes in an XML document. The at (@) symbol prefixes all attributes in an XPath expression. The @ symbol signifies that selected elements represent the attributes of an element. For example, the expression that selects all attributes named as price is:

```
//@price
```

The expression that selects all <item> elements with an attribute named itemcode is:

//item[@itemcode]

XPath syntax can be either abbreviated or unabbreviated. The unabbreviated syntax for a location step consists of an axis name and node test separated by a double colon. The syntax may or may not include predicates in square brackets. The child axis includes the child nodes of the context node. This axis is the default axis, and you can eliminate it from the location step. For example, you can write the abbreviated form of an XPath expression as:

```
/customer
```

The above syntax is the same as the following unabbreviated XPath expression:

```
 /child::customer
```

The descendant axis consists of nodes that are descendants of the context node. The descendants can be a child node or a grandchild node. For example, an XPath expression that selects descendant nodes is:

```
/descendant::*
```

In the above syntax, XPath selects all descendant nodes of the root element node.

Note The descendent axis cannot include the attribute and namespace nodes.

PREVIOUS NEXT

XPath Expressions

XPath expressions select a part of data from an XML document using a location path, which notifies XPath about the path that needs to be followed in an XML document. An XPath processor returns one of the following objects after evaluating an XPath expression:

- Nodeset: Represents an unordered collection of nodes.
- Boolean: Represents either TRUE or FALSE.
- Number: Represents a floating-point number.
- String: Represents a sequence of Unicode characters.

Structure of an XPath Expression

A DOM or SAX parser parses an XPath expression by separating the character string into tokens, such as identifiers and numbers. The DOM or SAX parser then parses the resulting tokens. This process is called the tokenization. You can form tokens from a sequence of characters. XPath expressions are formed from these tokens. You can identify various tokens in an XML document using lexical structures, which defines the syntactic structure of an XPath expression. For example, the lexical structure for the node type token in an XPath expression is:

```
Node Type::='comment'|'text'|'processinginstruction'
```

The above structure shows that the type of node specified in an XPath expression can be a comment node, text node, or a processing instruction node.

XPath Functions for Nodeset

A nodeset is a collection of nodes. A nodeset is formed when the | operator combines two or more location steps. The following XPath functions operate on nodesets:

- `count()`: Accepts a nodeset as its argument and returns the total number of nodes present in the nodeset.
- `id()`: Returns a nodeset that contains nodes with the given ID attribute.
- `local-name()`: Returns the local part of the name of a node.
- `last()`: Returns the value of the context size. In the XPath data model, you can determine the context size by counting the number of nodes that represent the same element in an XML document.
- `name()`: Returns a QName representing the name of a node.
- `position()`: Returns the value of the context position, which is the position of the context node in the nodeset selected prior to the predicate that the XPath processor evaluates.
- `namespace-uri()`: Returns a string value that represents the namespace URI in the expanded name of the node.

Types of XPath Expressions

In addition to location path expressions, XPath supports string, numerical, equality, relational, and boolean expressions. You can use these expressions if you do not want to retrieve a nodeset from an XPath data model. You can use these expressions in XPath predicates.

You can use numerical expressions to perform arithmetic operations on numbers in an XML document. XPath converts each operand to a number before performing arithmetic evaluation.

Table 5-1 describes various numerical operators that you can use with numerical expressions:

Table 5-1: XPath Numerical Operators

Numerical operator	Description	Example	Output
+	Addition	2+1	3
-	Subtraction	2-1	1
*	Multiplication	2*1	2
Div	Division	6 div 2	3
Mod	Modulus	6 mod 4	2

The following numerical XPath expression returns all descendant nodes whose price value is divisible by 2:

```
//[price mod 2 ==0]
```

You can use boolean expressions to compare two element nodes in an XML document. XPath converts each operand in the boolean expression to a boolean value before evaluating it. The boolean operators that you can use with an XPath expression are:

- Or: Performs the OR operation.

- And: Performs the AND operation.

For example, the following XPath expression selects all child <books> element nodes that have the title and price child elements:

```
/books[title and price]
```

The following XPath expression selects all child <books> elements that have either the <bookcode> child element node with a value greater than 20 or the <price> child element node with a value less than 500:

```
/books[(bookcode>20) or (price<500)]
```

A string is defined as a sequence of one or more characters. You can also use string expressions in XPath predicates. For example, the following XPath expression selects all <title> child elements that have the name attribute value, Operating System, with the <books> element node as its parent:

```
/books/title[@name='Operating System']
```

FunctionCall XPath Expression

An XPath processor evaluates a FunctionCall XPath expression using a function necessary for evaluating an expression. FunctionName identifies the name of the function that evaluates the expression. All XPath functions are available in the XPath function library. The XPath processor calls a function to evaluate an XPath expression. The processor then evaluates the function arguments by converting each argument to the data type that the function accepts. The XPath processor passes these converted arguments to the called function. The value that the called function returns represents the result of the FunctionCall expression. The called function displays an error if the number of arguments passed to the called function is incorrect or if arguments are not converted to the type required by the called function.

The syntax for the XPath FunctionCall expression is:

```
FunctionCall=FunctionName (arg1, arg2, ..., argn)
```

In the above syntax, the function, FunctionName, operates on n number of arguments.

If the XPath processor calls a string function in an expression, arguments are converted to the string type. Similarly, if the processor calls a number function, arguments are converted to the number type.

Note You cannot convert an argument, which is not a nodeset type, to a nodeset.

XPath includes various functions to perform string operations, such as changing string characters from uppercase to lowercase. Various string functions are:

- `string()`: Converts its arguments to a string value. For example, it converts boolean arguments to string values, TRUE or FALSE.
- `string-length()`: Returns the total number of characters in a string.
- `starts-with()`: Determines whether or not a string starts with another string.
- `substring()`: Returns a portion of a string determined by the numeric arguments that describe the substring. For example, the function, substring('Tokyo', 1, 3) returns the substring, Tok. The numeric value, 1, in this function indicates the starting location and 3 indicates the number of characters in the substring.
- `concat()`: Combines two or more string arguments in the specified sequence. For example, the function, concat('The', 'world') returns the value, The world.
- `contains()`: Checks whether or not one string value contains another string value as its substring. For example, the function, contains('priority', 'prior') returns the value TRUE because the string, priority, contains prior as its substring.
- `normalize-space()`: Removes the leading and trailing whitespace from a string. It also converts the irrelevant whitespaces within a string by converting the sequence of whitespaces to a single space string. For example, the function, normalize-space('Global Systems') returns the string 'Global Systems'.
- `substring-after()`: Returns that portion of a string, which appears after the first occurrence of the specified substring. For example, the function substring-after('unauthorized', 'un') returns the string, authorized, because this string appears after the specified substring, un.
- `substring-before()`: Returns that portion of a string, which appears before the first occurrence of the specified substring. For example, the substring-before('predict', 'dict') function returns the string, pre.
- `translate()`: Accepts a string and two sequences of characters as its arguments. It then replaces the characters of the string by replacing characters in the first character sequence with equivalent characters in the second character sequence. For example, the function translate(SQL, LMNRSPQR, lmnrspqr) provides the output, sql.

XPath also consists of functions that operate on numbers in an XML document. A number represents a floating-point number. It can have any double precision 64-bit format Institute of Electrical and Electronics Engineers (IEEE) 754 value. The number functions available in XPath are:

- `number()`: Converts the arguments to a number. For example, this function converts the boolean argument TRUE to the value 1 and FALSE to 0.
- `round()`: Returns the integer value that is the closest to the numeric value of the argument.
- `floor()`: Returns the largest integer value that is either less than or equal to the argument of the function. This function rounds off a noninteger value to the immediate lower integer value.
- `sum()`: Adds a series of numbers present as a nodeset. This function converts each node in the nodeset to its

corresponding string value and then, converts these string values into corresponding numeric values. This function adds all numeric values and generates the result.

- `ceiling()`: Returns the smallest integer value that is greater than or equal to the function argument. It rounds off a noninteger value to the immediate higher integer value.

Note The round(), floor(), and ceiling() number functions accept only a single argument.

A boolean object type contain either of the two values, TRUE or FALSE. Various boolean functions in XPath are:

- `boolean()`: Converts its argument to a boolean value. For example, if the argument is a number greater than 0, the boolean() function evaluates it to TRUE. Otherwise, it evaluates the number to FALSE. Similarly, if the argument is a string and the string length is 0, the boolean() function returns the boolean value FALSE. Otherwise, it returns the boolean value TRUE.
- `false()`: Returns the boolean value FALSE. While working with boolean values, you can use the false() function in an XPath expression.
- `true()`: Returns the boolean value TRUE. While working with boolean values, you can use the true() function in an XPath expression.
- `lang()`: Returns a boolean value, which checks if the language of the context node is similar to the language supplied as an argument.
- `not()`: Returns the negation of its argument. For example, if the argument is TRUE, the not() function returns the value, FALSE.

Note The language of the context node is defined by inserting the xml:lang attribute in an XML document.

PREVIOUS NEXT

Location Paths

A location path is an XPath expression that notifies the XPath processor about how to navigate around an XML document. It is a sequence of location steps separated by a slash, /. The XPath processor evaluates a location path from left to right starting with an initial context node. Each node that results from the evaluation of one location step represents the context node to evaluate the next location step. XPath, then, combines the results of all location steps and returns selected XML elements.

There are two types of location paths, absolute and relative. You start the absolute location path with a /, but cannot start the relative location path with a /. In an absolute location path, the current nodeset consists of the root node. In a relative location path, the location steps are separated by a /. The / represents a direct parent-child relationship between the nodes involved in the location step.

Each location step selects a nodeset with respect to its context node. The nodes in a nodeset represent the context node for the next location step. For example, the location path, child::customer/child::item, selects the child item element of the child customer element of the context node. The child axis is the default value for an axis.

Creating Location Steps

A location step determines the nodes through which an XML document should be traversed to arrive at a final location. The syntax for a location step is:

```
axis_name::nodetest[predicate]
```

For example, you navigate through an XML source document, companydetails.xml, using XPath.

Listing 5-4 shows the content of the XML document, companydetails.xml:

Listing 5-4: The companydetails.xml Document

```
<company name="Blue Moon Systems">
<Managing Director>Ron Floyd</Managing Director>
<Department name="Administration">
<Name>Tom</Name>
<Age>35</Age>
<Address> 17, landmark plaza, New York</Address>
<EmployeeId>a01</EmployeeId>
</Department>
<Department name="Sales">
<Name>John</Name>
<Age>43</Age>
<Address> 34, landmark plaza, New York</Address>
<EmployeeId>s02</EmployeeId>
</Department>
</company>
```

The above listing shows the content of the companydetails.xml document that contains information about the Blue Moon Systems company. The companydetails.xml document consists of three element nodes: company name, Managing Director, and Department. The Department element node consists of an attribute node, name. For each Department attribute node, there are four child nodes: Name, Age, Address, and EmployeeId.

For example, the location path, child::company name/ child::Department [attribute::name="Sales"] consists of two location steps:

- child::company name
  - Axis: child
  - Node test: company name
  - Predicate: null
- child::Department [attribute::name="Sales"]
  - Axis: child
  - Node test: Department
  - Predicate: [attribute::name="Sales"]

The first location step does not contain a predicate but the second location step includes a predicate. The predicate in the second location step selects the node representing a Department element only if it contains a name attribute with the value, Sales.

Identifying Axes

An axis within an XPath location step determines the direction in which the XPath processor should navigate an XML document. There are two types of XPath axes, forward and backward. A forward axis consists of either the context node or nodes that appear after the context node in an XML document. A backward axis consists of the context node together with nodes that appear before it in an XML document. The backward axis is also known as the reverse axis.

The XPath processor uses the position() function to evaluate the position of a node using the information about the type of axis. If an axis is forward, the position of a node in the nodeset obtained from an XPath tree model is equal to the position of the node in the XML document order. For example, the first child element node in a tree model is the first child element node in an XML document order. If the axis is reverse, the position of a node in the nodeset obtained from an XPath tree model is equal to the position of the node in the reverse document order. For example, the first ancestor element node in a tree model is the last ancestor element in reverse document order. XPath consists of 13 axes. They are:

- `Child`: Contains child nodes of the context node. The attribute and namespace nodes are not the child nodes of the element node to which they belong. A child axis is the default axis and never returns any attribute and namespace nodes in the return nodeset.
- `Parent`: Contains the node that represents the parent of the context node. If the root node of a document is the context node, it does not have any parent.
- `Ancestor`: Contains the ancestors of a context node. The ancestors of a context node include parents of the context node, parents of the parent, and other parents in the hierarchy. The ancestor axis always includes the root node except when the root node itself is defined as the context node. The root node does not have any ancestors.
- `Descendant`: Contains the descendants of a context node. The descendants of a context node include the child nodes of the context node, child nodes of child nodes, and other child nodes in the hierarchy. As a result, the child axis is a part of the descendant axis. It does not contain the attribute and namespace nodes.
- `Descendant-or-self`: Contains descendant nodes together with the context node.
- `Ancestor-or-self`: Contains ancestor nodes together with the context node.
- `Following-sibling`: Contains all element nodes that are sibling of the context node and appear after the context node in an XPath data model. If the context node is either an attribute node or a namespace node, then the following-sibling axis does not include any node. For example, the customerdetails.xml file contains the context node as the node that represents the <Department> element with attribute value, Administration. The XPath expression that uses the following-sibling axis to retrieve the <Department> element node with attribute value, Sales, is:

  ```
  child::Department [@name=Administration]/following-sibling::Department [@name=Sales]
  ```
- `Preceding-sibling`: Contains all element nodes that are sibling of the context node and precedes the context node in the data model. If the context node is either an attribute node or a namespace node, the preceding-sibling axis does not include any node. For example, the customerdetails.xml file contains the context node defined as the node that represents the <Department> element with attribute value, Sales. The XPath expression that uses the preceding-sibling axis to return the <Department> element with attribute value, Administration is:

  ```
  child::Department [@name=Sales]/preceding-sibling::Department [@name=Administration]
  ```
- `Following`: Contains all nodes that appear after the context node in an XML document. It excludes the descendant nodes, attribute nodes, and namespace nodes.
- `Preceding`: Contains all nodes that appear before the context node in an XML document except the ancestor node, attribute node, and the namespace node.
- `Attribute`: Contains the attributes of a context node. It includes a node only if an element node is the context node. In all other cases, the attribute axis is empty.
- `Namespace`: Contains the namespace nodes of the context node. It includes a node only if an element node is the context node. If the context node is other than an element node, the attribute axis is empty.
- `Self`: Contains only the context node.

Figure 5-2 shows the data model for an XML document:

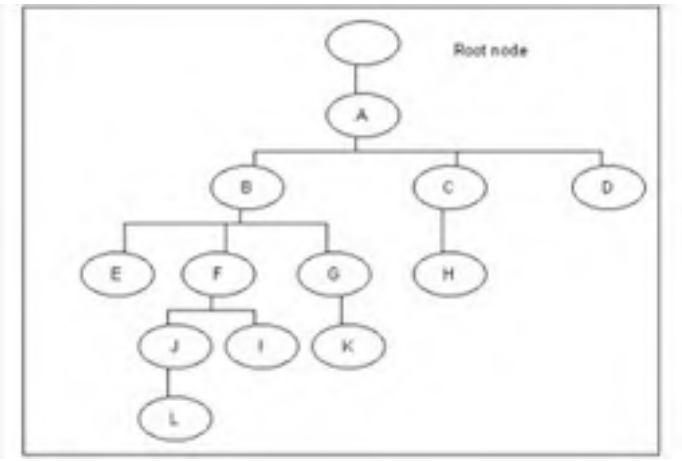


Figure 5-2: Data Model

You can determine various nodes corresponding to a specific axis by choosing a context node.

Table 5-2 lists the nodes corresponding to various axes, choosing node F as the context node:

Table 5-2: Nodes Corresponding to an Axis

Axis Name	Nodes
Self	F
Child	I, J
Parent	B
Ancestor	B, A, root node
Descendant	I, J, L
Descendant-or-self	F, I, J, L
Ancestor-or-self	F, B, A, root node
Following-sibling	G
Preceding-sibling	E
Following	G, K, C, D, H
Preceding	E

The child, parent, descendant, descendant-or-self, following-sibling, following, attribute, namespace, and self axes are forward axes. The ancestor, ancestor-or-self, preceding-sibling, and preceding axes are reverse axes.

Performing the Node Test

A node test describes a test on those XPath tree nodes that an XPath processor selects along the specified location step axis. The nodes that succeed this test act as a nodeset for the next location step. A node test identifies a node within an axis. Various node tests that an XPath processor can perform on XPath expressions are:

- `QName`: Selects nodes that have a QName and represent principal nodes for an axis.
- `node()`: Selects all nodes regardless of their name and type.
- `text()`: Selects all text nodes.
- `comment()`: Selects all comment nodes.
- processing instruction(): Selects all processing instruction nodes.
- `prefix:*`: Selects principal nodes that belong to the namespace defined by a prefix.

Note The * node test selects all principal nodes for an axis. For a child axis, the principal node type is an element node. If the context node includes child nodes other than element nodes, the resultant nodeset does not include those nodes.

Listing 5-5 shows the content of the XML document, course.xml:

Listing 5-5: The course.xml Document

```
<?xml version="1.0" ?>
<!--An example XML representation of the courses in a college-->
<courses>
<college>St. Peters</college>
<branches>
<branch name="EC">
<semester number="1">
<subject name="Digital Circuits">
First subject in semester 1 of EC branch.
</subject>
<subject name="Analog Circuits">
</subject>
<subject name="Electromagnetics">
</subject>
</semester>
<semester number="2">
<subject name="Radar Theory">
</subject>
<subject name="Satellite Communication theory">
</subject>
<subject name="Active Networks">
</subject>
</semester>
</branch>
<branch name="CS">
<semester number="1">
<subject name="Microprocessor">
</subject>
<subject name="Data Structure">
</subject>
<subject name="Computer System Organization">
```

```
</subject>
</semester>
<semester number="2">
<subject name="Operating System">
</subject>
<subject name="Java">
</subject>
</semester>
</branch>
</branches>
<CommonSubjects>
<Common name="PDC"> </Common>
<Common name="NA"> </Common>
<Common name="CP"> </Common>
</CommonSubjects>
</courses>
```

In the above listing, the course.xml document contains information about the semester-wise courses for two disciplines, EC and CS. The course.xml document also lists the subjects common for these two disciplines.

The following location step expressions demonstrate how to implement a node test:

- `child::subject`: Selects the <subject> child nodes of the context node. If the context node is the <semester> element with the number attribute value, 2 in the EC branch, the returned nodeset contains three <subject> child nodes. In addition, if the <courses> element is the context node, the location step expression returns an empty nodeset. This is because the <courses> element consists of the <subject> child node as its descendant node.
- `child::*`: Selects all element nodes that are child nodes of the context node. If the <courses> element is the context node, the expression returns a nodeset that contains the <college>, <branches>, and <CommonSubjects> element nodes.
- `child::text()`: Selects text nodes that are child nodes of the context node. For example, the context node is the <subject> element node with the name attribute value, Digital Circuits in semester 1 of EC branch. The XPath expression, returns the nodeset that consists of one text node having the string value, First subject in semester 1 of EC branch.
- `child::branch/descendant::subject`: Selects <subject> element nodes that are descendant nodes of the <branch> child node of the context node. If you select the <branches> element node as the context node, the expression returns a nodeset that contains 11 <subject> elements. This is because all <subject> element nodes are descendant nodes of the <branches> element node.

In the above examples, the XPath expressions use the relative location path. Alternatively, you can use an absolute location path to retrieve specific data from an XML document. For example, the location step that selects all element nodes that are child nodes of the root node expression is:

```
/child::*
```

There is only one element node for a root node. As a result, the above location step returns a nodeset that consists of only the <course> element node.

Setting Predicates

The location step consists of a predicate that describes the conditions that a node must satisfy to be selected. A predicate acts as a filter for the nodeset that the node test returns. The resultant nodeset of a location step includes nodes that satisfy the predicate. You must write a predicate within square brackets. The comparison operators that a predicate uses to compare a node property with some specific value are: =, !=, >, <, >=, or <=. A node property can be the attribute value or sibling order value of a node, which the position() XPath function returns.

The following location step expressions use Listing 5-4 to demonstrate how to implement predicates:

- `child::subject [position()=2]`: Selects the second <subject> child node of the context node. If the context node is the first <semester> element node in the EC branch, the expression returns the second <subject> element node with the name attribute value, Analog Circuits.
- `preceding::branch [attribute::name]`: Selects the <branch> element nodes that possess a name attribute and precedes the context node. If the context node is the <subject> element node with the name attribute value, Microprocessor, the expression returns the <branch> element node with name attribute value, EC.

Each of the above expressions returns a nodeset that consists of a single node. XPath functions you define in a predicate can also return more than one node. For example, the following location step selects all <subject> element nodes that appear after position 1 and represent child nodes of the context node in Listing 5-4:

```
child::subject [position()>1]
```

If the context node is the <semester> element node with number attribute value, 1 in EC branch, the expression returns second and third <subject> element nodes in the nodeset.

Using XPath with PHP

Extensible Stylesheet Language (XSL) converts an XML document to another data format. Different XSL transformations use same XML document to generate different output, such as an HTML Web page.

Accessing XPath Using DOM

PHP uses a tree-based approach called DOM to parse an XML document. This approach helps create and manipulate the hierarchical tree structure of an XML document. You can implement DOM in any programming language, such as PHP, Java, and Visual Basic (VB). Implementing DOM in PHP, you can access an XML document using XPath. DOM is a parser that accesses and manipulates structured data by representing a document in the form of a tree hierarchy of objects.

Objects represent different structures that occur within an XML document. For example, the Element object represents elements and Attr objects represent attributes. Each object consists of standard properties and methods that navigate the object tree and access specific elements, attributes, or character data. The XPath processor can navigate a document tree using the parent-child relationship that exists between tree nodes. Node properties extract all information required from a document tree.

PHP consists of XPath classes that make the DOM parser flexible. The XPath classes build a collection of nodes that match the criterion specified in an XPath expression. The XPath classes available in PHP are XPathContext and XPathObject.

The XPathContext class sets up a context node for all XPath evaluations. You can create an XPathContext class object by calling the xpath_new_context() function. This function must be passed as a reference to a DOM object. The XPath evaluations provide instances of the XPathObject class.

The xpath_eval() method of the XPathContext class creates an instance of the XPathObject class. The xpath_eval() method accepts the XPath address as its argument. The xpath_eval() method returns an instance that contains the nodeset matching a specified XPath expression. As a result, the PHP XPath implementation to query an XML document parses the document into a DOM tree.

For example, the XML document, books.xml, lists information about books published by a publisher.

Listing 5-6 shows the content of the books.xml document:

Listing 5-6: The books.xml Document

```
<?xml version="1.0"?>
<books>
<book>
<title>Introduction to computers</title>
<publisher>Global Education Ltd.</publisher>
<author>John Mitchell</author>
<price>500</price>
<publishedYear>1993</publishedYear>
</book>
<book>
<title>C Programming</title>
<publisher>Global Education Ltd.</publisher>
<author>Taub Schiling</author>
<price>900</price>
<publishedYear>1996</publishedYear>
</book>
<book>
<title>Operating Systems</title>
<publisher>Global Education Ltd.</publisher>
<author>David Kennedy</author>
<price>1800</price>
<publishedYear>1993</publishedYear>
</book>
</books>
```

In the above listing, the books.xml document contains information about books, such as the book title, author, price, and date of publishing, published by the publisher, Global Education Ltd.

For example, you need to solve the query to search titles and authors of the books published by Global Education Ltd. in 1993. You can execute this XPath query using the PHP document, books.php, as shown in Listing 5-7:

Listing 5-7: Executing the XPath Query Using DOM Implementation in PHP

```
<?php>
$doc=xmldocfile("books.xml");
$xpath=$doc->xpath_new_context();
$output=$xpath->xpath_eval("/books/book[normalize-space(publisher/text())='Global Education
Ltd.' and normalize-space(publishedYear/text())='1993']");
$nodeset=$output->nodeset;
foreach ($nodeset as $node)
{
   foreach ($node->child_nodes() as $children)
   {
      if ($children->node_type()==XML_ELEMENT_NODE)
      {
         if ($children->tagname()=="title")
         {
```

```
            print ("Title:");
            foreach ($children->child_nodes()as $subcontent)
            {
               if ($subcontent->node_type()==XML_TEXT_NODE)
               {
                  print ($subcontent->content);
               }
            }
            print ("::");
         }
         if ($children->tagname()=="author")
         {
            print ("Author:");
            foreach ($children->child_nodes() as $subcontent)
            {
               if ($subcontent->node_type()==XML_TEXT_NODE)
               {
                  print ($subcontent->content);
               }
            }
            print ("<br />");
         }
      }
   }
}
?>
```

In the above listing, the PHP code, books.php, selects the title and author name of all books published by Global Education Ltd. in 1993 from the books.xml document. The xpath_eval() method evaluates the XPath expression provided as its argument.

Figure 5-3 shows the output of the PHP code in the books.php document:

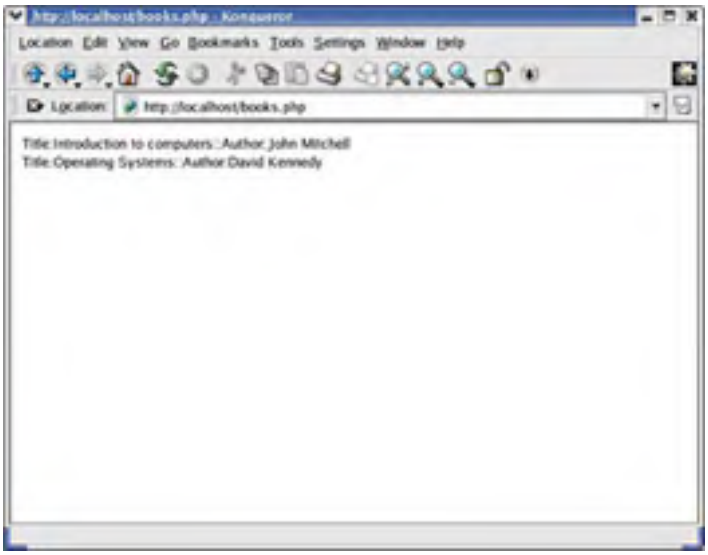


Figure 5-3: Output of the Code in the books.php Document

Note The PHP XPath implementation performs slowly for large documents and is resource intensive.

Accessing XPath Using SAX

SAX is an event-driven model for parsing an XML document. When the SAX parser encounters an XML construct, such as a tag, it generates an event. These events are passed to event handlers, which, in turn, provide access to the content of a document. You can parse an XML document using XPath expressions in PHPs SAX implementation.

You can set handlers for XML location paths using the XML parsing class, path parser (class_path_parser.php). For each handler, you can set up a PHP function that the parser calls whenever it finds an element matching the given location path. When the parser detects an element that matches the location path, the function accepts element names, attributes, and content. A function can handle multiple location paths.

For example, the XML document, employees.xml, contains information pertaining to employees in a company.

Listing 5-8 shows the content of the employees.xml document:

Listing 5-8: The employees.xml Document

```
<?xml version="1.0"?>
<employees>
<employee name='Peter'>
</employee>
<employeeId>e01</employeeId>
<department>administration</department>
<employee name='Julie'></employee>
<employeeId>e02</employeeId>
<department>sales</department>
<employee name='Taub'></employee>
<employeeId>e03</employeeId>
<department>administration</department>
<employee name='David'></employee>
<employeeId>e04</employeeId>
<department>Planning</department>
</employees>
```

You can implement SAX in PHP to retrieve XML elements that match the pattern described by an XPath expression. For example, you can retrieve the name of all employees from the employees.xml document.

Listing 5-9 shows the content of the employees.php document:

Listing 5-9: Retrieving Employee Names

```
<?php
include_once("/class_path_parser.php");
function result($name,$attribs,$content)
{
   print("$name  ");
   print($attribs [name]);
   {
      print ("<br/>");
   }
}
$parser = new Path_parser();
$parser->set_handler("/employees/employee", "result");
if(!$parser->parse_file("employees.xml"))
{
   print("Error:".$parser->get_error()."\n");
}
?>
```

The parse_file() method parses an XML document from a file or a URL. The syntax of the parse_file() method is:

```
boolean parse_file (string $xml)
```

In the above syntax, the parameter, $xml, is the name of the file or URL that contains the file that the parser needs to parse. The parse_file() method returns TRUE, if the file is successfully parsed, otherwise it returns FALSE.

> **Tip** If the parser does not successfully parse a file, the parse_file() method returns an error message. You can use the get_error() method to retrieve the error message.

The set_handler() method processes XML element nodes that match the pattern specified by an XPath expression. The syntax of the set_handler() method is:

```
set_handler (string $path, string $handlername)
```

The above syntax shows that the $path represents the absolute path in an XML document. The $handlername must have three arguments: $name, $attribs, and $content. The $name argument denotes the element name, $attribs denotes element attributes, and $content denotes the text within the element node.

Figure 5-4 shows the output of the PHP code, employees.php:

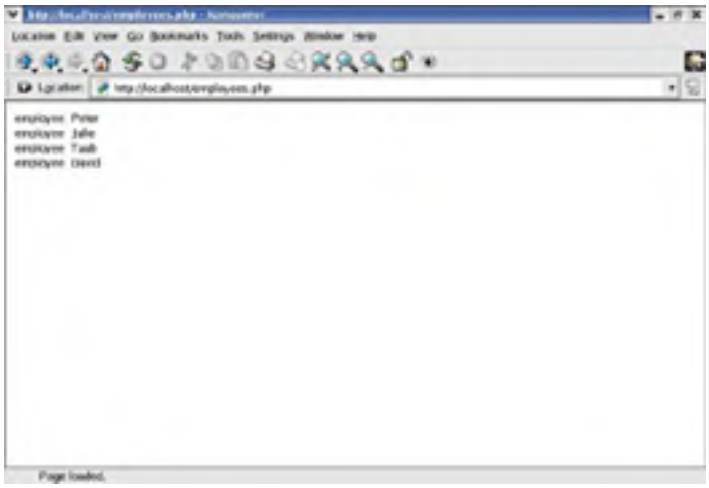


Figure 5-4: Output of the employees.php Code

You can parse an XML file using the Path_parser class and set handlers for specific XML elements defined by an XPath expression. The handler name can accept the element name, attributes, and the content.

Accessing XPath Using XSLT

XSLT is a language that can transform XML documents into a text-based format, such as HTML or another XML-based structure, such as the Web Distributed Data eXchange (WDDX) format. To perform an XSLT transformation, you require an XML document that XSLT should transform, XSLT stylesheet, and an XSLT engine.

The XSLT stylesheet contains the instructions that you need to write to achieve the required transformation. You use XSL, an XML-based language to create stylesheets. You need to define the output layout or result tree and the input document or source tree from where the XSL retrieves data. An XSLT engine transforms an XML document into other document types using a stylesheet. You can use XPath expressions in the XSLT stylesheet to retrieve elements from an XML document and transform these elements into the required format. For example, the XML document, chapter.xml contains information about the headings in a chapter.

Listing 5-10 shows the content of the chapter.xml document:

Listing 5-10: The chapter.xml Document

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="chapter.xsl"?>
<chapter>
<h>Heading 1</h>
<h>Heading 2</h>
<h>Heading 3</h>
</chapter>
```

You can create a stylesheet file to convert the chapter.xml document to an HTML document.

Listing 5-11 shows the content of the chapter.xsl stylesheet file:

Listing 5-11: The chapter.xsl Stylesheet File

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
<HTML>
<BODY>
<xsl:for-each select="/chapter/h">
<S><xsl:value-of select="."/></S>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

In the above listing, <xsl: for-each> applies to any <h> elements, which are the child nodes of a <chapter> root element. The output contains <S> elements with the value of <xsl:value-of> element as its content.

You need to apply the stylesheet to the chapter.xml document.

Listing 5-12 shows output of the chapter.xml file after you apply the stylesheet to the chapter.xml document:

Listing 5-12: Output After Applying the Stylesheet

```
<HTML>
<BODY>
<S>Heading 1</S>
<S>Heading 2</S>
<S>Heading 3</S>
</BODY>
</HTML>
```

The above listing shows the HTML format for the chapter.xml document.

XPath expressions enable you to specify the location in an XML document and process this information using XSLT. You can process specific XML elements using XPath expressions with DOM, SAX, or XSLT. Parts of XSLT operate using a subset of XPath. You can use this subset to test if an XPath node matches a pattern defined by the location path. XPath operates only on the logical structure of an XML document.

Chapter 6: PHP and XML-Remote Procedure Calls

Download CD Content

Extensible Markup Language-Remote Procedure Call (XML-RPC) is a protocol that allows applications running on various operating systems to make procedure calls on a remote server for executing a procedure.

Implementing XML-RPC in PHP helps build network-based services. You can implement XML-RPC in PHP to encode and decode Remote Procedure Call (RPC) messages and create and manage a remote server. This implies that you can use PHP scripts to invoke procedures on remote servers by transferring messages that are in the XML format.

You can access different remote procedures directly from a PHP script using standard client-server protocols, such as HyperText Transfer protocol (HTTP). After executing a remote procedure, the remote server returns a value, which can be used in other applications.

This chapter explains the XML-RPC protocol, and how to create and transfer a request to a server using XML-RPC. This chapter also explains the data types that the XML-RPC protocol supports.

Understanding XML-RPC

RPC provides a client-server framework that allows methods or procedures on the server to be remotely executed by the client in a secure and efficient manner. It uses HTTP to transfer data between the client and server. The client that invokes the procedure may be either on the same computer as the server or a different computer on the network. You can implement RPC over the Web using XML encoding and the HTTP protocol for transferring data.

The XML-RPC protocol encodes remote procedure calls in XML. It uses HTTP to transfer client requests to a server and receive the responses. XML-RPC uses an XML format to pass the method parameters and receive the returned values from the server. XML-RPC implements client requests and server responses in various languages, such as Lisp, JavaScript, C, Perl, and PHP. XML-RPC supports various data types, such as integer, string, Boolean, and double.

Introducing RPC

RPC is a programming interface that allows a program to use the services of another program available on a remote computer. The calling program transfers a message to the remote program, which executes the procedure and returns the result to the calling program.

Figure 6-1 shows the communication in a network using RPC:

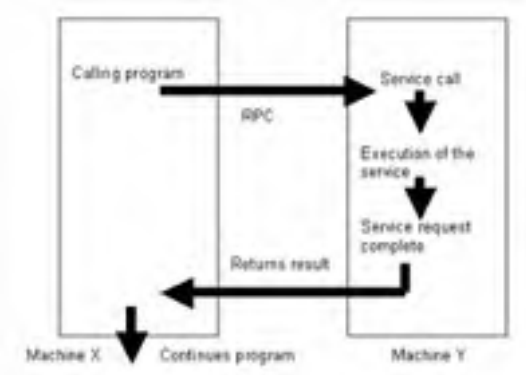


Figure 6-1: Network Communication using RPC

The client process, Computer X, transfers a request to the Computer Y server process, and passes the parameters for the called procedure. The server executes the called procedure and returns the result to the client.

RPC consists of three components, which are:

- An Application Programming Interface (API): Handles procedure calls on the client-side and executes client requests on the server-side.
- A set of rules: Encode and decode RPC requests and responses.
- A network transmission layer: Establishes communication between the client and server.

RPC provides a framework to execute the procedures on a remote computer in an efficient manner. To make a remote procedure call:

1. The RPC client invokes a remote procedure.
2. The RPC client encodes the procedure call request, along with its parameters, in a form suitable for transmission over a network transmission layer, such as HTTP.
3. The network transmission layer transfers the procedure call request to the RPC server in the form of packets.

4. The RPC server extracts the procedure name and its parameters from the request packet.
5. The RPC server invokes the specified procedure and obtains the result.
6. The RPC server encodes the result of the procedure in the form of response packets and transmits the response packets to the RPC client.
7. The RPC client receives the response packet, decodes it, and extracts the return value of the procedure. The RPC client can use the value that the called procedure returns in other programs.

There are two methods to implement an RPC, XML-RPC and Simple Object Access Protocol (SOAP). To create an XML-RPC call, the client issues a request to the server specifying a method name, its parameters, and the server name. The methods that you define in PHP are called XML-RPC methods and the parameters correspond to the arguments that you pass to these XML-RPC methods. The arguments can be of any data type, such as integer and Boolean.

The client packages the request in the XML format and issues an HTTP-POST request that transfers the message to the server. The HTTP-POST request informs the server that the client is ready to transfer data. The server calls the requested method and passes the parameters to this method. The method on the server returns a response, and the server packages the response into the XML format. The server returns the response to the client, which parses the XML package to retrieve the returned value. The value that a remote method returns is called server response.

SOAP is a specification for creating structured data packets that an application needs to transfer across a network. SOAP enables information exchange among the computers in a network. Unlike XML-RPC, it is not necessary that SOAP returns the HTTP status code, 200 OK, to indicate the server response. Instead, the SOAP specification uses standard HTTP error code to identify the successful processing of a client request.

Working with the XML-RPC Protocol

The XML-RPC protocol is a specification with a set of implementations that allow software running on different operating systems to make remote procedure calls. The XML-RPC request and response that you create consists of an HTTP header and an XML body. The use of the HTTP protocol ensures that XML-RPC client requests are synchronous and stateless.

A synchronous XML-RPC client request means that the server immediately responds to the client request. The XML-RPC server transfers the response on the same HTTP layer as the client request. A stateless XML-RPC client request means a client request does not preserve any information from one request to another. For example, if a client invokes a remote method on a server, receives the server response, and again invokes the same method on the server, the client requests are regarded as two different requests.

The XML-RPC client does not perform any other function until it receives the server response. A client request can invoke only a single remote method and each server response can return only a single value. The single value that the XML-RPC server returns can be either an array or a structure with multiple values. There are two types of XML-RPC servers:

- Mini Web serverM: Processes an XML-RPC request.
- XML-RPC listener: Assists the Web server in processing an XML-RPC request.

XML-RPC packages data for information exchange between two computers by defining the structure and format of the procedure calls and responses. XML-RPC is useful in applications where a server provides various services, such as remote procedure access to its clients. To use XML-RPC, you need to install the XML-RPC package on your computer.

Figure 6-2 shows the architecture of XML-RPC:

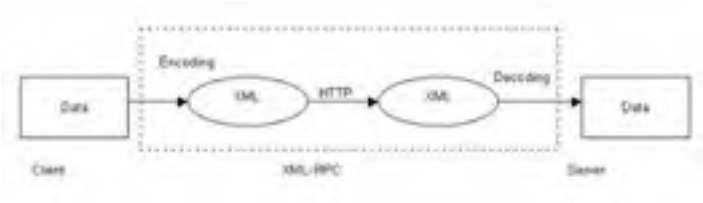


Figure 6-2: Architecture of XML-RPC

The data types that XML-RPC supports are:

- `<boolean>` : Represents Boolean values, 0 or 1. The following syntax shows how to represent the <boolean> data type in XML-RPC:

  ```
  <value><boolean>n</boolean></value>
  ```

  In the above syntax, n represents a Boolean value.

- `<double>`: Represents floating-point data. The following syntax shows how to represent floating-point data in XML-RPC:

  ```
  <value><double>n</double></value>
  ```

  In the above syntax, n denotes a floating-point value.

- `<int> or <i4>`: Represents signed 32-bit integers. You can represent the integer data type in XML-RPC using the following syntax:

  ```
  <value><i4>n</i4></value>
  <value><int>n</int></value>
  ```

  In the above syntax, n denotes an integer value.

- `<string>`: Represents the ASCII string. The following syntax shows how to represent the string data type in XML-RPC:

  ```
  <value>s</value>
  ```

  In the above syntax, s denotes a string value. The string data type is the default value. If you do not indicate the variable data type, the value inside the <value> tag is called as the string value.

- `<base64>`: Represents binary data encoded with BASE64. The following syntax shows how to represent the binary data type in XML-RPC:

  ```
  <value><base64>n</base64></value>
  ```

  In the above syntax, n denotes a binary value.

- `<array>`: Represents an array of data. The <array> element contains a single <data> element, which consists of many <value> elements. The following syntax shows how to represent the <array> element in XML-RPC:

  ```
  <value>
  <array>
  <data>
  <value>n</value>
  ......
  <value>n</value>
  </data>
  </array>
  </value>
  ```

- `<struct>`: Represents an associative array that contains string values. It contains many <member> elements, with each element containing a <name> and a <value> element. The following syntax shows how to represent the <struct> element in XML-RPC:

  ```
  <value>
  <struct>
  <member>
  <name>n</name>
  <value>v</value>
  </member>
  ....
  <member>
  <name>n</name>
  <value>v</value>
  </member>
  </struct>
  </value>
  ```

- `<dateTime.iso8601>`: Represents the date and time according to the International Standard Organization (ISO) 8601 standard. The following syntax shows how to represent date and time in XML-RPC:

  ```
  <value><dateTime.iso8601>YYYYMMDDTHH:MM:SS</dateTime.iso8601></value>
  ```

Note An XML-RPC array data type is similar to an indexed array, and an XML-RPC struct data type is similar to an associative array. For example, the array data type is represented as $array=array (‘one’, ‘two’), and the struct data type is represented as $struct=array (‘var1’=’one’, ‘var2’=’two’).

Listing 6-1 shows the example of an XML-RPC request:

Listing 6-1: XML-RPC Client Request

```
POST /myserver.php HTTP/1.0
User-Agent: Konqueror
Host: 192.148.0.146
Content-Type: text/xml
Content-Length: 256
<?xml version="1.0"?>
<methodCall>
<methodName>getCharacter</methodName>
<params>
<param>
<value>
<string>Phoenix</string>
</value>
</param>
</params>
</methodCall>
```

In the above listing:

- The server transfers the request to the Uniform Resource Identifier (URI), /myserver.php.
- The client calls the getCharacter() method on the server to obtain the number of characters in the Phoenix file.
- The <methodCall> structure consists of the <methodName> element, which specifies the name of the method on the remote server that an XML-RPC client needs to call, which is the getCharacter() method. The <methodName> element is a string that can contain characters such as alphabets, numerics, underscore, period, colon, or slash.
- The server interprets the <methodName> string either as the location of a file or name of a file that contains the program that a client requests for execution.

Note The client request should contain the <params> element even if a remote method does not need any parameters.

The <methodCall> structure includes the <params> element to indicate the parameters that a remote method accepts. The <param> element represents individual arguments of a remote method. The User-Agent specifies the medium through which the client and server communicate. The Host specifies the IP address of the server. In addition to the User-Agent and Host, you should also indicate the Content-Length, which represents the accurate character length of the message.

The path that you describe after the POST method indicates the script that receives the client data. The HOST attribute, in the header of the XML-RPC client request, indicates the name of the server that services the XML-RPC request.

Note You can download the XML-RPC package for PHP either in a zipped format or tar.gz archives from the following Web site: http://xmlrpc.usefulinc.com/php.html.

The XML-RPC server decodes and interprets the client request and provides result to the client. If no error occurs while processing a client request, the XML-RPC server returns the response. Every XML-RPC response should return the 200 OK HTTP status code even if an error occurs while processing the XML-RPC client request. The <methodResponse> structure returns a single value and so the <params> element consists of a single <param> element.

Listing 6-2 shows the example of an XML-RPC response:

Listing 6-2: XML-RPC Server Response to a Successful Client Request

```
HTTP/1.0 200 OK
Connection: close
Server: 192.168.0.146/myserver.php
Content-Type: text/xml
Content-Length: 210
<?xml version="1.0"?>
<methodResponse>
<params>
<param>
<value>
<string>100</string>
</value>
</param>
</params>
</methodResponse>
```

The above listing shows the value that the getCharacter() method returns to the XML-RPC client.

If the server does not process a client request successfully, the <fault> element replaces the <params> element in the <methodResponse> structure.

Listing 6-3 shows the XML-RPC response to an unsuccessful request:

Listing 6-3: XML-RPC Server Response to an Unsuccessful Client Request

```
HTTP/1.0 200 OK
Connection: close
Server: 192.168.0.146/RPC
Content-Type: text/xml
Content-Length: 210
<?xml version="1.0"?>
<methodResponse>
<fault>
<value>
<struct>
<member>
<name>faultCode</name>
<value>
<int>x</int>
</value>
</member>
<member>
<name>faultString</name>
<value>
<string>No characters in the file</string>
</value>
</member>
</struct>
</value>
</fault>
</methodResponse>
```

In the above listing:

- The <fault> element replaces the <params> element, which appears in the server response to a successful client request.
- The <fault> element contains the x fault code along with the fault string that describes the type of fault. The fault string indicates that there are no characters in the Phoenix file.
- The <struct> element contains two elements, faultCode and faultString.

- The faultCode element is a numeric value corresponding to a fault.
- The faultString element is a string that contains the text description of the fault.

Note The <methodResponse> structure cannot simultaneously include the <params> and <fault> elements.

Extending the XML-RPC Protocol

Implementing XML-RPC in PHP provides the introspection feature. The introspection feature lists and describes the services that an XML-RPC server provides. It provides a way to query methods and obtain the description of methods from the server. PHP XML-RPC provides the following introspection features:

- `system.listMethods()`: Generates a list of methods available with the XML-RPC server.
- system.describeMethods(): Describes the server methods. The description includes the expected arguments for the method, method return type, and optional arguments for the method.
- `system.methodHelp()`: Generates documentation for a particular method. The system.methodHelp() method accepts the name of a method that the XML-RPC server implements as its argument and returns a string that describes the use of that method.
- `system.methodSignature()`: Returns the signature for a particular method. The signature describes the data type of the parameters that a client should pass to a method. It also indicates the return type for a method.

In addition to features that the XML-RPC protocol supports, the protocol can support features, such as an asynchronous client request and authentication of Web services. The three extensions of XML-RPC protocol are:

- Asynchronous protocol for XML-RPC
- State-preserving protocol for XML-RPC
- A protocol with authentication for XML-RPC

In the asynchronous XML-RPC protocol, the server does not immediately transfer the response to a client request. The server transfers the response to the client at a time other than the time when the client makes a request. For implementing the asynchronous XML-RPC protocol, both the XML-RPC client and server process should be able to perform the functions of each other. This means that the client process can perform the functions of the server process and the server process can perform the functions of the client process. The client and server processes should contain the code for handling asynchronous responses.

To create a system that uses the asynchronous XML-RPC protocol, the server process should return a unique identifier. To communicate using the asynchronous XML-RPC protocol:

1. The client makes a request to the server.
2. The server returns an ID for the request without processing it.
3. The client receives the ID and continues its current operation.
4. The server processes the client request and creates a result container.
5. The server makes a call to the client by passing the ID and the result of the client request.
6. The client receives the response of the client request.
7. The client processes the response.

The XML-RPC protocol is stateless and does not preserve any information from one client request to another. However, for applications, such as Web-based shopping carts, it is necessary to preserve the server state for use in another client request. For example, a client request creates a particular method on a server and another client request requires a change in the created method. In such situations, it is necessary to preserve the server state. To create subsequent requests that utilize the server states, the client uses the identifiers held either in cookies or in the Uniform Resource Locator (URL) of the page.

You can implement a state preserving system in XML-RPC by representing the first argument of all the remote procedure calls as a session identifier. XML-RPC keeps track of the client requests, and uses the information of an earlier request in the consequent request.

The authentication protocol for XML-RPC ensures that the message content is from an authorized client. As a result, the message cannot be modified during transfer through the HTTP transport layer. The authentication protocol is based on Message Authentication Code (MAC), which is a keyed form of the request message. This means that the client attaches a secret code to the request message.

For many-to-many client/server relationships, it is not possible to maintain databases on each server. As a result, both the client and server maintain keys with a keyserver that authenticates the clients to the servers. If your Web server supports HTTP, the client can invoke a remote method on the server by supplying a user name and password that are validated by the server.

Using XML-RPC for Web Services

A Web service is an application that runs on a Web server and enables the client programs to call remote procedures using the HTTP transport layer. The process of data exchange is similar in Web services and Web browsers. The Web browser transfers a request to a Web site in the HTML form and obtains a Web page in response. A Web service uses the XML format while a browser uses the HTML format to transfer data.

A Web service packages a collection of functions as a single entity, and publishes these functions over the Web to be used by other programs.

A Web service contains three components: service provider, service discovery agency, and service requestor. The service requester requests the service provider for the execution of a Web service. The service requester searches the description through the discovery agency and uses the service description to bind with the service provider to interact with the Web service. The service provider processes a Web service request, provides the Web service description, and publishes it to a requester or discovery agency. The service discovery agency publishes a Web service description to multiple service registries.

Figure 6-3 shows the Web service architecture:

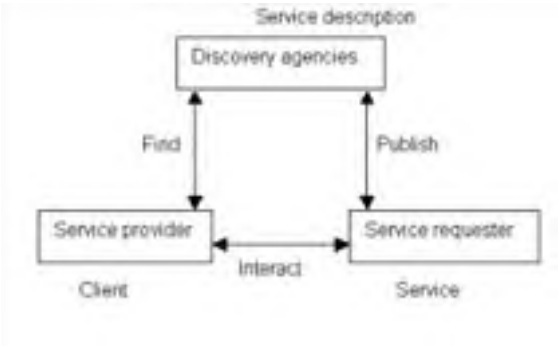


Figure 6-3: The Web Service Architecture

The components of a Web service interact to perform the following functions:

- Define the interface and invocation methods of a Web service.
- Publish the Web service to the intranet or Internet storage locations so that the end users can easily locate the service.
- Locate a Web service.
- Invoke a Web service for use by end users.

A Web service has the following advantages:

- Provides interoperability that helps to execute Web services regardless of the platform used.
- Permits just-in-time integration by dynamically locating and invoking the Web service.
- Allows a change in either a Web service or the application that uses it, without affecting the other components.

XML-RPC helps to access Web services for many end users. A Web service executes procedures available on a remote computer from another computer. It provides interoperability among applications running on different computers. For example, there are paid Web services, where you need to pay a prescribed fee for information exchange. An application can access a Web service over the Internet using standard Web protocols, such as HTTP. A Web service is heterogeneous because it allows communication between a client and server running on different platforms.

The XML-RPC protocol acts as a middleware between Web clients and remote Web services. XML-RPC accepts client requests, translates the request into an XML format, and transfers it using the HTTP protocol to access the remote Web service.

Figure 6-4 shows how to access a Web service using the XML-RPC protocol:

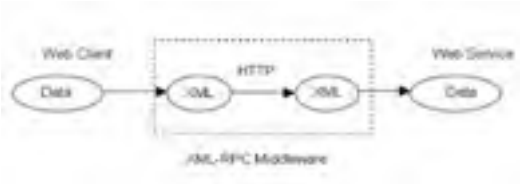


Figure 6-4: Accessing a Web Service Using the XML-RPC Protocol

PREVIOUS NEXT

Implementing XML-RPC in PHP

The XML-RPC client and server classes exist for various languages including PHP. The xmlrpc.inc file provides XML-RPC client functions, and the xmlrpcs.inc file provides XML-RPC server functions. To check if the XML-RPC server successfully executes a client request, include the .inc files in the same directory as the PHP code.

Creating a Client Application

The XML-RPC client transfers a request to execute a particular method on the server. The request message consists of the arguments that the remote method accepts.

To create an XML-RPC client request using the XML-RPC class, the client needs to generate a client object. The client transfers and receives data from the XML-RPC server using the client object. The xmlrpc_client class creates a client object. The information that the client object constructor receives from the server includes:

- The path to the script handling that the XML-RPC client requests.
- The hostname or the IP address of the server.
- The port number of the server.

The syntax of the xmlrpc_client class constructor is:

```
$client=new xmlrpc_client($serverpath, $hostname, $port);
```

Note The port parameter in the xmlrpc_client class constructor is optional. If you exclude the port parameter, its default value is set to 80 when you use HTTP as the transport layer.

The xmlrpc_client class constructor supports the following methods:

- `send() method`: Sends the XML-RPC client message to the server. The syntax of the send() method is:

  ```
  $response=$client->send ($xmlrpc_message, $timeout, $server_method);
  ```

  In the above syntax, $xmlrpc_message is an instance of the xmlrpcmsg class. The $timeout and $server_method parameters are optional. The $timeout parameter is set to 0, and the $server_method parameter to HTTP, if these are not specified in the syntax.

- `setDebug() method`: Enables the client to display the debugging information on the browser. When debugging mode is on, the client object displays the response received from the server. The debugging information helps you locate low-level errors, such as no network connection to the server and the timeout expired errors. You can locate these errors because debugging information displays data that the server returns. The syntax of the setDebug() method is:

  ```
  $client->setDebug($debugmode);
  ```

  ```
  $client->setDebug($debugmode);
  ```

  In the above syntax, the value of the $debugmode parameter is either 0 or 1. The value, 0, of the $debugmode parameter indicates that the client does not display debugging information on the browser. If the $debugmode parameter is set to 1, the client displays the debugging information on the browser.

- `setCredentials() method`: Enables you to set a user name and password for authorizing the client to access a server. The syntax of the setCredentials() method is:

  ```
  $client->setCredentials($user name, $password);
  ```

Note The xmlrpc_client class constructor supports the setCertificate() method, which allows you to define a certificate necessary for establishing a connection over the Hyper Text Transfer Protocol Secure Sockets (HTTPS) protocol.

To create a client request, you can invoke the methods defined in the sample file, server.php, which is downloaded with the xmlrpc library. For example, you can create a client request that calls the interopEchoTests.echoValue method in the server.php file. The interopEchoTests.echoValue method echoes the value passed to it. The interopEchoTests.echoValue method does not accept any parameters.

Listing 6-4 shows an XML-RPC client request created using the PHP language:

Listing 6-4: XML-RPC Client Request in PHP

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
$msg=new xmlrpcmsg("interopEchoTests.echoValue");
$client=new xmlrpc_client("/server.php", "localhost", 80);
$client->setDebug(1); //Debug mode is turned on.
$response=$client->send($msg); //Invokes the remote method.
?>
```

The above listing shows how the client request calls the interopEchoTests.echoValue() method on the server.php server file. The client message does not pass any arguments to the called method. As a result, the server response consists of the client message itself.

Instead of calling methods predefined in the server.php file, you can create a server and define methods within it. For example, create a server, myserver.php, which contains the examples.multiply() method to multiply two integer values.

Listing 6-5 shows an XML-RPC client request in PHP to call the examples.multiply() method:

Listing 6-5: XML-RPC Client Request that Invokes the examples.multiply() Method

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
$msg=new xmlrpcmsg("examples.multiply", array(new xmlrpcval(13, "int"),
new xmlrpcval(2, "int")));
$client=new xmlrpc_client("/myserver.php", "localhost", 80);
$client->setDebug(1);
$response=$client->send($msg);
?>
```

The above listing shows the client request to invoke the examples.multiply() method available in the myserver.php file. The client request passes two integer parameters, 3 and 2, to this method.

Similarly, you can create a client request to add three floating-point values.

Listing 6-6 shows the client request to add three floating-point numbers:

Listing 6-6: The Client Request to Add Three Floating-point Numbers

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
$msg=new xmlrpcmsg("addDouble", array(new xmlrpcval(3.4, "double"),
new xmlrpcval(2.5, "double"), new xmlrpcval(1.2, "double")));
$client=new xmlrpc_client("/myserver.php", "localhost", 80);
$client->setDebug(1);
$response=$client->send($msg);
?>
```

The above listing shows the client request that passes three parameters of double data type to the remote method on the myserver.php server.

Figure 6-5 shows the server response after servicing the request to add three floating-point values:

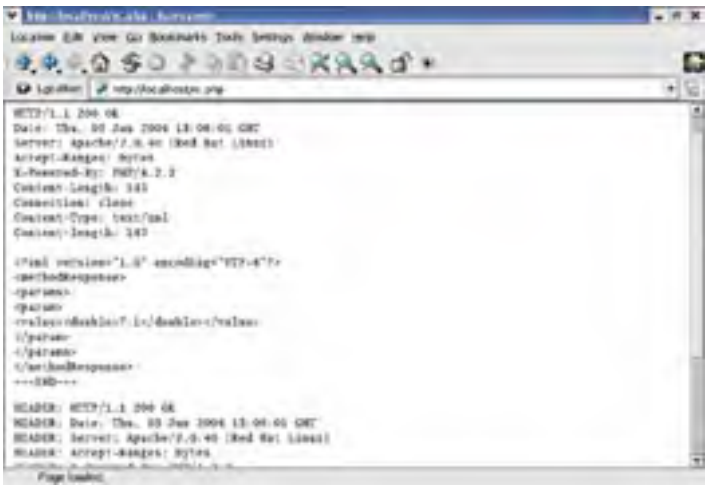


Figure 6-5: The Server Response of the Client Application to Add Double Values

Creating a Request using XML-RPC

After you create the client object, you should pass a message object to it. The XML-RPC client transfers the message object to the server. The message consists of the name of the remote method that a client needs to invoke, and the parameters that this method accepts. To create a client request, you need to create an instance of the xmlrpcmsg class. The xmlrpcmsg class provides the method name along with its parameters that an XML-RPC client needs to invoke on a server. The syntax of the xmlrpcmsg class is:

```
$msg=new xmlrpcmsg($method_name, $parameter_array);
```

In the above syntax, the $method_name parameter is the string that denotes the name of the method to be invoked on the server. The $parameter_array parameter is an array of objects that the xmlrpcval class creates.

An alternative method to represent the xmlrpcmsg class constructor is:

```
$msg=new xmlrpcmsg($method_name);
```

In the above syntax, the client does not pass any parameters to the remote method.

The xmlrpcval class encloses values of the specified data types to obtain the XML-RPC server response. It stores complex values using the XML-RPC data types. The following multiple forms show the use of the xmlrpcval class:

```
$val=new xmlrpcval();
$val=new xmlrpcval($stringvalue);
$val=new xmlrpcval($scalarvalue, "int"|"boolean"|"string"|"double"|"dateTime.iso8601"|"base64");
$val=new xmlrpcval($arrayvalue, "array"|"struct");
```

In the above forms, the first xmlrpc class constructor creates an empty value. The second constructor creates a string value. The third constructor consists of two parameters. The first parameter of the third constructor indicates that the xmlrpcval class creates a scalar value. The second parameter denotes the XML-RPC data type. The fourth constructor contains the $arrayvalue parameter, which is either a simple array or an associative array, depending upon the data type.

You should provide parameters within an array, even if the client has only one parameter to pass to the remote method. It is not necessary to provide parameters for the remote method during the initialization of the xmlrpcmsg class instance. You can use the addParam() method to add the method parameters after the XML-RPC client creates the xmlrpcmsg class instance. The syntax of the addParam() method is:

```
$msg->addParam($xmlrpcval);
```

For example, the code snippet showing the client request using the XML-RPC library classes, xmlrpcmsg and xmlrpcval, is:

```
$msg=new xmlrpcmsg("calculate.area", array(new xmlrpcval(14, "int"),
new xmlrpcval(23, "int")));
```

The above code shows the client request that invokes the calculate.area() method to calculate the area of a rectangle. The client passes two integer parameters, 23 and 14, which denote the length and breadth of the rectangle.

You can also provide a parameter array to the remote method. For example, the following code shows the client request that transfers an array of parameters to the remote method, examples.salary:

```
$input=array("Kim"=>2300, "Erwin"=>5000, "Crisp"=>9000);
while(list($key, $val)=each($input))
{
   $outp[ ]=new xmlrpcval(array("name"=>new xmlrpcval($key),"sal"=>new xmlrpcval($val,
   "int")), "struct");
}
$f=new xmlrpcmsg("examples.salary",array(new xmlrpcval($outp, "array")));
```

The above code shows the client request that invokes the examples.salary() method to sort the employee names on the basis of the salary. The input parameters passed to the remote method are the name and salary of the employee.

Sending a Request to the Server

After the client creates the client object and request message, it transfers the request message to the server. To transfer the client request to a remote server, the client creates a request message and passes it to the send() method. There are two ways to invoke the send() method:

```
$xmlrpc_resp=$client->send($msg);
$xmlrpc_resp=$client->send($msg, $timeout);
```

In the above syntax, the second form describes a timeout parameter after which the message is timed out. You should represent the timeout parameter in seconds. The $msg parameter specifies the method name and its parameters.

If the client request executes successfully, the send() method returns an instance of the xmlrpcresp class. If a low-level error occurs, the send() method returns 0. The low-level error occurs either if the client cannot connect to the server, or if the client cannot transfer the request to the server. The xmlrpcresp class contains the responses to the XML-RPC requests.

> **Note** Turn on the client object debugging mode to check the cause of the low-level error.

An XML-RPC server is a PHP script similar to an XML-RPC client. The XML-RPC server contains methods that a client can access remotely. It is not necessary that the name of the method, which a client uses, should be the same as the method name on the server. PHP maps the Web service API procedure names, and the name of the PHP functions that implement those procedures on the server. For example, the examples.multiply() method that the client invokes is mapped to the multiply() method on the server. The following code shows the multiply() method on the XML-RPC server:

```
function multiply($m)
{
   $s=$m->getParam(0);
   $t=$m->getParam(1);
   return new xmlrpcresp(new xmlrpcval($s->scalarval()*$t->scalarval(), "int"));
}
```

In the above code:

- The multiply() method accepts the xmlrpcmsg class instance as a single argument.
- The getParam() method accesses the parameters of the multiply() method as instances of the xmlrpcval class in the request message.
- An instance of the xmlrpcresp class returns the value of the multiply() method to the calling function.

You can create an instance of the xmlrpcresp class using one of the following syntaxes:

- `new xmlrpcresp (0, $errno, $errmsg)`: Contains three arguments. The first argument, 0, indicates that the value that a server returns is a fault condition. The second argument, $errno, indicates the fault number and the third parameter, $errmsg, indicates the error description.
- `new xmlrpcresp ($xmlrpcVal)`: Returns the value of the method if the client request is successfully executed.

To obtain the response to an XML-RPC client request, the server should execute the invoked method. A dispatch map stores the description of all methods defined in the XML-RPC server. A dispatch map is an associative array that maps the Web service API procedure names with the names of the PHP methods that implement those procedures on the server. The following code shows the dispatch map for the client request that calls the examples.multiply() method on the server:

```
//Represents the signature of the multiply() method.
$multiply_sig=array(array($xmlrpcInt, $xmlrpcInt, $xmlrpcInt));
//Represents the documentation for the multiply() method.
$multiply_doc='Multiplies two integers together and return an result type';
//Represents dispatch map.
$dm=array("examples.multiply"=>array("function"=>"multiply", "signature"=>$multiply_sig,
"docstring"=>multiply_doc));
```

The above code shows the dispatch map for the multiply() method. The dispatch map points to the signature of the multiply() method, along with the documentation for the multiply() method. PHP maps the examples.multiply() method to the PHP function that implements the called procedure on the server.

Each instance of the array in the dispatch map needs the following parameters:

- `function`: Represents the exact name of the PHP function on the server that defines the functioning of the method that the XML-RPC client calls. It is a string value.
- `signature`: Represents the data type of the parameters, along with the method return type, that you need to pass to the method. You can define multiple signatures so that a method can have different number and type of parameters. When the client invokes a particular method, the server verifies if the data type of the arguments passed by the client matches any of the signatures in the dispatch map. An error occurs if the data type of the arguments does not match.
- `docstring`: Represents a string that contains the documentation for a method on the server. The method documentation explains the operation that a method performs.

You need to pass the dispatch map as an instance of the xmlrpc_server class. To get the client request serviced by a server:

1. Include the xmlrpcs.inc file, in addition to the xmlrpc.inc file, in the server PHP script.
2. Create the xmlrpc_server class instance that accepts the dispatch map as its argument.

The xmlrpc_server class instance services the client request.

Listing 6-7 shows the PHP server script that services the client request to multiply two integer values:

Listing 6-7: PHP Server Script to Multiply Two Integers

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
include("/xmlrpc-1.0.99.2/xmlrpcs.inc");
$multiply_sig=array(array($xmlrpcInt, $xmlrpcInt, $xmlrpcInt));
$multiply_doc='Multiplies two integers together and return the result';
function multiply($m) {
$s=$m->getParam(0);
$t=$m->getParam(1);
return new xmlrpcresp(new xmlrpcval($s->scalarval()*$t->scalarval(), "int"));
}
$res=new xmlrpc_server(array("examples.multiply"=>array("function"=>"multiply",
"signature"=>$multiply_sig, "docstring"=>multiply_doc), "interopEchoTests.echoValue"
=>array("function" => "i_echoValue", "docstring" => $i_echoValue_doc)));
?>
```

The above listing shows the server script that services the client request to multiply two integer values.

Similarly, you can create the server that arranges employee names based on their salaries.

Listing 6-8 shows the PHP server script that services the client request to arrange employee names based on their salaries:

Listing 6-8: PHP Server Script to Sort Employees

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
include("/xmlrpc-1.0.99.2/xmlrpcs.inc");
function salary_compare($a, $b)
{
   global $salary_arr;
   $a=ereg_replace("-", "", $a);
   $b=ereg_replace("-", "", $b);
   if ($salary_arr[$a]==$salary[$b]) return 0;
   return ($salary_arr[$a] > $salary_arr[$b]) ? -1 :1;
}
$salary_sig=array(array($xmlrpcArray, $xmlrpcArray));
$salary_doc='Send this method an array of [string, int] structs';
function salary($m)
{
   global $salary_arr, $xmlrpcerruser, $s;
   xmlrpc_debugmsg("Entering 'salary'");
   // get the parameter
   $sno=$m->getParam(0);
   // error string for [if|when] things go wrong
   $err="";
   // create the output value
   $v=new xmlrpcval();
   $agar=array();
   if (isset($sno) && $sno->kindOf()=="array")
   {
      $max=$sno->arraysize();
```

```
        // print "<!-- found $max array elements -->\n";
        for($i=0; $i<$max; $i++)
        {
           $rec=$sno->arraymem($i);
           if ($rec->kindOf()!="struct")
           {
              $err="Found non-struct in array at element $i";
              break;
           }
           // extract name and salary from struct
           $n=$rec->structmem("name");
           $a=$rec->structmem("sal");
           // $n and $a are xmlrpcvals,
           // so get the scalarval from them
           $agar[$n->scalarval()]=$a->scalarval();
         }
         $salary_arr=$agar;
         uksort($salary_arr, salary_compare);
         $outAr=array();
         while (list( $key, $val ) = each( $salary_arr ) )
         {
             // recreate each struct element
             $outAr[]=new xmlrpcval(array("name" =>  new xmlrpcval($key), "salary" => @ new
             xmlrpcval($val, "int")), "struct");
          }
          // add this array to the output value
          $v->addArray($outAr);
    }
   else
    {
      $err="Must be one parameter, an array of structs";
    }
   if ($err)
    {
      return new xmlrpcresp(0, $xmlrpcerruser, $err);
    }
   else
    {
      return new xmlrpcresp($v);
    }
}
$res=new xmlrpc_server(array("examples.salary"=>array("function"=>"salary",
"signature"=>$salary_sig, "docstring"=>salary_doc)));
?>
```

The above listing shows that the PHP server script services the client request to sort employee names based on their salaries. The client calls the examples.salary function, and PHP maps the called function to the salary() method on the server. The salary() method calls the salary_compare() method to compare the salaries of the employees.

XML-RPC messages support various data types. You can create a client request that passes floating-point data as the method parameter.

Listing 6-9 shows the PHP server script that adds three double data type values:

Listing 6-9: PHP Server Script to Add Double Values

```
<?php
include("/xmlrpc-1.0.99.2/xmlrpc.inc");
include("/xmlrpc-1.0.99.2/xmlrpcs.inc");
$add_sig=array(array($xmlrpcDouble, $xmlrpcDouble, $xmlrpcDouble, $xmlrpcDouble));
$add_doc='Add three double values and return a value with type double.';
function add($m)
{
   $s=$m->getParam(0);
   $t=$m->getParam(1);
   $r=$m->getParam(2);
   return new xmlrpcresp(new xmlrpcval($s->scalarval()+$t->scalarval()+$r->scalarval(),
   "double"));
}
$resp=new xmlrpc_server(array("addDouble"=>array("function"=>"add",
"signature"=>$add_sig, "docstring"=>$add_doc)));
?>
```

The above listing shows that PHP maps the addDouble() function to the add() function in the PHP script. The signature of the add function specifies that the function accepts three double values as input and returns a single double type value.

Processing the Server's Response

The XML-RPC server returns a response when it successfully completes a transaction with the client object. The response that the server returns denotes an instance of the xmlrpcresp class. If the XML-RPC server does not complete a successful transaction with the client object, it returns a fault condition. As a result, the instance of the xmlrpcresp class is either a normal response or a fault condition. The XML-RPC response is a fault condition either if the server does not understand the client request or if there is any low-level error.

The xmlrpcresp class provides the following two methods to check if the server response is a fault condition:

- `faultCode() method`: Returns 0, if there is no fault condition. If a fault occurs, it returns the ID number that the server assigns to the fault.
- `faultString() method`s: Returns the description of the fault that occurs.

If the server successfully executes the client request, the instance of the xmlrpcresp class returns the data from the called procedure. The value() method accesses data that an instance of the xmlrpcresp class returns. The syntax of the value() method is:

```
$xmlrpcVal=$resp->value();
```

In the above syntax, $xmlrpcVal is an instance of the xmlrpcval class that contains the value that the server returns. The client process does not use the value that the value() method returns, if the faultCode is nonzero.

If the client request to multiply two integer values executes successfully, the server response returns the required result.

Figure 6-6 shows the server response to the client request for multiplying two integers, 13 and 2:

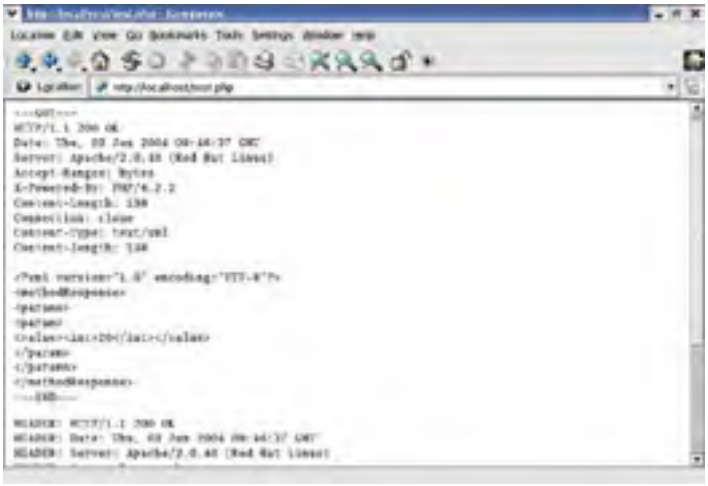


Figure 6-6: The Server Response of Multiplying Two Integers

If the client cannot successfully communicate with the server, a fault condition occurs in the server response.

Figure 6-7 shows the server response to an unsuccessful client request to multiply two integers:

Figure 6-7: Unsuccessful Server Response of Multiplying Two Integers

Similarly, you can obtain the server response for the client request to sort employee names on salary basis.

Figure 6-8 shows the server response with employee names sorted on the basis of their salaries:

Figure 6-8: The Server Response to Sort Employees

When an XML-RPC client makes a request to the server, the server response can be in one of the following forms:

- Low-level I/O error
- Valid XML-RPC response
- XML-RPC error condition

PHP has certain fault codes that XML-RPC classes use to indicate the type of fault. If an XML-RPC error condition occurs, the XML-RPC classes display the corresponding fault code and its description.

Table 6-1 describes the various fault codes:

Table 6-1: Fault Codes in PHP

Fault code	Fault description	Cause of error
1	Unknown method	Occurs when the client requests for a method that is not defined in the server.
2	Invalid return payload	Occurs when the client receives an invalid XML-RPC response from the server.
3	Incorrect parameters passed to method	Occurs when the client passes either incorrect number of parameters or incorrect data type of the parameter to the remote function.
4	Didn't receive 200 OK from the server	Occurs when there is an HTTP transport error.

XML-RPC provides interoperability between different platforms and languages. You can implement XML-RPC in PHP to write client and server scripts. You use the extensions of the XML-RPC protocol to authenticate the end users using the server scripts and provide an asynchronous transmission system.

Chapter 7: PHP and Web Distributed Data Exchange

Download CD Content

Web Distributed Data eXchange (WDDX) is an eXtensible Markup Language (XML)-based technology that helps exchange of data between Web programming languages, such as Hypertext Pre-Processor (PHP), Java, Active Server Pages (ASP), and Perl. WDDX provides a standard format for creating XML-based data structures, which can be easily transmitted across the Internet.

The WDDX module in PHP contains certain functions that convert PHP data to WDDX packets and WDDX packets to PHP data. WDDX functions use WDDX data structures to support information transfer between Web applications. These data structures are platform-independent.

The WDDX Application Programming Interface (API) in PHP supports encoding and decoding functions. Encoding functions convert PHP data to WDDX packets. Decoding functions convert WDDX packets to PHP data.

This chapter describes how to encode and decode data using the WDDX module in PHP. It also explains the data types that WDDX supports.

Introducing WDDX

Using WDDX, you can represent data corresponding to a specific programming language in a language independent format using a process called serialization. Serialization is a process that converts language-specific data structure into a WDDX packet. Deserialization converts a WDDX packet to language-specific data structure. The application that accepts data in the form of WDDX packets uses the deserialization process to retrieve data in its original form.

Understanding WDDX

WDDX API for programming languages provides functions that automatically encode data structures specific to a language into WDDX packets. The WDDX API also provides functions to decode the WDDX packets into language-specific data structures. WDDX packets store data in the XML format. A WDDX packet is created when a Web application converts language-specific data structure into WDDX.

The two important applications of WDDX are server-to-browser data exchange and server-to-server data exchange. In the server-to-browser data exchange, you can retrieve data from an application server, convert data into the WDDX packets using the WDDX API, and transfer the data to the browser in the form of an HyperText Transfer protocol (HTTP) response. In the server-to-server data exchange, the two servers that perform data exchange should have the WDDX API, to map data in a WDDX packet to the corresponding language-specific data structure.

The WDDX API consists of two parts, the WDDX Document Type Definition (DTD) and a set of serialization/deserialization modules. WDDX DTD is a specification that defines WDDX structures. The WDDX serialization/deserialization modules handle translation of data structures specific to a language to platform-independent XML representation or WDDX packets. You can develop WDDX serialization/deserialization modules in programming languages, such as PHP, Perl, ASP, Java, Python, and JavaScript.

WDDX DTD validates a WDDX packet that consists of data stored in the XML format. In the XML format, you enclose the data within the opening and closing tags, similar to HTML. WDDX supports data types similar to the data types in Web programming languages, such as Cold Fusion Markup Language (CFML) and PHP. Data types that WDDX supports are:

- Numbers: Represent floating-point values. Numbers range from +/-1.7E to +/-308.
- Date-time values: Represent date and time values that are encoded according to the International Standardization Organization 8601 (ISO8601) standard. You need not prefix 0 for the single digit values of month, day, hour, minute, or second.
- Strings: Contain arbitrary number of characters within it. A string data type should not contain null values. You can encode a string value using double-byte characters.
- Binary: Represents strings of binary data. The binary data is encoded in Multipurpose Internet Mail Extensions (MIME) base64 format.
- Array: Represents a collection of objects. It is an integer index data type.
- Structure: Represents a collection of objects of arbitrary data type that are indexed using strings.
- Recordset: Represents encapsulated data in a tabular form. A recordset is a collection of rows, in which each row consists of a set of named fields or columns. You can store only simple data types in a recordset. For storing complex data types in the tabular form, you should use an array of structures. The field names in a table should be in the [_.0-9A-Za-z]+ form, where the character, ., represents a period.

Note WDDX supports the null data type. You cannot associate null values with a number or a string.

Installing WDDX

You need to install the expat library available with Apache 1.3.7 or its higher version to use WDDX. The expat library is a C code library that implements a Simple Application Programming Interface for XML (SAX) parser. To install the expat library:

1. Download the expat package from the following URL:

   http://sourceforge.net/projects/expat/.

2. Unzip the package using the following command:
   ```
   tar -xvfz expat-1.95.5.tar.gz
   ```
3. Type the following command to change your working directory to expat-1.95.5:
   ```
   pushd expat-1.95.5
   ```

After you install the expat library, recompile PHP by adding -enable-wddx option to the configuration script.

Note The windows version of PHP has built-in support for the WDDX functions.

To check the installation of WDDX, run any script that uses WDDX functions provided in PHP. The following code uses a WDDX function:

```
<?php
print wddx_serialize_value("WDDX example");
?>
```

The above code produces a WDDX packet that contains the string, WDDX example, in the header tags. This WDDX packet ensures that the WDDX functions are already installed on your computer. If the code does not produce any output, install WDDX on your computer.

Exchanging Data using WDDX

WDDX is based on XML 1.0 version, and uses protocols, such as HTTP, to transfer data between Web applications. You can encode and decode data in the form of WDDX packets using serialization/deserialization functions.

Figure 7-1 shows the process of encoding data to the WDDX format and decoding data from the WDDX format:

Figure 7-1: Data Exchange Using WDDX Packet

The serialization process takes a part of data and converts it into a WDDX packet. Data in the WDDX packet is either a simple data type or complex data type. The simple data types include string, numeric, and Boolean data. The complex data types are arrays, structures, and recordsets. The deserialization process converts data in a WDDX packet to its original data type.

The syntax for the structure that holds data contained in a WDDX packet is:

```
<wddxPacket version='1.0'>
<header></header>
<data>
<data type>n</data type>
</data>
</wddxPacket>
```

In the above syntax, the <data /> tag pair denotes the start and end of serialized data. The <header /> tag contains a comment. The value n enclosed in the <data type /> tag represents a value that corresponds to the specified data type.

You need to enclose the WDDX packet in the <wddxPacket /> tag pair. The <wddxPacket> tag need to include the current version of WDDX. The data types that you can serialize into a WDDX format are: <number>, <Boolean>, <array>, <struct>, and <binary>. The following code shows the WDDX packet that contains a serialized string value:

```
<wddxPacket version='1.0'>
<header></header>
<data>
<string>This is a WDDX packet</string>
</data>
</wddxPacket>
```

The above WDDX packet contains a string value, and the application stores the value inside the tags as a string. If an application needs to deserialize the WDDX packet, it checks the <data type> tag pair to uncover the type of data stored in the packet.

Note In a WDDX packet, the application ignores the space between any pair of tags.

Using WDDX with PHP

PHP consists of WDDX modules that contain serialization and deserialization functions to convert PHP variables into WDDX compatible data structures, and then reconvert them into the PHP format. PHP generates WDDX packets as a single string. For example, you can encode a PHP associative array in WDDX and transfer the array to a Perl script. The Perl script decodes the data in the WDDX packet and uses the data for further processing.

The WDDX API of PHP provides five functions to encode PHP to WDDX packet. These encoding functions are wddx_serialize_value(), wddx_serialize_vars(), wddx_add_vars(), wddx_packet_start(), and wddx_packet_end(). The WDDX API for PHP also provides a function to decode data from a WDDX packet.

Encoding Data using WDDX

The WDDX technology enables data exchange between the Web applications that are created in different programming languages and run on different platforms. The PHP functions that encode data into a WDDX packet are:

- `wddx_serialize_value()`: Encodes a single variable into a WDDX packet.
- `wddx_serialize_vars()`: Encodes more than one variable into a WDDX packet.
- `wddx_add_vars()`: Adds PHP variables to a WDDX packet.
- `wddx_packet_start()`: Creates a new WDDX packet.
- `wddx_packet_end()`: Closes a WDDX packet.

Using the wddx_serialize_value() Function

The wddx_serialize_value() function creates a WDDX packet that contains a single value. The syntax of the wddx_serialize_value() function is:

```
string wddx_serialize_value (mixed var [, string comment])
```

In the above syntax:

- var denotes the name of the variable that you need to add to a WDDX packet.
- The wddx_serialize_value() function either returns a string that contains a WDDX packet, or the value, FALSE, if an error occurs. The wddx_serialize_value() function accepts variables as its first argument and converts the variables to a WDDX packet. The second argument of the wddx_serialize_value() function is optional and adds a comment to a WDDX packet.

Listing 7-1 shows how to use the wddx_serialize_value() function:

Listing 7-1: Serializing a Variable Using the wddx_serialize_value() Function

```
<?php
$country="Australia";
$va=wddx_serialize_value($country);
$fp=fopen("/var/www/html/example1.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to encode the $country variable to a WDDX packet using the wddx_serialize_value() function. The Web browser generates a parsed output and does not display the WDDX tags that are generated during the serialization process. The fwrite() function in the above listing writes the contents of the WDDX packet to the example.xml file.

Run the XML file to display the contents of the WDDX packet.

Figure 7-2 shows how the wddx_serialize_value() function generates a WDDX packet:

Figure 7-2: Using the wddx_serialize_value() Function to Generate a WDDX Packet

You can use the second parameter of the wddx_serialize_value() function to add a comment to the header of the WDDX packet, as shown in Listing 7-2:

Listing 7-2: Adding a Comment to a the WDDX Packet

```
<?php
$country="Australia";
$va=wddx_serialize_value($country, "An example of WDDX");
$fp=fopen("/var/www/html/example2.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows the WDDX packet that contains the comment, An example of WDDX.

Figure 7-3 shows a WDDX packet that contains a comment:

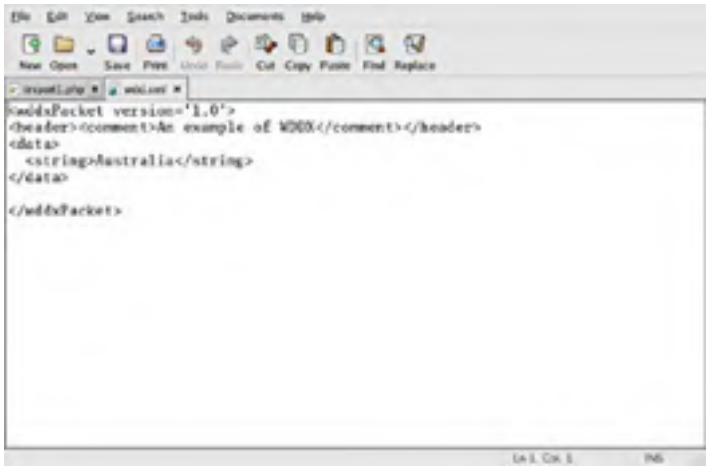


Figure 7-3: WDDX Packet Containing a Comment

You can serialize an array of values using the wddx_serialize_value() function, as shown in Listing 7-3:

Listing 7-3: Serializing an Array Using the wddx_serialize_value() Function

```
<?php
$countries=array("Australia", "India", "Kenya");
$va=wddx_serialize_value($countries);
$fp=fopen("/var/www/html/example3.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to serialize three variables, India, Kenya, and Australia, using the wddx_serialize_value() function.

Figure 7-4 shows the WDDX packet representing an array of values:

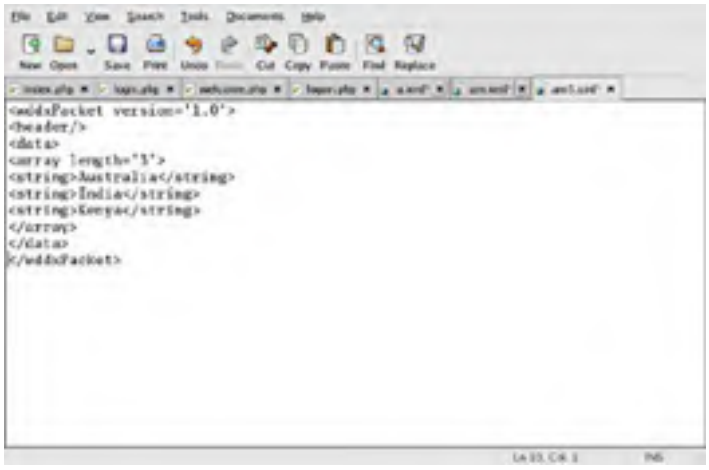


Figure 7-4: WDDX Packet Representing an Array of Values

Using the wddx_serialize_vars() Function

The wddx_serialize_vars() function creates a WDDX packet with a serialized representation of the passed variables. The syntax of the wddx_serialize_vars() function is:

```
string wddx_serialize_vars (mixed var [, mixed..])
```

The wddx_serialize_vars() function accepts one or more variables as its arguments, which can either be strings or arrays containing strings.

Listing 7-4 shows how to serialize a single variable using the wddx_serialize_vars() function:

Listing 7-4: Serializing a Variable Using the wddx_serialize_vars() Function

```
<?php
$country="Australia";
$va=wddx_serialize_vars("country");
$fp=fopen("/var/www/html/example4.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing uses the wddx_serialize_vars() function to serialize a single variable into a WDDX packet.

Figure 7-5 shows the WDDX packet that serializes a variable using the wddx_serialize_vars() function:

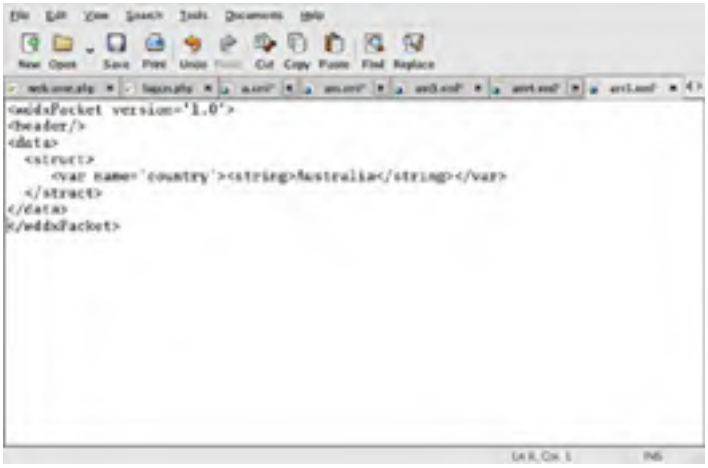


Figure 7-5: WDDX Packet Representing a Single Encoded Variable

Note The serialization of a single variable using the wddx_serialize_vars() function generates a WDDX packet, which is different from the WDDX packet that the wddx_serialize_value() function generates.

Listing 7-5 shows how to use the wddx_serialize_vars() function:

Listing 7-5: Serializing Multiple Values Using the wddx_serialize_vars() Function

```
<?php
$x="This is an example of the wddx_serialize_vars function.";
$y=array("emp_name"=>"Kenny", "emp_id"=>"E01", "emp_dept"=>"Sales");
$va=wddx_serialize_vars("x", "y");
$fp=fopen("/var/www/html/example5.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to serialize two variables, x and y, using the wddx_serialize_vars() function.

Figure 7-6 shows the WDDX packet that serializes the two variables, x and y:

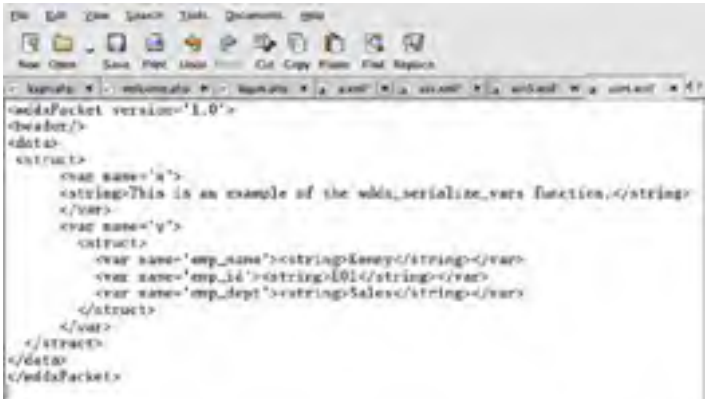


Figure 7-6: WDDX Packet that Serializes Two Variables

You can serialize more than two variables using the wddx_serialize_vars() function, with each variable containing an array of values, as shown in Listing 7-6:

Listing 7-6: Using the wddx_serialize_vars() Function

```
<?php
$student_names=array("Peter", "Crisp", "Ron");
$colleges=array("AEC", "BMBS");
$branches=array("IT", "EC", "EE");
$va=wddx_serialize_vars("student_names", "colleges", "branches");
$fp=fopen("/var/www/html/example6.xml", "w+");
fwrite($fp, $va, strlen($va));
?>
```

The above listing shows how to use the wddx_serialize_vars() function to serialize three variables, $student_names, $branches, and $colleges, each containing an array of values.

Figure 7-7 shows the WDDX packet that serializes three variables: $student_names, $branches, and $colleges:

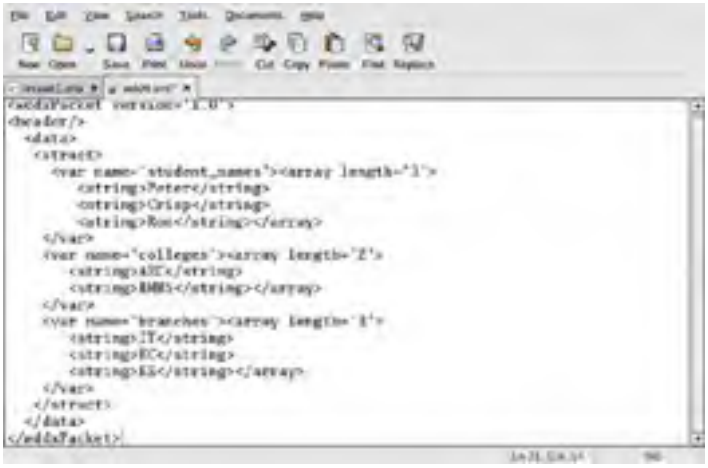


Figure 7-7: WDDX Packet that Uses the wddx_serialize_vars() Function

Using the wddx_add_vars() Function

The wddx_add_vars() function adds one or more variables to a WDDX packet, which is represented by a packet ID. The syntax for the wddx_add_vars() function is:

```
boolean wddx_add_vars(int packet_id, mixed var1 [,mixed var2,...])
```

The wddx_add_vars() function returns either TRUE or FALSE. You should use two additional functions to generate a WDDX packet that uses the wddx_add_vars() function, which are wddx_packet_start() and wddx_packet_end().

The wddx_packet_start() function creates a new WDDX packet with the WDDX structure inside the packet. The syntax of the wddx_packet_start() function is:

```
int wddx_packet_start([string comment])
```

In the above syntax, the int keyword indicates that the return type of the wddx_packet_start() function is an integer.

The wddx_packet_start() function accepts an optional comment string as an argument and returns a packet ID. This function automatically creates a structure definition for including variables inside a WDDX packet.

The wddx_packet_end() function closes a WDDX packet specified by the packet ID and returns a string that contains the WDDX packet. The syntax of the wddx_packet_end() function is:

```
string wddx_packet_end(int packet_id)
```

In the above syntax, the string keyword indicates that the return type of the wddx_packet_end() function is a string. The argument that you pass to the wddx_packet_end() function identifies the WDDX packet that your application should decode.

To create a WDDX packet and add variables to the packet using the wddx_add_vars() function:

1. Create an empty WDDX packet that can hold the data. You can create a WDDX packet using the wddx_packet_start() function.
2. Add data to the WDDX packet. You use the wddx_add_vars() function to add each variable to the WDDX packet.
3. Close the WDDX packet using the wddx_packet_end() function.

The first argument in the wddx_add_vars() function specifies the WDDX packet ID to which you need to add data.

Listing 7-7 shows how to use the wddx_add_vars() function to create a WDDX packet:

Listing 7-7: Creating a WDDX Packet Using the wddx_add_vars() Function

```
<?php
$country="Australia";
$id=wddx_packet_start("PHP");
wddx_add_vars($id, "country");
// contents of the packet are transferred to example7.xml file.
$packet=wddx_packet_end($id);
$fp=fopen("/var/www/html/example7.xml", "w+");
fwrite($fp, $packet, strlen($packet));
?>
```

The above listing shows how to create a WDDX packet using the wddx_packet_start() function. You can add variables to a WDDX packet using the wddx_add_vars() function.

Figure 7-8 shows the variables that you add to a WDDX packet using the wddx_add_vars() function:

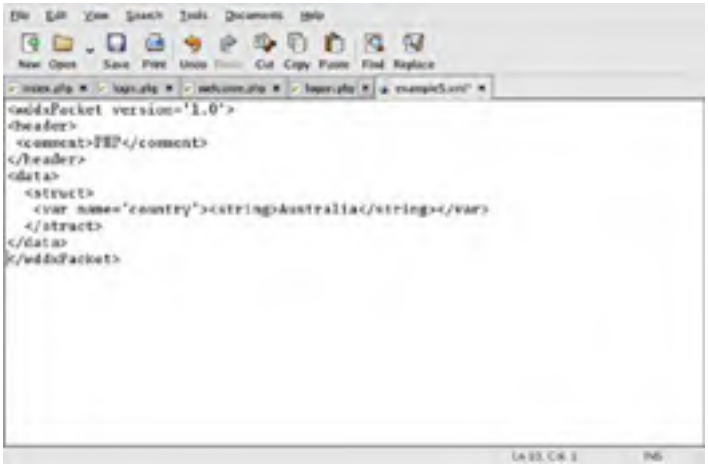


Figure 7-8: Adding Variables to a WDDX Packet Using the wddx_add_vars() Function

You can add more than one variable using the wddx_add_vars() function, as shown in Listing 7-8:

Listing 7-8: Using the wddx_add_vars() Function to Add More than One Variable

```
<?php
$name="Ronald";
$department="Administration";
$Age="35";
$id=wddx_packet_start("PHP example");
wddx_add_vars($id, "name", "department", "Age");
$packet=wddx_packet_end($id);
$fp=fopen("/var/www/html/example8.xml", "w+");
fwrite($fp, $packet, strlen($packet));
?>
```

The above listing shows a WDDX packet that serializes three variables, name, department, and age.

The wddx_serialize_vars() and wddx_add_vars() functions use the <struct> element to store variables and their values. The wddx_serialize_value() function uses the <struct> element to store variables that represent associative arrays.

Figure 7-9 shows the WDDX packet that serializes three variables:

Figure 7-9: Adding Multiple Variables Using the wddx_add_vars() Function

Note The wddx_serialize_value() and wddx_serialize_vars() functions automatically create a WDDX packet, but you should provide a starting and a closing header for the wddx_add_vars() function.

Decoding Data using WDDX

Decoding involves parsing a WDDX packet into an appropriate PHP variable. PHP contains a single function, wddx_deserialize(), to retrieve data from a WDDX packet. The wddx_deserialize() function decodes a WDDX packet. The syntax of the wddx_deserialize() function is:

```
mixed wddx_deserialize(string packet);
```

The above syntax shows that the wddx_deserialize() function accepts a single string argument that represents a WDDX packet. The wddx_deserialize() function returns a number, a string, or an array.

Listing 7-9 shows the deserialization of a PHP variable from a WDDX packet:

Listing 7-9: Deserializing a Single PHP Variable

```
<?php
$country="Australia";
//Serialize the variable using the wddx_serialize_value() function.
$id=wddx_serialize_value($country);
//Write the contents of the WDDX packet into an XML file.
$fp=fopen("/var/www/html/example9.xml", "w+");
fwrite($fp, $id, strlen($id));
//Deserialize the WDDX packet.
$output=wddx_deserialize($id);
//Display the decoded value of the PHP variable.
print($output);
?>
```

The above listing shows how to retrieve data from a WDDX packet using the wddx_deserialize() function.

Figure 7-10 shows deserialized data of the WDDX packet in Listing 7-9:

Figure 7-10: Output of the wddx_deserialize() Function

The wddx_deserialize() function decodes a variable containing an array of values, as shown in Listing 7-10:

Listing 7-10: Deserializing an Array of Values

```
<?php
$names=array("John", "Christine", "Joseph");
//Serializing an array of values.
$id=wddx_serialize_value($names);
//Deserializing the WDDX packet identified by $id.
$output=wddx_deserialize($id);
print_r($output);
?>
```

The above listing shows how to deserialize a WDDX packet that contains an array of values.

Figure 7-11 shows the output of Listing 7-10:

Figure 7-11: Output of a WDDX Packet

◄ PREVIOUS NEXT ►

Chapter 8: Working with Databases using PHP and XML

Download CD Content

You can store data in databases or as eXtensible Markup Language (XML) documents. Hypertext Preprocessor (PHP) converts data stored in the databases into XML documents and vice versa. Applications on the Internet can retrieve and update the data stored in the form of XML documents.

This chapter explains how to store XML documents in a database using PHP. This chapter describes how to export data stored in databases into an XML document and how to import data from an XML document into a database using the Simple API for XML (SAX) approach. This chapter also describes how to convert an XML document into an HTML format.

Storing XML Documents

XML is a standard language to store information on the Internet in a structured manner. XML is a subset of Standard Generalized Markup Language (SGML) and consists of tags to represent data.

An XML document stores data in a tree structure. You can query the contents of the XML document using a semi-structured language, such as XSL Transformations (XSLT). You can use XML documents to create Web applications and exchange information on the Internet.

You can store XML documents in a database and retrieve them. You can store the XML documents in a database by mapping the tree structure of the XML document to the database schemas.

When you map the tree structure of the XML document to the database schema, you design the database schemas according to the Document Type Declaration (DTD) of the XML structure. DTD defines the rules that an XML document should follow. The advantages of using database schemas to store XML documents are:

- You can create XML documents using relational tables, without depending on the DTD.
- You can perform various query operations on the XML documents using index structures. Indexing is of two types, position-based and path-based. Position-based indexing manipulates the elements and attributes to perform a query operations based on their position within the document. Path-based indexing performs the query operation using the path of the elements and attributes in the tree structure.

The disadvantages of using database schemas to store XML documents are:

- The database schemas cannot be designed efficiently in accordance with DTD. Relational databases and object-oriented databases cannot represent DTD to store XML documents efficiently.
- Maintaining the consistency of the database schemas is not possible. You need to change the structure of the database schemas every time there is a change in the structure of the XML document.

Storing XML Documents in Databases

An XML document is represented in a tree structure, which contains various nodes, such as element node, attribute node, and text node. The element node contains a text label, and can contain child nodes, such as the attribute node and the text node. The attribute node contains the name and value of the attribute. An attribute node does not have any child nodes. The text node in an XML document contains character data and represents the value for the element.

The process of storing the XML documents involves decomposing the XML tree structure into relations, which the relational tables can reuse and access.

The database relations that you can use for storing the XML document include element, attribute, text, and path. A relation stores data corresponding to the node of the tree structure. For example, the attribute relation stores the data stored in the attribute node. The various relations and their database attributes are:

- Element Relation: Stores data about the element node using database attributes, such as docID, pathID, index, and pos. The docID represents the document identifier, and pathID represents the path identifier. The index attribute represents the order among the sibling having the same path identifier. The pos attribute specifies the position of the element node with respect to the other nodes.
- Attribute Relation: Stores data about the attribute nodes. The database attributes in the attribute relation are docID, pathID, Attvalue, and pos. The Attvalue attribute specifies the value of an attribute.
- Text Relation: Stores data about the text nodes. The database attributes in the text relation are docID, pathID, value, and pos.
- Path Relation: Stores data about simple paths. The database attributes in the path relation are docID, pathID, Attvalue, and pos.

You can use query languages such as XML Query Language (XQL) to perform data manipulation, such as joins, on the XML data.

Organizing XML Documents with Entities

An XML document consists of storage units called entities, which are a collection of text and markup tags. In an XML document, entities are used to store data. The advantages of organizing an XML document with entities are:

- Ensures consistency in representing repeating text.

- Ensures that every entity instance is exactly the same.
- Limits repetitive typing of same information.

PREVIOUS NEXT

Exporting Database Records to Create XML Documents

Using PHP, you can export and convert the data stored in the database into an XML document. You can also manipulate the data stored in databases using various tools, such as XPath and XSLT.

To export the data from a database in an XML document:

1. Establish a connection with the database.
2. Retrieve data from the database by executing SQL queries.
3. Create an XML document from the retrieved information in the memory of the system.

You can manipulate and parse the generated XML document using Document Object Model (DOM).

Connecting to a Database to Export Data

The first step in exporting data from a database into an XML document is to establish a connection with the MySQL database. PHP provides a specific set of APIs that perform various query operations on the MySQL database. The PHP functions that you use to work with MySQL database server are:

- `mysql_connect`: Connects to the MySQL database. The syntax of the mysql_connect() function is:

```
resource mysql_connect ( [string Host_Name [:port][:/path/to/socket]], string username
[, string password ]]])
```

In the above syntax, you need to pass the hostname, username, and password parameters as arguments to the mysql_connect() function. You can also specify optional parameters, such as the port and path of the MySQL socket.

- `mysql_select_db`: Selects a MySQL database from the specified link identifier. The syntax of the mysql_select_db() function is:

```
bool mysql_select_db(string database, resource[link indentifier])
```

In the above syntax, the mysql_select_db() function returns the value, True, on establishing a connection with the database; and returns the value, False, when a connection is not established with the database.

- `mysql_query`: Performs query operation on the currently selected database. The syntax of the mysql_query() function is :

```
resource mysql_query ( string query [, resource link identifier])
```

In the above syntax, if no resource link identifier is specified, the query is generated from the currently opened linked.

- `mysql_fetch_object`: Returns the result set of the query as an object. The syntax of the mysql_fetch_object() function is:

```
object mysql_fetch_object(resource result, int [result _type])
```

- `mysql_close`: Closes the connection to the database. The mysql_close() function returns a Boolean value. The syntax of the mysql_close() function is:

```
bool mysql_close(resource[link identifier])
```

- `mysql_num_rows`: Returns the number of rows in a result set. The syntax of the mysql_num_rows() function is:

```
int mysql_num_rows(resource result)
```

- `mysql_error`: Returns the numerical value of the error message generated after running the MySQL statement. The syntax of the mySQL_error() function is:

```
string msql_error(resource [link identifier ])
```

- `mysql_db_query`: Selects a database and runs a query on the database. The syntax of the mysql_db_query() function is:

```
resource mysql_db_query(string database, string query, resource[link identifier])
```

Listing 8-1 shows how to establish a connection with the database using the database access functions provided by PHP:

Listing 8-1: Creating a Connection to the Database

```
$connect = mysql_connect("localhost", "root", "root123") or die ("could not connect");
print("Successful Connection");
mysql_select_db("information") or die ("Unable to select database!");
$query = "SELECT * FROM employee";
$result = mysql_query($query)
or die ("query failed, error code = : "  . mysql_errno());
```

In the above listing:

- The mysql_connect() function creates a connection to the MySQL database and returns value, True, when the connection is established.
- You need to provide the hostname, username, and the password as arguments to the mysql_connect() function.

After connecting to a database, you can perform various operations such as create a table, and insert, update, and delete rows from the table.

Listing 8-2 shows how to create a table and insert rows in the MySQL database, information:

Listing 8-2: Creating the Employee table in MySQL

```
mysql> USE information -A;
mysql> CREATE TABLE employee
-> (
-> EmployeeName Varchar(50) NOT NULL,
-> EmpID Varchar(5) NOT NULL,
-> Salary Integer
-> );
```

The above listing creates the Employee table. After creating the Employee table, you can insert rows in the Employee table.

Listing 8-3 shows how to insert rows in the Employee table:

Listing 8-3: Inserting Rows in the Employee Table

```
mysql> INSERT INTO employee
VALUES("Joseph", "1005", 20000);
mysql> INSERT into employee
-> VALUES("John", "1007",25000);
mysql> INSERT into employee
VALUES("Harry", "1008", 35000);
```

The above listing inserts three rows in the Employee table. You can retrieve data from the database using the SELECT command.

Figure 8-1 shows the output of the SELECT command:

Figure 8-1: Contents of the Employee Table

You can export the content of the Employee table into an XML tree using the DOM parser. To do this, you need to first establish a connection with the database, and then retrieve data using the MySQL functions provided by PHP.

Creating an XML Document

After establishing the connection with a database, you need to create an XML document from the database. You can create an XML document from the data retrieved after querying the database, after implementing the DOM functions, such as new_xmldoc().

Listing 8-4 shows how to create an XML document from the data retrieved from the MySQL database:

Listing 8-4: Creating the XML Document

```
<?php
$connection = mysql_connect("localhost", "root", "root123") or die ("Unable to connect!");
mysql_select_db("information") or die ("Unable to select database!");
$query = "SELECT * FROM employee";
$result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
if (mysql_num_rows($result) > 0)
{
   // Create the DomDocument object.
   $doc = new_xmldoc("1.0");
   // Add root node.
   $root = $doc->add_root("EmployeeInfo");
   // Iterate through result set.
   while(list($EmployeeName, $EmpID, $Salary) = mysql_fetch_row($result))
   {
      // Create item node.
      $rec = $root->new_child("employee", "");
      $rec->set_attribute("EmpID", $EmpID);
      $rec->new_child("EmployeeName", $EmployeeName);
      $rec->new_child("Salary", $Salary);
```

```
    }
    // Print the tree.
    echo $doc->dumpmem();
}
?>
```

In the above listing:

- An XML document is created from the data obtained after querying the Employee table stored in the information database.
- The new_xmldoc() function creates a new XML document.
- The new_child() function allows you to add child nodes, such as EmpName and Salary, to the EmployeeInfo root node.
- The dumpmem() function converts the generated XML document into a string that can be printed using the echo command.

Figure 8-2 shows the output of Listing 8-4:

Figure 8-2: Contents of XML File

You can also export data from multiple tables simultaneously. In PHP, you can create an XML document from the data retrieved after performing the query operation on multiple tables. For example, you want to display the Employee designation and the department along with other personnel information stored in the Employee table. The Employee information, such as designation and department, are stored in the Employee_Info table in the information database.

Figure 8-3 shows the contents of the Employee_Info table created in the information database:

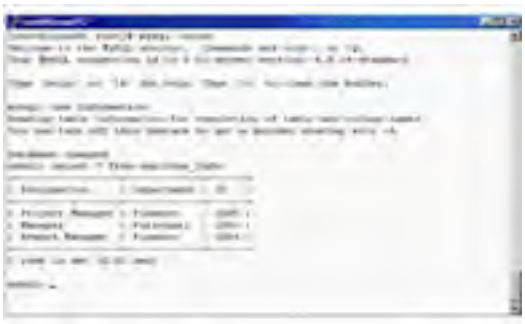


Figure 8-3: Contents of Employee_Info Table

Listing 8-5 shows how to export the data retrieved from the database after querying the Employee table and Employee_Info table:

Listing 8-5: Exporting Data Retrieved from Multiple Tables

```
<?php
$connection = mysql_connect("localhost","root","root123") or die ("Unable to connect!");
mysql_select_db("information") or die ("Unable to select database!");
$query = "SELECT * FROM employee";
$result = mysql_query($query) or die ("Error in query: $query. " .
mysql_error());
if (mysql_num_rows($result) > 0)
{
   // Create the DomDocument object.
    $doc = new_xmldoc("1.0");
   // add root node
   root = $doc->add_root("EmployeeInfo");
   // Iterate through result set.
   while(list($EmployeeName, $EmpID, $Salary) = mysql_fetch_row($result))
   {
      // Create item node.
      $rec = $root->new_child("employee","");
      $rec->set_attribute("EmpID", $EmpID);
```

```
        $rec->new_child("EmployeeName", $EmployeeName);
        $rec->new_child("Salary", $Salary);
        // Add node from table Employee_Info.
         $query2 = "SELECT Designation,Department FROM Employee_Info where ID = '$EmpID'";
         $result2 = mysql_query($query2) or die ("Error in query: $query2. " . mysql_error());
        // Print each track as a child node of <tracks>.
        while(list($Designation,$Department) = mysql_fetch_row($result2))
        {
           $rec->new_child("Designation","$Designation");
           $rec->new_child("Department","$Department");
        }
    }
    // Dump XML document to a string.
}
$varf = $doc->dumpmem();
echo $varf;
$fp = fopen("/var/www/html/out.xml", "w+");
fwrite($fp, $varf, strlen($varf));
?>
```

The above listing shows how to export data retrieved from the database after querying the Employee and Employee_Info tables. In the above listing:

- The new_xmldoc() function generates an XML document.
- Data from the Employee and Employee_Info tables are inserted in the xml file created.
- The nodes, such as designation, department, and employee are added to the root node after querying the Employee_Info table.
- The dumpmem() function converts the XML document into a string, $varf.
- The generated XML file is stored as a separate file using the fwrite() function.

Figure 8-4 shows the output of Listing 8-5:

Figure 8-4: Output of Exporting Data from Multiple Tables

Closing the Connection to a Database After Exporting Data

You need to close the MySQL connection after you have generated the result set from the MySQL database. The mysql_close() function lets you close the MySQL connection.

Listing 8-6 shows how to use the mysql_close() function to close database connection:

Listing 8-6: Closing Connection to MySQL Database

```
<?php
$connection = mysql_connect("localhost", "root", "root123")
or exit ("Could not connect");
print ("Connected successfully");
//Close connection.
mysql_close($connection);
?>
```

In the above listing:

- The mysql_close() function closes the connection to the MySQL database after generating the result.
- The link identifier is specified as the argument to the mysql_close() function.
- The mysql_close() function closes the last active link to the MySQL server by default.

Formatting XML Document into HTML Format

In addition to exporting data into an XML document, you can transform the data from the database into HTML format. To format a generated XML document into HTML format using the SAX parser:

1. Generate the XML document after retrieving the records from the database.
2. Transform the generated XML document into HTML pages.

You can transform an XML document into HTML format using the SAX parser. SAX is an event-based approach that parses an XML document in chunks, and processes the tags as the parser encounters them in the document. You can export the content of the Employee table into an XML file and then convert the XML document into HTML format.

Listing 8-7 shows how to generate the XML document from the database and convert the XML document into HTML:

Listing 8-7: Generating XML and Converting into HTML

```
<?php
// Defining the database parameters.
$hostname = "localhost";
$username = "root";
$password = "root123";
$database = "information";
$table = "employee";
//  Exporting the database records and generating a XML document.
// Querying the database.
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($database) or die ("Unable to select database!");
$query = "SELECT * FROM $table";
$res = mysql_query($query) or die ("Error in query: $query. " . mysql_error());
if(mysql_num_rows($res) > 0)
{
   // Creating the DomDocument object.
   doc = new_xmldoc("1.0");
   // Adding the root node.
   $root = $doc->add_root("table");
   $root->set_attribute("name", $table);
   // Creating nodes.
   $struct = $root->new_child("struct", "");
   $data = $root->new_child("data", "");
   // Create elements for each field name, type and length.
   $fields = mysql_list_fields($database, $table, $connection);
   for ($t=0; $t<mysql_num_fields($fields); $t++)
   {
      $field = $struct->new_child("field", "");
      $name = mysql_field_name($fields, $t);
      $length = mysql_field_len($fields, $t);
      $type = mysql_field_type($fields, $t);
      $field->new_child("name", $name);
      $field->new_child("type", $type);
      $field->new_child("length", $length);
   }
   // Move on to getting the raw data (records).
   // Iterate through result set.
   while($row = mysql_fetch_row($res))
   {
      $record = $data->new_child("record", "");
      foreach ($row as $field)
      {
         $record->new_child("item", $field);
      }
   }
   // Dumping the XML tree as a string.
   $xml_string = $doc->dumpmem();
}
// Closing the database connection.
mysql_close($connection);
// Converting the XML document into an HTML page using SAX parser.
// Array to hold HTML markup for starting tags.
$startTags = array(
'TABLE' => '<html><head></head><body><table border="1"
cellspacing="0"cellpadding="5">',
'STRUCTURE' => '<tr>',
'FIELD' => '<td bgcolor="silver"><font face="Times Roman" size="-1">',
'RECORD' => '<tr>',
'ITEM' => '<td><font face="Arial" size="-1">',
'NAME' => '<b>',
'TYPE' => '  <i>(',
'LENGTH' => ', '
);
// Array to hold HTML markup for ending tags.
$endTags = array(
'TABLE' => '</body></html></table>',
'STRUCTURE' => '</tr>',
'FIELD' => '</font></td>',
'RECORD' => '</tr>',
'ITEM' => ' </font></td>',
'NAME' => '</b>',
'TYPE' => '',
```

```
'LENGTH' => ')</i>'
);
// Call this when a start tag is found.
function startElementHandler($parser, $name, $attributes)
{
   global $startTags;
   if($startTags [$name])
   {
      // Look up array for this tag and print corresponding markup.
      echo $startTags[$name];
   }
}
// Call this when an end tag is found.
function endElementHandler($parser, $name)
 {
   global $endTags;
   if($endTags [$name])
   {
      // Look up array for this tag and print corresponding markup.
      echo $endTags [$name];
   }
}
// Call this when character data is found.
function characterDataHandler($parser, $data)
{
   echo $data;
}
// Initialize the parser.
$xml_parser = xml_parser_create();
// Set callback functions.
xml_set_element_handler($xml_parser, "startElementHandler", "endElementHandler");
xml_set_character_data_handler($xml_parser, "characterDataHandler");
// Parse the XML document.
if (!xml_parse($xml_parser, $xml_string, 4096))
{
   $ec = xml_get_error_code($xml_parser);
   die("XML parser error (error code " . $ec . "): " . xml_error_string($ec) .
   "<br>Error occurred at line " . xml_get_current_line_number($xml_parser) . ", column "
   .
   xml_get_current_column_number($xml_parser) . ", byte offset " .
   xml_get_current_byte_index($xml_parser));
}
xml_parser_free($xml_parser);
?>
```

The above listing converts the content of the Employee table into an XML document using the DOM approach. The generated XML document is transformed into HTML format using the SAX parser.

You need to define various database parameters, such as hostname, username, and password, to establish a connection to the database. After establishing a connection with the database, you need to process queries to retrieve data from the database.

In the Listing 8-7:

- The XML tree is created using the new_xmldoc() function, which belongs to the DOMDocument class.
- The nodes are added to the XML tree after being retrieved from the result set.
- The dumpmem() function lets you dump the generated XML tree in a string. You can also store the XML tree in a file. For example, in Listing 8-7, you use the following code to store the XML document in the export.xml file:

```
$fp = fopen("/var/www/html/export.xml", "w+");
fwrite($fp, $doc->dumpmem(), strlen($doc->dumpmem()));
```

Figure 8.5 shows the contents of the export.xml file:

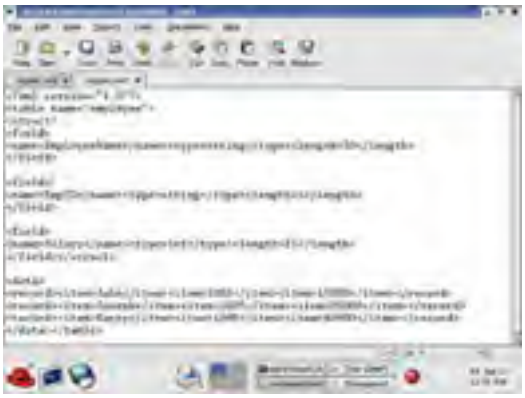


Figure 8-5: The export.xml File

After the XML document is generated using the DOM approach, you need to close the database connection using the mysql_close() function.

Figure 8-6 shows the output obtained after running Listing 8-7 from the Web browser:

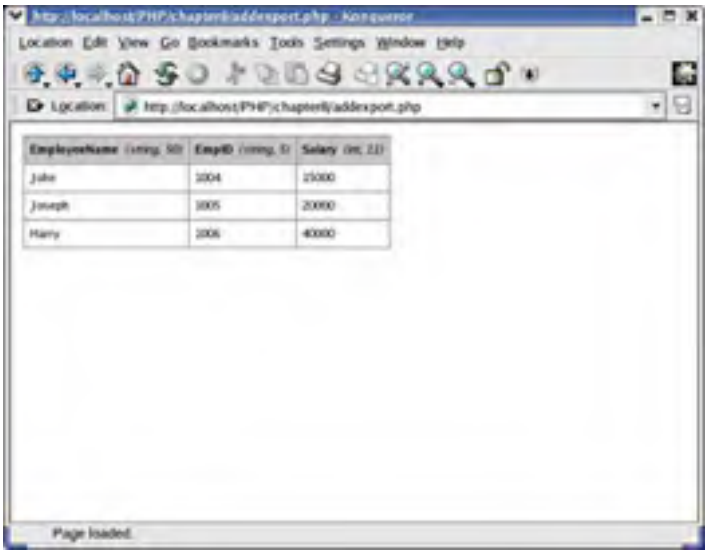


Figure 8-6: Generating the Contents of Employee Table in HTML

PREVIOUS NEXT

Importing XML Data to a Database

In addition to exporting database records to XML documents, PHP lets you import data from an XML document into a database. Using the DOM and SAX approach, you can construct SQL queries and insert data into a database from an XML document. To import XML data into a database:

1. Parse the XML data using the DOM or the SAX approach.
2. Create a list of field value pairs.
3. Insert the field value pairs in the database.
4. Run the database query to test whether data is inserted or not and close the database connection.

Connecting to a Database to Import Data

You need to establish a connection with the MySQL database to import data from an XML document to the database. The following code shows how to connect to the MySQL database:

```
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($db) or die ("Unable to select database!");
```

In the above code, a connection to the database is established using the mysql_connect() function. You need to specify the hostname, username, and password as arguments to the mysql_connect() function.

Parsing an XML Document

To import XML document into a database, you need to parse the XML document. You can parse the XML document using both the SAX and DOM approach. For example, to import the XML document, student.xml file, that stores the student information in a database, you need to parse the student.xml file.

Listing 8-8 shows the contents of the student.xml file:

Listing 8-8: Content of the student.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
<item>
<name>John</name>
<age>15</age>
<address>New York</address>
<standard>10</standard>
</item>
<item>
<name>Joseph</name>
<age>16</age>
<address>New York</address>
<standard>12</standard>
</item>
<item>
<name>Mary</name>
<age>14</age>
<address>New Jersey</address>
<standard>8</standard>
</item>
</list>
```

The above listing shows the content of the student.xml file that stores student information, such as name, age, address, and standard.

You need to create a table in the database that can store the data imported from the student.xml file.

Listing 8-9 shows how to create the student table in the information database:

Listing 8-9: Creating the Student Table

```
use information
mysql> CREATE TABLE student
-> (
-> name Varchar(30) NOT NULL,
-> age Integer NOT NULL,
-> address Varchar(30) NOT NULL,
-> standard Integer
-> );
```

The above listing creates the student table, in which you can insert data from the student.xml file.

Listing 8-10 shows how to import the student.xml file into a database, by parsing the student.xml file using the SAX approach:

Listing 8-10: Parsing the student.xml File using SAX

```
<?php
$cTag = "";
// Array to hold the values for the SQL statement.
$values = array();
$elements = array("name", "age", "address", "standard");
// XML file to parse
$xmlFile = "student.xml";
// Database parameters
$hostname = "localhost";
$username = "root";
&password = "root123";
$database = "information";
$table = "student";
// Processes on encountering a opening tag in the XML.
function startElementHandler($parser, $n1, $attributes)
{
   global $cTag;
   $cTag = $n1;
}
// Processes on encountering a closing tag in the XML.
function endElementHandler($parser, $n1)
{
   global $values, $cTag;
   // Import database link and table name.
   global $connection, $table;
   // if ending <item> tag
   // Implies end of record.
   if (strtolower($n1) == "item")
   {
      // Generating the query string.
      $query = "INSERT INTO student";
      $query .= "(name, age, address, standard) ";
      $query .= "VALUES(\"" . join("\", \"", $values) . "\");";
      // Processing the query.
      $result = mysql_query($query) or die ("Error in query: $query. " .
      mysql_error());
      // Reset all internal counters and arrays.
      $values = array();
      $cTag = "";
   }
}
// Processes on encountering a closing tag in the XML.
function characterDataHandler($parser, $data)
{
    global $cTag, $values, $elements;
   // lowercase tag name
   $cTag = strtolower($cTag);
   // Look for tag in $elements[] array.
   if (in_array($cTag, $elements) && trim($data) != "")
   {
      array
      $values[$cTag] = mysql_escape_string($data);
   }
}
// Initializing the SAX parser.
$xml_parser = xml_parser_create();
// Set callback functions.
xml_set_element_handler($xml_parser, "startElementHandler", "endElementHandler");
xml_set_character_data_handler($xml_parser, "characterDataHandler");
// Open connection to database.
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($database) or die ("Unable to select database!");
// read XML file
if (!($fp = fopen($xmlFile, "r")))
{
   die("File I/O error: $xmlFile");
}
// parse XML
while ($data = fread($fp, 4096))
{
   // error handler
   if (!xml_parse($xml_parser, $data, feof($fp)))
   {
      $error_code = xml_get_error_code($xml_parser);
      die("XML parser error (error code " . $error_code . "): " .
      xml_error_string($error_code) . "<br>Error occurred at line " .
      xml_get_current_line_number($xml_parser));
   }
}
?>
```

The above listing shows how to parse the student.xml file using the SAX approach. In the above listing:

- The instance of the SAX parser is initialized using the xml_parser_create() function, and is configured to call various functions, such as startElemenHandler() and endElementHandler().

- The startElemenHandler() function executes when a starting tag is processed in the XML document, and the endElementHandler() function executes when the closing tag in the XML document is processed.

You need to pass the tag name and the parser as arguments to the tag handler functions, such as startElemenHandler() and endElementHandler(). The characterDataHandler() function executes when the character data in the XML document is processed. You need to specify the Character Data (CDATA) text as arguments in the characterDataHandler() function.

The SAX parser retrieves and stores data from the student.xml file in the associative array, $values. The SAX parser retrieves data after finding character data having element name, such as name, age, address, and standard. The SAX parser creates a query string from the values obtained from the $values array.

You can also parse the XML document using the DOM approach.

Listing 8-11 shows how to parse the XML document using DOM:

Listing 8-11: Parsing XML Document using DOM

```
<?php
$xml_file = "student.xml";
$doc = xmldocfile($xml_file);
$name = array();
$age = array();
$address = array();
$standard = array();
$root = $doc->root();
$nodes = $root->children();
$a=0;
$b=0;
$c=0;
for($x=0;$x<sizeof($nodes);$x++)
{
   if ($nodes[$x]->type==XML_ELEMENT_NODE)
   {
      $text=$nodes[$x]->children();
      for($i=0;$i<sizeof($text);$i++)
      {
         $temp=$text[$i]->children();
         for($j=0;$j<sizeof($temp);$j++)
         {
            echo $temp[$j]->content;
         }
      }
   }
}
?>
```

The above listing shows how to parse the student.xml file using the DOM approach.

Closing the Connection to a Database After Importing Data

You need to close the database connection after importing the XML document in the database. The code to close the database connection is:

```
xml_parser_free($xml_parser);
mysql_close($connection);
```

The above code shows how to close the connection to the database using the mysql_close() function. The xml_parser_free() function frees the parser and returns true if $xml_parser is valid otherwise returns the value, False.

Alternative Method to Import XML Documents

Using PHP, you can import XML documents in a database without specifying the table name, field names, and values in the SQL statement. For example, you want to import the datastudent.xml file into a table, stud.

Listing 8-12 shows the contents of the datastudent.xml file:

Listing 8-12: Contents of datastudent.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<table name1="stud">
<record>
<age>12</age>
<address>New York</address>
<standard>12</standard>
<name>Mary</name>
</record>
<record>
<age>14</age>
<address>New York</address>
<standard>10</standard>
<name>John</name>
</record>
<record>
<age>15</age>
<address>New Jersey</address>
<standard>12</standard>
```

```
<name>Tom</name>
</record>
</table>
```

The above listing stores student information, such as name, record, address, age, and standard. The name of the table, in which the contents of the datastudent.xml file are imported, is specified as an attribute to the <table> element of the XML document. The student information, such as name, address, standard, and age, are specified with the <record> element.

Listing 8-13 shows how to import an XML document in a database without specifying the table and field names:

Listing 8-13: Importing the XML Document

```
<?php
// Initialize some variables
$cTag = "";
$fields = array();
$values = array();
// XML file to parse
$xml_file="datastudent.xml";
// Database parameters
// Get these via user input
$hostname = "localhost";
$username = "root";
$password = "";
$db = "information";
// Called when parser finds start tag.
function startElementHandler($parser, $name1, $attributes)
{
   global $cTag, $table;
   $cTag = $name1;
   // Get table name.
   if (strtolower($cTag) == "table")
   {
      foreach ($attributes as $v)
      {
         $table=$v;
      }
   }
}
// Called when parser finds end tag.
function endElementHandler($parser, $name1)
{
   global $fields, $values, $count, $cTag;
   // Import database link and table name.
   global $connection, $table;
    if (strtolower($name1) == "record")
    {
      // Generate the query string.
      $query = "INSERT INTO $table";
      $query .= "(" . join(", ", $fields) . ")";
      $query .= " VALUES(\"" . join("\", \"", $values) . "\");";
      // Execute query
      mysql_query($query) or die ("Error in query: $query. " .mysql_error());
      // Reset all internal counters and arrays.
      $fields = array();
      $values = array();
      $count = 0;
      $cTag = "";
    }
 }
// Called when parser finds cdata
function characterDataHandler($parser, $data)
{
   global $fields, $values, $cTag, $count;
   if (trim($data) != "")
   {
      // Add field-value pairs to $fields and $values array.
      // The index of each array is used to correlate the field-value pairs.
      $fields[$count] = $cTag;
      // Escape quotes with slashes
      $values[$count] = mysql_escape_string($data);
      $count++;
   }
}
// Initialize parser
$xml_parser = xml_parser_create();
// Set callback functions.
xml_set_element_handler($xml_parser, "startElementHandler", "endElementHandler");
xml_set_character_data_handler($xml_parser, "characterDataHandler");
// Open connection to database.
$connection = mysql_connect($hostname, $username, $password) or die ("Unable to connect!");
mysql_select_db($db) or die ("Unable to select database!");
// read XML file
if (!($fp = fopen($xml_file, "r")))
```

```
{
   die("File I/O error: $xml_file");
}
// Parse XML
while ($data = fread($fp, 4096))
{
   // Error handler
   if (!xml_parse($xml_parser, $data, feof($fp)))
   {
      $ec = xml_get_error_code($xml_parser);
      die("XML parser error (error code " . $ec . "): " . xml_error_string($ec) .
      "<br>Error occurred at line " . xml_get_current_line_number($xml_parser));
   }
}
// All done, clean up!
xml_parser_free($xml_parser);
mysql_close($connection);
?>
```

The above listing imports the datastudent.xml file without specifying the table name and field name values in the PHP script. You need to specify the hostname, username, and password of the MySQL database to connect to the database.

In the above listing, the xml_parser_create() function initializes the SAX parser, and the SAX parser processes various handler functions, such as xml_set_elemet_handler and xml_set_character_data_handler.

Figure 8-7 shows the output of Listing 8-13:

Figure 8-7: Contents of Table Imported from the XML Document

◄ PREVIOUS NEXT ►

Chapter 9: Creating an Online Shopping Cart Application

Download CD Content

The online shopping cart application allows an end user to search for a specific book in a database, place an order for the book, and purchase the book online. This application contains two sections, Admin and Client. The Admin section lets you administer the application by managing information related to the books, such as creating and removing a book category, adding a book to a category, and modifying the book information. The Client section lets the end user perform online transactions, such as searching for books based on either author or book title, and buying books.

This chapter discusses the architecture and database structure of the online shopping cart application. This chapter shows how to create the shopping cart application using PHP as the scripting language and MySQL as the database. This chapter discusses the process of administering the data related to books, such as adding and removing a new category, adding and removing a book from the database, viewing the book information, and placing an order to purchase the selected books. The chapter also describes the process of performing online transactions, such as buying a book, or searching for specific books based on the author name or book title.

Application Architecture

The architecture of the online shopping cart application contains two sections, Admin and Client. In addition, this application uses a database created in MySQL server to store data.

The Admin and Client Sections

The Admin section of the online shopping cart application performs administrative tasks to maintain the state of the application, such as adding or removing a category and adding or removing a book from a category. The Client section enables the end user perform activities related to online transactions, such as registering with the application, navigating through the various available categories to search for specific books, and adding the selected books to the cart.

Figure 9-1 shows the architecture of the online shopping cart application:

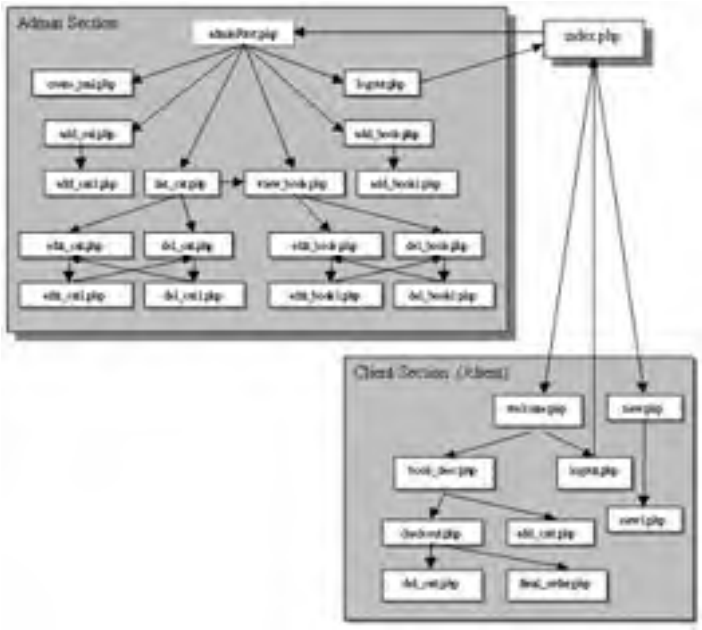


Figure 9-1: Online Shopping Cart Application Architecture

The index.php page is the first page of the application that allows the end users to log on to the application. The end user logs on as an administrator, using the administrative user ID. The index.php file contains a hyperlink to the new.php file that registers new end users and creates new client accounts.

The adminFirst.php page is the first Web page of the Admin section. It contains hyperlinks to six other Web pages, which perform specific operations related to the administration of the application. The Web pages include:

- The logout.php Web page is displayed when the end user logs out from the application.
- The add_book.php Web page adds a book in a category. The view_book.php Web page displays the details of a book.
- The list_cat.php Web page lists all the existing categories of the books. The list_cat.php Web page contains hyperlinks to the edit_cat.php and del_cat.php Web pages.
- The add_cat.php Web page adds a new category to the list of existing categories.

- The create_xml.php Web page generates the orders by creating an Extensible Markup Language (XML) file.
- The view_book.php Web page also lets you edit and delete a book from a category. It contains hyperlinks to the edit_book.php and del_book.php Web pages.

The edit_cat.php Web page receives the required parameters from the end user to edit a category and calls the edit_cat1.php Web page to edit a category. The del_cat.php Web page receives the required parameters from the end user to delete a category and calls the del_cat1.php Web page to delete a category.

The edit_book.php Web page receives the required parameters from the end user to edit the book information and calls the edit_book1.php Web page to edit a book. The del_book.php Web page receives the required parameters from the end user to delete a book and calls the del_book1.php Web page to delete a book.

The Web pages of the client section are stored in the client folder of the application. The index.php Web page directs all registered clients with valid account IDs to the welcome.php Web page. This is the first page of the client section. The Welcome.php Web page contains hyperlinks to the logout.php and book_desc.php Web pages. The logout.php Web page lets a client log out from the application and go back to the index.php Web page. The book_desc.php Web page contains a hyperlink to the checkout.php and add_cart.php Web pages. The add_cart.php Web page adds the selected book in the cart of the end user or client. The check_out.php Web page shows the status of the cart and contains a link to the final_order.php Web page, which displays the order number of the client.

Database Structure

The online shopping cart application uses the MySQL database, which consists of the category, book, user_profile, order1, transaction, and tmp tables. The category table stores information related to the categories of the books, such as category ID and name of the category.

Table 9-1 lists the structure of the category table:

Table 9-1: Structure of the category Table

field name	data type
tbl_id	int(11), primary key, auto_increment
item_type	varchar(50), unique, NOT NULL

The book table stores information about the books available in the online shopping cart application. The book table consists of information, such as book ID, category, book title, and author name.

Table 9-2 lists the structure of the book table:

Table 9-2: Structure of the book Table

field name	data type
item_no	varchar(20), primary key, NOT NULL
item_type	varchar(50) references category(item_type)
title	varchar(60), NOT NULL
author	varchar(60), NOT NULL
price	float, NOT NULL

The user_profile table stores the registration information of the end users registered using the online shopping cart application. The user_profile table stores information, such as the user name, password, address, phone number, and credit card information.

Table 9-3 lists the structure of the user_profile table:

Table 9-3: Structure of the user_profile Table

field name	data type
name	varchar(40), NOT NULL
user_id	varchar(20), primary key
password	varchar(20), NOT NULL
address_line1	varchar(40), NOT NULL
address_line2	varchar(40), NOT NULL
city	varchar(20), NOT NULL
country	varchar(20), NOT NULL
pin	varchar(20), NOT NULL
email_id	varchar(20), NOT NULL
phone_number	varchar(20), NOT NULL
card_no	varchar(20), NOT NULL

card_type	varchar(20), NOT NULL
expirty_date	varchar(20), NOT NULL
fax_number	varchar(20), NOT NULL

The order1 table stores the order information, such as the order number, book ID, and user ID.

Table 9-4 lists the structure of the order1 table:

Table 9-4: Structure of the order1 Table

field name	data type
tbl_id	int(11) references category(tbl_id)
order_no	int(11) primary key auto_increment
item_no	varchar(20) references book(item_no)
user_id	varchar(40) references user_profile(user_id)

The tmp table stores the session information for each order.

Table 9-5 lists the structure of the tmp table:

Table 9-5: Structure of the tmp Table

field name	data type
order_no	int(11), primary key auto_increment
user_id	varchar(20), NOT NULL
item_no	varchar(20), NOT NULL
sesid	varchar(50), NOT NULL
date	date, NOT NULL

Creating the Database and Tables

The online shopping cart application uses the shop.sql script to create the required databases and tables. You need to run the shop.sql script from the command prompt of the MySQL server to create the databases and the tables. The code to run the shop.sql script from the command line is:

```
mysql shop < shop.sql
```

In the above code, the shop parameter specifies the name of the database, and shop.sql is the script file that consists of the SQL statements to create the tables required to work with the online shopping cart application.

Listing 9-1 shows the content of the shop.sql script:

Listing 9-1: Creating Database Tables

```
# Table structure for table 'category'
CREATE TABLE category (
   tbl_id int(11) NOT NULL auto_increment,
   item_type varchar(20) NOT NULL,
   PRIMARY KEY (tbl_id),
   UNIQUE item_type (item_type)
);
# Table structure for table 'book'
CREATE TABLE book (
   item_no varchar(20) NOT NULL,
   item_type varchar(20) NOT NULL references category(item_type),
   title varchar(60) NOT NULL,
   author varchar(60) NOT NULL,
   price float DEFAULT '0' NOT NULL,
   PRIMARY KEY (item_no)
);
# Table structure for table 'user_profile'
CREATE TABLE user_profile (
   name varchar(40) NOT NULL,
   user_id varchar(20) NOT NULL,
   password varchar(20) NOT NULL,
   address_line1 varchar(40) NOT NULL,
   address_line2 varchar(40),
   city varchar(20) NOT NULL,
   country varchar(20) NOT NULL,
   pin varchar(20) NOT NULL,
   email_id varchar(20) NOT NULL,
   phone_number varchar(20) NOT NULL,
   card_no varchar(20) NOT NULL,
   expiry_date varchar(20) NOT NULL,
card_type varchar(20) NOT NULL,
   fax_number varchar(20) NOT NULL,
```

```
   PRIMARY KEY (user_id)
);
# Table structure for table 'order1'
CREATE TABLE order1 (
   tbl_id int(11) NOT NULL references category (tbl_id),
   order_no int(11) auto_increment,
   item_no varchar(20) NOT NULL references book(item_no),
   user_id varchar(40) NOT NULL references user_profile(user_id),
   PRIMARY KEY (order_no)
);
# Table structure for table 'tmp'
CREATE TABLE tmp (
order_no int(11) NOT NULL auto_increment,
   user_id varchar(20) NOT NULL,
   item_no varchar(20) NOT NULL,
   sesid varchar(50) NOT NULL,
   date date DEFAULT '0000-00-00' NOT NULL,
   PRIMARY KEY (order_no)
);
# Table structure for table 'transaction'
CREATE TABLE transaction (
   order_no int(11) NOT NULL auto_increment,
   user_id varchar(20) NOT NULL,
   date date DEFAULT '0000-00-00' NOT NULL,
   status varchar(20) NOT NULL,
   PRIMARY KEY (order_no)
);
```

The above listing creates the book, category, order1, user_profile, tmp, and transaction tables in the shop database. After creating all the tables, you need to insert a record in the user_profile table that contains the user ID and password. Use the following code to insert a row in the user_profile table:

```
INSERT INTO user_profile VALUES ( 'admin', 'admin', 'admin', '', '', '', '', '', '', '', '', '', '',
```

The above code creates an account that is used by the end user working as an administrator, with admin as the user ID and password.

◄ PREVIOUS NEXT ►

The Admin Section of the Online Shopping Cart Application

The administration process for the online shopping cart application involves adding a category, modifying a category, deleting a category, adding a new book, updating book information, and removing a book from the shop database.

The files of the Admin section are stored in the project folder. The client folder, present in the project folder, stores all the files required to perform the online transactions, such as the new.php file used for client registration. The project folder, apart from storing the client folder, stores the files required for administration, such as the add_cat.php file used for creating a new category. You need to specify the directory path of the files of the admin section in the address bar of the Web browser, to administer the data for the online shopping cart application. For example, if the PHP files are stored in the document root, /var/www/html, of the Web server, the path to administer the online shopping cart application is, http://localhost/Project/index.php.

Accessing the Shopping Cart Application

To work with the application, end users need to log on either as a client or as an administrator. The index.php file, present in the project folder, represents the login page. The user ID and password for the administrator is admin. The end user needs to register to create a new client account. If the end user is logged in with the admin account, only then the end user can work as an administrator.

Listing 9-2 shows the index.php file that creates a login form to accept the user name and password information from the end user:

Listing 9-2: The index.php File

```
<html>
<body>
<table border="0" width="100%" cellpadding="1" cellspacing="1" bgcolor="#FFFFFF">
<tr><td><?php
print "<form method=\"get\" action=\"login.php\">
<tr bgcolor=\"#E8E8E8\" height=\"4\">
<td align=\"right\">
<font face=\"arial\" size=2 color=\"#FF0000\">USER NAME:</font>
</td>
<td align=\"left\">
<INPUT TYPE=\"text\" NAME=\"user_id\">
</td></tr>
<tr bgcolor=\"#E8E8E8\" height=\"4\">
<td align=\"right\">
<font face=\"arial\" size=2 color=\"#FF0000\">PASSWORD:</font></td>
<td align=\"left\">
<INPUT TYPE=\"password\" NAME=\"passwd\">
</td></tr>
<tr bgcolor=\"#E8E8E8\" height=\"4\">
<td align=\"center\" colspan=\"2\">
<INPUT TYPE=\"submit\" value=\"LOGIN\"></td></tr>
<tr bgcolor=\"#E8E8E8\" height=\"1\">
<td width=\"100%\" valign=\"middle\" align=\"center\" colspan=\"2\"> </td>
</tr>
<tr bgcolor=\"#E8E8E8\" height=\"1\">
<td width=\"100%\" valign=\"middle\" align=\"center\" colspan=\"2\">
<a href=\"client/new.php\">New User</a>
</td> </tr></form>";
print"</td></tr></table></body></html>";
?>
</tr>
<?php
include "bottom.php";
?>
```

The above listing creates a login page to accept the user name and password information. The user name and password determines whether the end user is an administrator or a client.

Figure 9-2 shows the login page created by the index.php file:

Figure 9-2: Viewing index.php File in Mozilla Web Browser

To log on as an administrator, end users need to use the login ID, admin. The password for the admin login ID is also admin. The administrator can check or change the password of the admin login in the user_profile table. When end user clicks the Submit button, the login.php file is processed to verify the user name and password.

Listing 9-3 shows the content of the login.php file:

Listing 9-3: Verifying the User Name and Password Information

```
<?php
$user_id=$_GET['user_id'];
$passwd=$_GET['passwd'];
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select * from user_profile where user_id='$user_id' and
password='$passwd'", $db);
$row = mysql_fetch_array($result);
$user=$row["user_id"];
if ($user=="admin")
{
   print "<html><head><title></title><head><body>
   <form name=\"sess\" action=\"adminFirst.php\" method=\"post\">
   </form>
   <script>
   document.sess.submit();
   </script>
   </body></html>";
}
else
{
   print"<html><head><title></title><head><body>
   <form name=\"sess\" action=\"client/logon.php\" method=\"get\">
   <input type=\"hidden\" name=\"login\" value=\"$user\">
   </form>
   <script>
   document.sess.submit();
   </script>
   </body></html>";
}
?>
```

The above listing verifies the submitted user name and password. If the specified user name and password do not match with the data in the tables, an error message is displayed to the end user. If the admin login ID is used, the home page of the administrative interface, adminFirst.php, is displayed in the Web browser. If any other login ID is used, the logon.php file present in the client folder of the application is displayed in the Web page.

Creating the Home Page for the Administrative Interface

All the pages in the Admin section of the application are divided into three sections: top, middle, and right.

Figure 9-3 shows the structure of the Web pages of the Admin section:

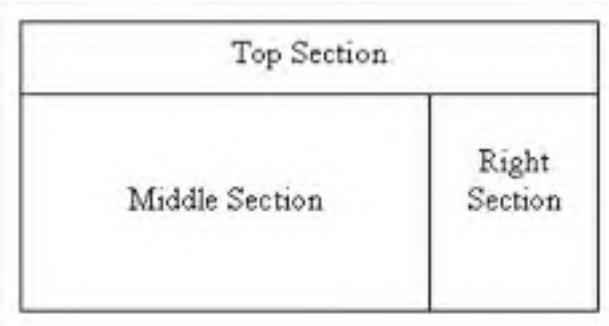


Figure 9-3: Structure of Web Pages of the Admin Section

```
<?php
include "tophtml.php";
include "middle.php";
include "right.php";
?>
```

The above code shows three PHP files included in the adminFirst.php file. The first file, tophtml.php, represents the top section.

Listing 9-4 shows the content of the tophtml.php file that the administrator can use to create the top section of the home page:

Listing 9-4: Creating the Top Section

```
<html>
<head>
<title>Shop Admin</title>
</head>
<body>
<div align="center">
<center>
<table border="0" width="600" cellspacing="0" cellpadding="0" bgcolor="#FFFFFF">
<tr>
<td width="600" height="20" colspan="5" bordercolor="#00FF00" bgcolor="#008000">
<p align="center"><b><font face="Arial" size="3" color="#FFFFFF"> SHOP
(ADMIN)</font></b></td>
</tr>
<tr height="2">
<td width="600" height="2" colspan="5" bgcolor="#000080"></td>
</tr>
```

The above listing creates the top section of the home page for the administrative interface of the online shopping cart application.

The middle section is represented by the middle.php file. The following code shows the content of the middle.php file:

```
<tr width="498" height="300" >
<td colspan="2" align="center">Select any option from the menu.</td>
```

The above code defines the first column of the row that contains the middle and right sections.

The right section displays the administrative options, such as adding a category, retrieving the category list, adding a book, and changing the password.

Listing 9-5 shows the content of the right.php file to create the right section of the home page:

Listing 9-5: Creating the Right Section of the Home Page

```
<td width="1" height="300" bgcolor="#000080"></td>
<td width="100" height="300" valign="top">
<table border="0" width="99" cellpadding="1" cellspacing="1" bgcolor="#FFFFFF">
<?php
echo "<tr  bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"logout.php\">
<font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B> LOGOUT</font></a></td></tr>
<tr bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"add_book.php\"><font face=\"verdana\" size=\"1\"
color=\"#FFFFFF\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B>Add New
Book</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"add_cat.php\"><font
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B>Add
Category</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a
href=\"create_xml.php\"><font face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font
style=\"font-size:11px;\" face=\"Helvetica\" color=\"#FFFFFF\"><B>Generate
Order</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"list_cat.php\"><font
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B> Category
List</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><b><font face=\"verdana\"
size=\"1\" color=\"#FFFFFF\">Select category</font></b></td></tr>";
$db=mysql_connect('localhost','root','');
if (!$db)
{
echo "Error When connecting to Database";
<td width="1" height="300" bgcolor="#000080"></td>^M
<td width="100" height="300" valign="top">^M
<table border="0" width="99" cellpadding="1" cellspacing="1" bgcolor="#FFFFFF">
<?php
echo "<tr  bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"logout.php\">
<font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B> LOGOUT</font></a></td></tr>
<tr bgcolor=\"#FF0000\">
<td align=\"center\"><a href=\"add_book.php\"><font face=\"verdana\" size=\"1\"
color=\"#FFFFFF\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FFFFFF\"><B>Add New
Book</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"add_cat.php\"><font
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B>Add
Category</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a
href=\"create_xml.php\"><font face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font
style=\"font-size:11px;\" face=\"Helvetica\" color=\"#FFFFFF\"><B>Generate
Order</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><a href=\"list_cat.php\"><font
```

```
face=\"verdana\" size=\"1\" color=\"#FFFFFF\"><font style=\"font-size:11px;\"
face=\"Helvetica\" color=\"#FFFFFF\"><B> Category
List</b></font></a></td></tr>
<tr bgcolor=\"#FF0000\"><td align=\"center\"><b><font face=\"verdana\"
size=\"1\" color=\"#FFFFFF\">Select category</font></b></td></tr>";
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
mysql_query("use $db");
$result = mysql_query("select item_type from category");
while($row = mysql_fetch_array($result))
{
   print "<tr bgcolor=\"#E1E1E1\"><td align=\"center\"><a
   href=\"view_book.php?item_type=$row[0]&p=0\"><font style=\"font-size:11px;\"
   face=\"Helvetica\"
   color=\"#FF0000\"><B>$row[0]</font></a></td></tr>";
}
?>
<tr bgcolor="#FFFFFF"><td
align="center"> </font></a></td></tr>
</table>
</td>
<td width="1" height="300"
bgcolor="#000080"></td></tr></table></body></html>
```

The above listing creates the right section of the home page for the administrative interface of the online shopping cart application.

Figure 9-4 shows the home page for the administrative interface of the online shopping cart application.

Figure 9-4: Home Page of the Administrative Interface

Logging Out as an Administrator

When the administrator clicks the LOGOUT link, the logout.php file logs you out from the online shopping cart application.

Listing 9-6 shows the content of the logout.php file that lets the administrator log out from the online shopping cart application:

Listing 9-6: The logout.php File for the Administrator

```
<?php
include "tophtml.php";
echo "<tr width=\"600\" height = \"300\"><td width = \"600\" colspan = \"5\">";
?>
<table border="0" width="100%" height="300" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td align="center" colspan="5"> <FONT SIZE="3" FACE="verdana"
color="#FF3300"><b>Logout successfully</b></FONT></td></tr>
<tr bgcolor="#E8E8E8"> <td align="center" colspan="5"><a
href="index.php">Sign in again</a></td></tr>
</table>
</td></tr></table>
```

Figure 9-5 shows a message, confirming that the administrator has logged out from the application:

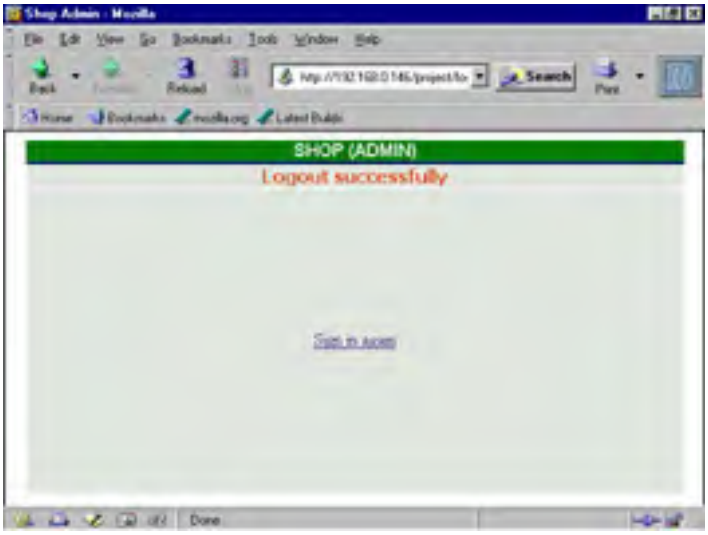


Figure 9-5: Message Confirming Administrative Log Off

Adding a New Category

When the administrator clicks the Add Category link shown in Figure 9-4, the Web server executes the add_cat.php file that displays a Web page to accept information for a new category.

Listing 9-7 shows how to create a Web page to add a new category in the database:

Listing 9-7: Creating a Web Page to Add a New Category

```
<?php
include "tophtml.php";
?>
<tr width="499"><td colspan="2">
<form name="addbook" action="add_cat1.php" action="GET">
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>ADD CATEGORY</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Category
Name: </font></td>
<td width="50%" valign="middle" align="left"><input type="text"
name="item_type"></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Add
Category" onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php include "right.php" ?>
<script>
function check()
{
   var error="false";
   if((document.addbook.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   if (error=="false")
   {
      document.addbook.submit();
   }
}
</script>
```

The above listing creates a Web page to accept information from the administrator to add a new category to the database.

Figure 9-6 shows the Web page to add a new category to the database table:

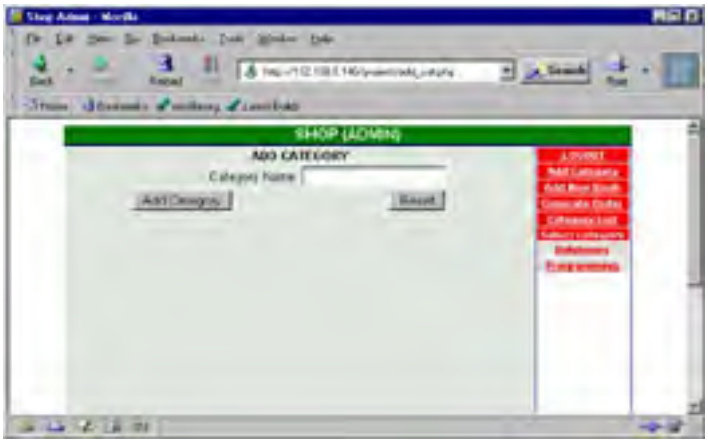


Figure 9-6: Web Page to Add a New Category

When the administrator clicks the Add Category button, the add_cat1.php file is executed by the Web server to add a new category to the database, as shown in Listing 9-8:

Listing 9-8: Adding a New Category to the Database

```
<?php
include "tophtml.php";
echo "<tr width=\"600\" height=\"300\"><td width=\"499\" colspan = \"2\">";
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$temp=$_GET['item_type'];
if(!$result_in=mysql_query("insert into category(tbl_id, item_type) values('NULL', '$temp')",
$db))
{
   $error=mysql_error();
}
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" height="4"><font face="Arial" size="2"><b>
<?php
if($error)
echo "<font color=\"#FF0000\">$error</font><br><br><a
href=\"javascript:history.back();\">Go Back</a>";
else
echo "<font color=\"#000000\">You have added category successfully</font>";
?>
</b></font></td></tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center"> 
</td></tr>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing inserts the category information in the database. If the category is added successfully, the message, You have added category successfully, is displayed to the administrator; else, an error message is displayed.

Adding a New Book

When the administrator clicks the Add New Book link shown in Figure 9-4, the add_book.php file creates a Web page to accept information for a new book from the administrator.

Listing 9-9 shows the add_book.php file that represents a Web page to add a new book to the database table:

Listing 9-9: The add_book.php File

```
<?php
include "tophtml.php";
?>
<tr width="499" ><td colspan="2">
<form name="addbook" action="add_book1.php" action="post">
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
```

```
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>ADD NEW BOOK</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book No:
</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="item_no"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Category:
</font></td>
<td width="50%" valign="middle" align="left">
<!--
<INPUT TYPE="text" NAME="item_type">
-->
<SELECT NAME="item_type">
<?php
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_cat=mysql_query("select * from category", $db);
while ($row_cat=mysql_fetch_array($result_cat))
{
   print "<option value=\"$row_cat[1]\">$row_cat[1]</option>";
}
?>
</SELECT>
</td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book
Title: </font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="title"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book
Author: </font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="author"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Book
Price($): </font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text"
NAME="price"></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Add Book"
onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php include "right.php"?>
<script>
function check()
{
   var error="false";
   if((document.addbook.item_no.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book No.");
   }
   /*
   if((document.addbook.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   */
   if((document.addbook.title.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Title");
   }
   if((document.addbook.author.value == "") && (error == "false"))
```

```
    {
       error="true";
       alert("Please Enter Author name.");
    }
    if((document.addbook.price.value == "") && (error == "false"))
    {
       error="true";
       alert("Please Enter Book Price.");
    }
    if (error=="false")
    {
       document.addbook.submit();
    }
}
</script>
```

The above listing creates a Web page to accept information to add a new book to the database, as shown in Figure 9-7:

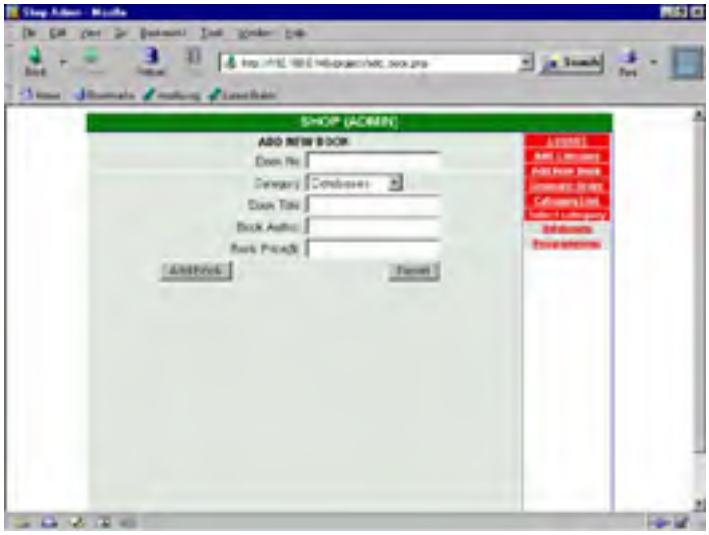


Figure 9-7: Web Page to Add a New Book

When the administrator clicks the Add Book button, the add_book1.php file adds a new book to the database.

Listing 9-10 shows the content of the add_book1.php file:

Listing 9-10: The add_book1.php File

```
include "tophtml.php";
$db=mysql_connect('localhost','root','');
echo "<tr width=\"600\" height=\"300\"><td width=\"499\" colspan=\"2\">";
if (!$db)
{
   echo "Error When connecting to Database";
}
$item_no=$_GET['item_no'];
$item_type=$_GET['item_type'];
$title=$_GET['title'];
$author=$_GET['author'];
$price=$_GET['price'];
mysql_select_db("shop", $db);
if(!$result_in=mysql_query("insert into book(item_no,item_type, title, author, price)
values('$item_no', '$item_type', '$title','$author',$price)", $db))
{
   $error=mysql_error();
}
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" height="4"><font face="Arial" size="2"><b>
<?php
if($error)
echo "<font color=\"#FF0000\">$error</font><BR><BR><a
href=\"javascript:history.back();\">Go Back</a>";
```

```
else
echo "<font color=\"#000000\">You have added one book successfully</font>";
?>
</b></font></td></tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center"> 
</td></tr>
</table>
</td>
<?php
include "right.php"
?>
```

The above listing inserts the book data in the database. If the book is added successfully, the message, You have added one book successfully, is displayed. If an error occurs, an error message is displayed in the middle section of the Web page.

Generating Orders

End users can place an order to buy the required books. The order is stored in an XML file. When the administrator clicks the Generate Order hyperlink, the Orders.xml XML file is created if it does not already exist in the /var/www/html directory. The create_xml.php file generates the Orders.xml file.

Listing 9-11 shows the content of the create_xml.php file:

Listing 9-11: The create_xml.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan = \"2\" align = \"center\">";
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select item_no, date, status, transaction.order_no from transaction, order1
where status='Pending' and transaction.order_no=order1.order_no", $db);
$doc = new_xmldoc("1.0");
$root = $doc->add_root("Orders");
while ($row=mysql_fetch_array($result))
{
   $result2=mysql_query("select title, author, price from book where item_no=$row[0]", $db);
   $row2=mysql_fetch_array($result2);
   $record = $root->new_child("Order", "");
   $record->set_attribute("number", $row[3]);
   $record->new_child("BookName", $row2[0]);
   $record->new_child("BookID", $row[0]);
   $record->new_child("Author", $row2[1]);
   $record->new_child("Price", $row2[2]);
   $record->new_child("Status", $row[2]);
   $record->new_child("Order Date", $row[1]);
}
$fp = fopen("/var/www/html/Orders.xml", "w+");
$rt= $doc->dumpmem();
fwrite($fp, $rt, strlen($rt));
echo "XML File Created";
?>
</td>
<?php
include "right.php"
?>
```

The above listing creates the Orders.xml file in the var/www/html directory. If the file is created successfully, the message, XML File Created, is displayed in the middle section of the Web page.

Listing Categories

When the administrator clicks the Category List hyperlink in the right panel of the home page, shown in Figure 9-4, the list_cat.php file is processed to retrieve and display all the category names.

Listing 9-12 shows how to create a Web page to display the names of all the categories:

Listing 9-12: The list_cat.php File

```
<?php
include "tophtml.php";
?>
<tr width="499"><td colspan="2">
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
```

```
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Category Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from category", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   //if ($j < $rec_per_page) {
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_cat.php?tbl_id=$row[0]\"><font face=\"Arial\"
   size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"del_cat.php?tbl_id=$row[0]&item_type=$row[1]\"><font face=\"Arial\"
   size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print "<tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php include "right.php"?>
```

The above listing retrieves a list of all the categories that exist in the database.

Figure 9-8 shows the output when list_cat.php is viewed in the Web browser:

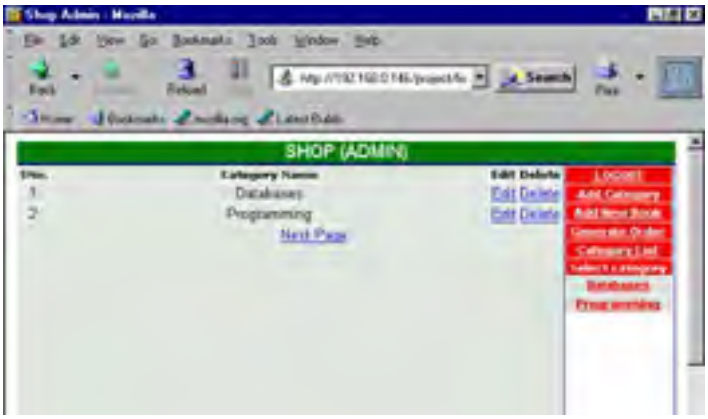


Figure 9-8: Web Page to Display All Category Names

Modifying a Category

When the administrator clicks the Edit link corresponding to a category name, as shown in Figure 9-8, the edit_cat.php file is processed. The edit_cat.php file creates a Web page that accepts data to update the category name.

Listing 9-13 shows the content of the edit_cat.php file:

Listing 9-13: The edit_cat.php File

```
<?php
include "tophtml.php";
$db=mysql_connect('localhost','root','');
echo "<tr width=\"499\"><td colspan = \"2\">";
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$temp=$_GET['tbl_id'];
$result=mysql_query("select * from category where tbl_id=$temp", $db);
$row=mysql_fetch_array($result);
$temp_val=$row[1];
?>
<form name="editcat" action="edit_cat1.php" action="post">
<input type="hidden" name="tbl_id" value='<?php echo "$temp" ?>'>
<input type="hidden" name="old_item_type" value='<?php echo "$row[1]"?>'>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>EDIT CATEGORY</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter
Category Name</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="item_type"
value='<?php echo "$temp_val" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Edit
Category" onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php
include "right.php"?>
<script>
function check()
{
   var error="false";
   if((document.editcat.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   if (error=="false")
   {
      document.editcat.submit();
   }
}
</script>
```

The above listing displays a Web page to modify the existing categories.

Figure 9-9 shows the Web page that appears when the administrator clicks the Edit hyperlink corresponding to the Databases category:

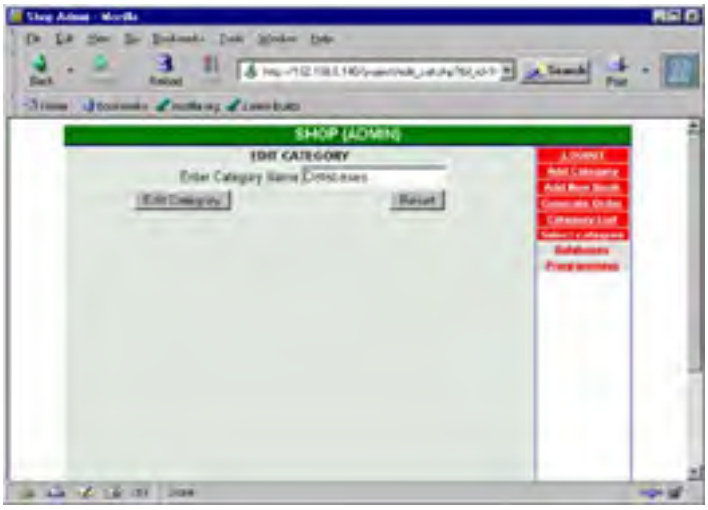


Figure 9-9: Modifying a Category Name

When the administrator clicks the Edit Category button, the edit_cat1.php file is processed to modify the data for an existing category in the database table.

Listing 9-14 shows the content of the edit_cat1.php file:

Listing 9-14: The edit_cat1.php File

```
<?php
include "tophtml.php";
$db=mysql_connect('localhost','root','');
echo "<tr width=\"499\"><td colspan=\"2\">";
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$temp1=$_GET['item_type'];
$temp2=$_GET['tbl_id'];
$result_catup=mysql_query("update category set item_type='$temp1' where tbl_id='$temp2'", $db);
$result_catup=mysql_db_query("$db","update $book_tbl set item_type='$item_type' where
item_type='$old_item_type'");
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Category Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from category", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   //if ($j < $rec_per_page) {
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a href=\"del_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
```

```
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing modifies the category name in the database table.

Deleting a Category

When the administrator clicks the Delete link corresponding to a category name, as shown in Figure 9-8, the del_cat.php file, which deletes the selected category from the database, is processed.

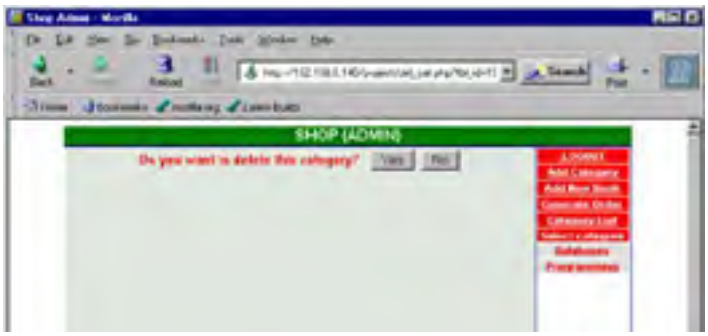
Listing 9-15 shows the content of the del_cat.php file that prompts the administrator before deleting a category from the database table:

Listing 9-15: The del_cat.php File

```
<?php
include "tophtml.php";
$temp2=$_GET['tbl_id'];
$temp1=$_GET['item_type'];
echo "<tr width=\"499\"><td colspan=\"2\">";
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="100%" valign="middle" align="center" colspan="4">
<form name="delcat" action="del_cat1.php" method="GET">
<input type="hidden" name="tbl_id" value='<?php echo "$temp2" ?>'>
<input type="hidden" name="item_name" value='<?php echo "$temp1" ?>'>
<font face="arial" size="2" color="#FF0000"><b>Do you want to delete this category?
</b></font>  <input type="submit" name="del"
value="Yes">  <input type="submit" name="del" value="No">
</form>
</td>
</tr>
<?php
print "<tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing creates a Web page, which prompts the administrator before deleting a category from the database.

Figure 9-10 shows the Web page that appears when the administrator clicks the Delete hyperlink corresponding to the Programming category, and verifies whether or not to delete the selected category:



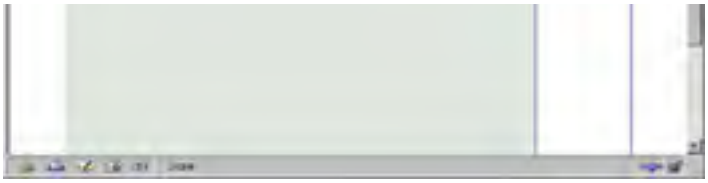


Figure 9-10: Web Page to Delete a Category

When the administrator clicks the Yes button, the del_cat1.php file is processed to delete the category from the database table, as shown in Listing 9-16:

Listing 9-16: The del_cat1.php File

```
<?php
include "tophtml.php";
$tbl_id=$_GET['tbl_id'];
$del=$_GET['del'];
$item_type=$_GET['item_name'];
echo "<tr width=\"499\"><td colspan = \"2\">";
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
if ($del=="Yes")
{
   $result_catdel=mysql_query("delete from category where tbl_id='$tbl_id'", $db);
}
$result_book=mysql_query("delete from book where item_type='$item_type'", $db);
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Category Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from category", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   //if ($j < $rec_per_page) {
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a href=\"del_cat.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
```

```
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing deletes a category from the database table. The administrator is redirected to the Web page that displays the updated category list.

Modifying Book Data

When the administrator clicks any of the category items displayed in the right section of Figure 9-10, the view_book.php file is executed by the Web server to display a list of all the books present in that category.

Listing 9-17 shows the content of the view_book.php file:

Listing 9-17: The view_book.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan=\"2\">";
$item_type=$_GET['item_type'];
echo $item_type;
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="5%" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="85%" valign="middle" align="center"><font size="1"
face="verdana"><b>Book Name</b></font></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="5%" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select * from book where item_type='$item_type'", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   $j++;
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"5%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"85%\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[2]</font></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"edit_book.php?tbl_id=$row[0]\"><font face=\"Arial\"
   size=\"2\">Edit</font></a></td>
   <td width=\"5%\" align=\"center\"><a
   href=\"del_book.php?tbl_id=$row[0]&item_type=$row[1]\"><font face=\"Arial\"
   size=\"2\">Delete</font></a></td>
   </tr>";
   //}
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"        
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
```

```
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"4\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing displays a Web page that contains information about all the books present in a category.

Figure 9-11 shows the output of the view_book.php file when the Database category is selected from the right section of the adminFirst.php Web page:

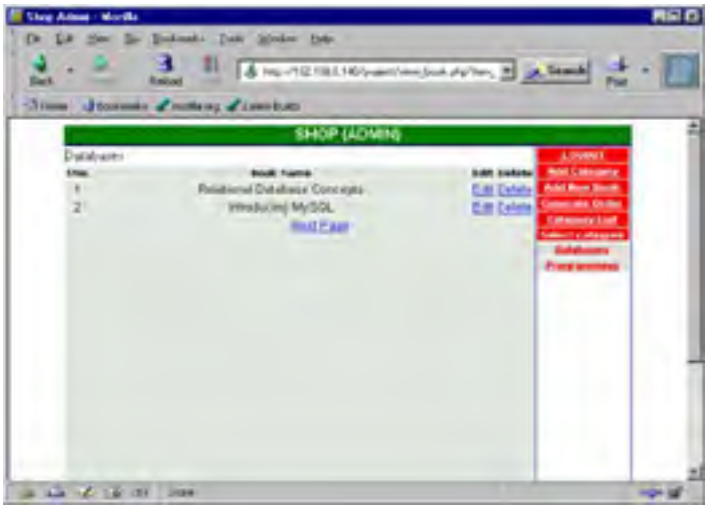


Figure 9-11: Books Present in the Database Category

The administrator needs to click the Edit hyperlink to modify the data of a book in the database. When the administrator clicks the Edit hyperlink corresponding to a book name, the edit_book.php file is executed, which lets the administrator enter new data for the selected book.

Listing 9-18 shows the content of the edit_book.php file:

Listing 9-18: The edit_book.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan=\"2\">";
$db=mysql_connect('localhost','root','');
$tbl_id=$_GET['tbl_id'];
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result=mysql_query("select * from book where item_no='$tbl_id'", $db);
$row=mysql_fetch_array($result);
?>
<form name="addbook" action="edit_book1.php" action="GET">
<input type="hidden" name="item_no" value='<?php echo"$tbl_id" ?>'>
<input type="hidden" name="item_type" value='<?php echo"$item_type" ?>'>
<input type="hidden" name="p" value='<?php echo"$p" ?>'>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2" height="4"><font face="Arial"
size="2"><b>EDIT YOUR BOOK</b></font></td></tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial"> Book
No.</font></td>
<td width="50%" valign="middle" align="left"><?php echo"$row[0]" ?></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Category</font></td>
<td width="50%" valign="middle" align="left">
<!--
<INPUT TYPE="text" NAME="item_type"  value='<?php echo"$row[1]" ?>'></td>
```

```
-->
<SELECT NAME="item_type">
<?php
$result_cat=mysql_query("select * from book", $db);
while ($row_cat=mysql_fetch_array($result_cat))
{
   if ($row[1] == $row_cat[1])
   print "<option value=\"$row_cat[1]\" selected>$row_cat[1]</option>";
   else
   print "<option value=\"$row_cat[1]\">$row_cat[1]</option>";
}
?>
</SELECT>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Title</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="title"
value='<?php echo"$row[2]" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Author</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="author"
value='<?php echo"$row[3]" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="right"><font size="2" face="Arial">Enter Book
Price ($)</font></td>
<td width="50%" valign="middle" align="left"><INPUT TYPE="text" NAME="price"
value='<?php echo"$row[4]" ?>'></td>
</tr>
<tr bgcolor="#E8E8E8" height="4">
<td width="50%" valign="middle" align="center"><INPUT TYPE="button" value="Edit Book"
onclick="check();"></td>
<td width="50%" valign="middle" align="center"><INPUT TYPE="reset"></td>
</tr>
<tr bgcolor="E8E8E8">
<td width="100%" align="center" colspan="2"><font face="Arial"
size="2"> </font></td></tr>
</table>
</form>
</td>
<?php
include "right.php" ?>
<script>
function check()
{
   var error="false";
   if((document.addbook.item_type.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Category");
   }
   if((document.addbook.title.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Title");
   }
   if((document.addbook.author.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Author name.");
   }
   if((document.addbook.price.value == "") && (error == "false"))
   {
      error="true";
      alert("Please Enter Book Price.");
   }
   if (error=="false")
   {
      document.addbook.submit();
   }
}
</script>
```

The above listing creates a Web page to accept modified data for an existing book from the administrator, as shown in Figure 9-12:

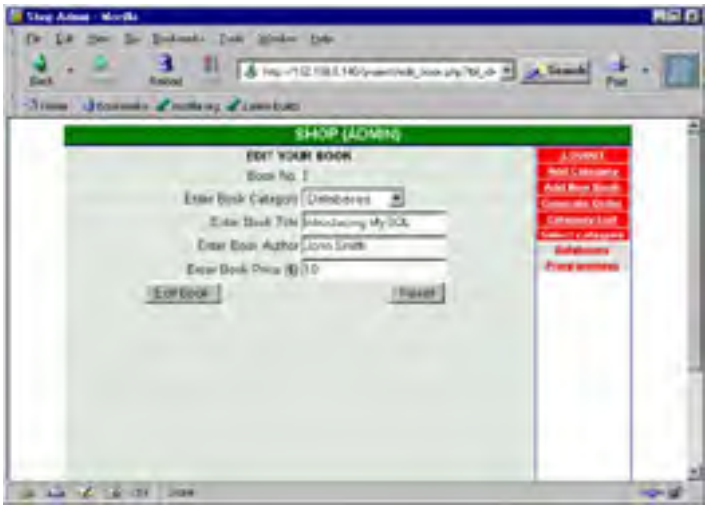


Figure 9-12: Modifying Book Data

When the administrator clicks the Edit Book button, the edit_book1.php file executes to modify the data for an existing book in the database table.

Listing 9-19 shows the content of the edit_book1.php file:

Listing 9-19: The edit_book1.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td colspan=\"2\">";
$item_no=$_GET['item_no'];
$item_type=$_GET['item_type'];
$author=$_GET['author'];
$price=$_GET['price'];
$title=$_GET['title'];
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_edit=mysql_query("update book set
item_type='$item_type',title='$title',author='$author',price=$price where
item_no='$item_no'", $db);
?>
<table border="0" width="100%" height="480" cellpadding="1" cellspacing="1"
bgcolor="#FFFFFF">
<tr bgcolor="#E8E8E8" height="4">
<td width="32" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="64" valign="middle" align="center"><font size="1"
face="verdana"><b>Item
No.</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Category</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Title</b></font></td>
<td width="68" valign="middle" align="center"><b><font size="1"
face="verdana">Author</font></b></td>
<td width="33" valign="middle" align="center"><b><font size="1"
face="verdana">Price($)</font></b></td>
<td width="39" valign="middle" align="center"><b><font size="1"
face="verdana">Edit</font></b></td>
<td width="39" valign="middle" align="center"><b><font size="1"
face="verdana">Delete</font></b></td>
</tr>
<?php
$j=0;
$rec=$rec_per_page + 1;
$result = mysql_query("select * from book where item_type='$item_type'", $db);
$num = mysql_num_rows($result);
while($row = mysql_fetch_array($result))
{
   print "
   <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"32\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"64\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[0]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
```

```
   size=\"2\">$row[2]</font></td>
   <td width=\"68\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[3]</font></td>
   <td width=\"33\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[4]</font></td>
   <td width=\"33\" align=\"center\"><a
   href=\"edit_book.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Edit</font></a></td>
   <td width=\"6\" align=\"center\"><a href=\"del_book.php?tbl_id=$row[0]\"><font
   face=\"Arial\" size=\"2\">Delete</font></a></td>
   </tr>";
}
print " <tr height=\"4\" bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"8\"><font face=\"Arial\" size=\"2\">";
if ($p > 0)
{
   $p1 = $p-1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p1\">Previous Page
   </a>";
}
print
"         
  ";
if ($num > $rec_per_page)
{
   $p2 = $p+1;
   echo "<a href=\"view_book.php?item_type=$item_type&p=$p2\">Next Page</a>";
}
print "</font></td></tr>";
print " <tr bgcolor=\"E8E8E8\">
<td width=\"100%\" align=\"center\" colspan=\"8\"><font face=\"Arial\"
size=\"2\"> </font></td></tr>";
?>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing modifies the book data in the database table and redirects the administrator to the Web page that displays the books data for the selected category.

Removing a Book

The administrator needs to click the Delete hyperlink corresponding to a book name, as shown in Figure 9-11, to delete a book from a category. The del_book.php file executes to delete the book from the category when the administrator clicks the Delete hyperlink. The del_book.php file represents a Web page that prompts the administrator before deleting the book from the database.

Listing 9-20 shows the content of the del_book.php file:

Listing 9-20: The del_book.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\" ><td colspan=\"2\" align=\"top\">";
$item_no=$_GET['tbl_id'];
?>
<table border="0" width="100%" height="100%" cellpadding="1" cellspacing="1"
bgcolor="#ffffff">
<tr bgcolor="#E8E8E8" height="4">
<td width="32" valign="middle" align="center"><font size="1"
face="verdana"><b>SNo.</b></font></td>
<td width="64" valign="middle" align="center"><font size="1"
face="verdana"><b>Item
No.</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Category</b></font></td>
<td width="85" valign="middle" align="center"><font size="1"
face="verdana"><b>Title</b></font></td>
<td width="68" valign="middle" align="center"><b><font size="1"
face="verdana">Author</font></b></td>
<td width="33" valign="middle" align="center"><b><font size="1"
face="verdana">Price($)</font></b></td>
</tr>
<?php
$j=0;
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result = mysql_query("select * from book where item_no='$item_no'");
```

```
while($row = mysql_fetch_array($result))
{
   $j++;
   print " <tr bgcolor=\"#E8E8E8\" height=\"4\">
   <td width=\"32\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$j</font></td>
   <td width=\"64\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[0]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[1]</font></td>
   <td width=\"85\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[2]</font></td>
   <td width=\"68\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[3]</font></td>
   <td width=\"33\" align=\"center\"><font face=\"Arial\"
   size=\"2\">$row[4]</font></td>
   </tr>";
   print " <tr bgcolor=\"#E8E8E8\" height=\"4\" >
   <td width=\"100%\" align=\"center\" colspan=\"6\"><font face=\"Arial\" size=\"2\"
   color=\"#FF0000\"><b>Do you want to delete this
   record.  </b>
   <a
   href=\"del_book1.php?item_no=$row[0]&item_type=$row[1]&title=$row[2]&p=$p&
   del=y\">Yes</a>   <a
   href=\"del_book1.php?item_no=$row[0]&item_type=$row[1]&p=$p&del=n\">
   No</a></font></td></tr>";
}
?>
<tr height="400" bgcolor="E8E8E8"><td colspan="6"> </td></tr>
</table>
</td>
<?php
include "right.php";
?>
```

The above listing creates a Web page that prompts the administrator before deleting a book from the database.

Figure 9-13 shows the Web page that lets the administrator delete a book from the shop database:

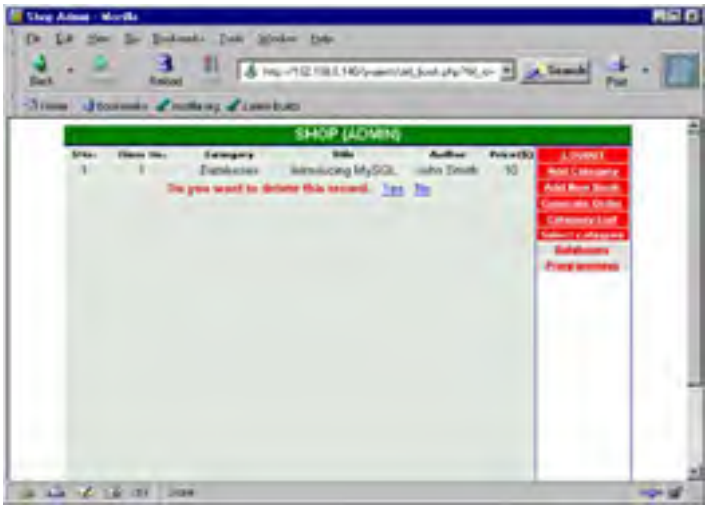


Figure 9-13: Deleting a Book

Listing 9-21 shows the content of the del_book1.php file:

Listing 9-21: The del_book1.php File

```
<?php
include "tophtml.php";
echo "<tr width=\"499\"><td align=\"center\" colspan=\"2\">";
$title=$_GET['title'];
$del=$_GET['del'];
$db=mysql_connect('localhost', 'root', '');
if(!$db)
{
   echo "Error when connecting to database";
}
mysql_select_db("shop", $db);
if ($del == 'y')
{
   mysql_query("delete from book where title='$title'", $db);
   print " Book Deleted !";
```

```
}else
print " Book is not deleted.";
echo "</td>";
include "right.php";
//print
"<script>location.replace(\"view_book.php?item_type=$item_type&p=$p\")</script
>";
?>
```

The above listing deletes a book from the database table if the administrator clicks the Yes hyperlink in Figure 9-13. If the administrator clicks the No hyperlink, the message, Book is not deleted, is displayed.The administrator is redirected to the Web page that displays the books for a selected category.

Working with the Online Shopping Cart Application

The Client section contains the Web pages that enables the end user as clients to perform activities, such as buying a book by adding it to the shopping cart, and performing the search operation based on author name or book title. All the pages in the Client section of the application are divided into three sections: top, right, and bottom. The top and bottom sections are same for all the Web pages of the Client section. The right section differs for each Web page.

The first Web page of the client section is represented by the welcome.php file. The following code shows the content of the welcome.php file:

```
<?php
echo "Welcome";
include "tophtml.php";
include "right.php";
include "bottomhtml.php";
?>
```

The above code shows that the welcome.php file includes three other files: tophtml.php, right.php, and bottomhtml.php.

Listing 9-22 shows the content of the tophtml.php file:

Listing 9-22: The tophtml.php File

```
<?php
$linkcol="#FF0000";
$linkcol1="#0000FF";
$bg="#7194D5";
$bg1="#4E6CA7";
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Book Shop</title>
<style type="text/css">
<!--
a {font: 10pt Arial, Verdana, Helvetica, sans-serif; text-decoration: none}
p {font: 12pt Arial, Helvetica, sans serif}
UL {font: 12pt Arial, Helvetica, sans serif}
-->
</style>
<script>
function usercheck()
{
   var user_id = document.newusr.user_id.value.replace(/\s+/,"");
   var passwd = document.newusr.passwd.value.replace(/\s+/,"");
   var cpasswd = document.newusr.cpasswd.value.replace(/\s+/,"");
   var name = document.newusr.name.value.replace(/\s+/,"");
   var address_line1 = document.newusr.address_line1.value.replace(/\s+/,"");
   //var address_line2 = document.newusr.address_line2.value.replace(/\s+/,"");
   var city = document.newusr.city.value.replace(/\s+/,"");
   var country = document.newusr.country.value.replace(/\s+/,"");
   var pin = document.newusr.pin.value.replace(/\s+/,"");
   var email_id = document.newusr.email_id.value.replace(/\s+/,"");
   var phone_number = document.newusr.phone_number.value.replace(/\s+/,"");
   var fax_number = document.newusr.fax_number.value.replace(/\s+/,"");
   var card_no = document.newusr.card_no.value.replace(/\s+/,"");
   var expiry_date = document.newusr.expiry_date.value.replace(/\s+/,"");
   var card_type = document.newusr.card_type.value.replace(/\s+/,"");
   var error="no";
   var message;
   var focus;
   if ((!user_id) && (error == "no"))
   {
      message = "Please enter User ID";
      focus="user_id";
      error="yes";
   }
   if ((!passwd) && (error == "no"))
   {
      message = "Please enter  Password";
      focus="passwd";
      error="yes";
   }
   if ((!cpasswd) && (error == "no"))
   {
      message = "Please enter Confirm Password";
      focus="cpasswd";
      error="yes";
   }
   if ((!name) && (error == "no"))
   {
      message = "Please enter  Name";
      focus="name";
```

```
      error="yes";
   }
   if ((!address_line1) && (error == "no"))
   {
      message = "Please enter  Address";
      focus="address_line1";
      error="yes";
   }
   if ((!city) && (error == "no"))
   {
      message = "Please enter  City";
      focus="city";
      error="yes";
   }
   if ((!country) && (error == "no"))
   {
      message = "Please enter  Country";
      focus="country";
      error="yes";
   }
   //PinCode Validation
   if ((!pin) && (error == "no"))
   {
      message = "Please enter  Pin Code";
      focus="pin";
      error="yes";
   }
   else
   {
   var valid ="0123456789";
   for (var i=0; i < pin.length; i++)
   {
      var temp = "" + pin.substring(i, i+1);
      if (valid.indexOf(temp) == "-1")
      {
         alert("Invalid characters in your pin code");
         focus="pin";
         error="yes";
         return false;
      }
   }
   if ((pin.length < 6) && (error == "no"))
   {
      alert("Invalid your pin code");
      focus="pin";
      error="yes";
   }
}
//validate phone number
if ((!phone_number) && (error == "no"))
{
   message = "Please enter  Phone Number";
   focus="phone_number";
   error="yes";
}
else
{
   var validph ="0123456789";
   for (var i=0; i < phone_number.length; i++)
   {
      var temp = "" + phone_number.substring(i, i+1);
      if (validph.indexOf(temp) == "-1")
      {
         alert("Invalid characters in your phone number.");
         focus="phone_number";
         error="yes";
         return false;
      }
   }
   if ((phone_number.length < 6) && (error == "no"))
   {
      alert("Invalid your phone number.");
      focus="phone_number";
      error="yes";
   }
}
if ((!email_id) && (error == "no"))
{
   message = "Please enter  Email ID";
   focus="email_id";
   error="yes";
}
//Card Number Validation
if ((!card_no) && (error == "no"))
{
   message = "Please enter  Card Number";
```

```
   focus="card_no";
   error="yes";
}
else
{
   var validcard ="0123456789";
   for (var i=0; i < card_no.length; i++)
   {
      var temp = "" + card_no.substring(i, i+1);
      if (validcard.indexOf(temp) == "-1")
      {
         alert("Invalid characters in your card number.");
         focus="card_no";
         error="yes";
         return false;
      }
   }
   if ((card_no.length < 6) && (error == "no"))
   {
      alert("Invalid your card number.");
      focus="card_no";
      error="yes";
   }
}
if ((passwd != cpasswd) && (error == "no"))
{
   message = "Check Confirm Password";
   focus="cpasswd";
   error="yes";
}
if (error == "yes")
{
   alert(message);
   var str ="document.newusr."+focus+".focus()";
   eval(str);
}
else
{
   document.newusr.submit();
}
</script>
<script language="JavaScript">
function MM_preloadImages()
{
   var d=document; if(d.images){ if(!d.MM_p) d.MM_p=new Array();
   var i,j=d.MM_p.length,a=MM_preloadImages.arguments; for(i=0; i<a.length; i++)
   if (a[i].indexOf("#")!=0){ d.MM_p[j]=new Image; d.MM_p[j++].src=a[i];}}
}
</script>
</HEAD>
<BODY BGCOLOR="#FFFFFF" onLoad="MM_preloadImages('/images/blank.gif')" >
<center>
<!-- begin table1-->
<TABLE BORDER="0" WIDTH="700" CELLPADDING="0" CELLSPACING="0" BGCOLOR="#FFFFFF" >
<tr>
<td>
<!-- begin table2-->
<table BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600" style="margin: 0px; padding: 0px">
<!--
<tr>
<td align="center">
<img src="/images/consbar.gif">
</td>
</tr>
-->
<tr>
<td HEIGHT="1" width="600" align="left" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1></td>
</tr>
<tr>
<td HEIGHT="1" width="600" align="left">
<table BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH="600">
<tr >
<td align="center" bgcolor="<?php if ($link == 1) echo $bg1; else echo $bg; ?>">
<A href="index.php?link=1"><font size="3" face="arial"
 color="#FFFFFF"><b> BOOK SHOP</b></font></A>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td HEIGHT="1" BGCOLOR="#1E237B" width="600" align="left" background="/images/line.gif">
```

```
<img SRC="/images/line.gif" ALT="-" BORDER=0 height=1 width=1></td>
</tr>
</table>
<!--end table2-->
</td>
</tr>
```

The above listing creates the top section of the home page of the online shopping cart application.

The middle section displays the options, such as searching a book, and listing the books from a category.

Listing 9-23 shows the content of the right.php file to create the middle section of the home page:

Listing 9-23: Creating the Middle Section of the Home Page

```
<td BGCOLOR="#FF0000" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="150" valign="top">
<table width="150" border="1" cellspacing="2" cellpadding="2">
<tr>
<td>
<table width="150" border="1" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<?php
$login=$_GET['login'];
if (($login == "ok") || ($sesid))
echo "<a href=\"../logout.php\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#33CC66\"><B>LOGOUT</B>";
else
echo "<a href=\"new.php\"><font style=\"font-size:11px;\" face=\"Helvetica\"
color=\"#FF0000\"><B>NEW USER</b></font></a><a
href=\"../index.php\"><font style=\"font-size:11x;\" face=\"Helvetica\"
color=\"#ffffff\"><B>LOGIN</B>";
?>
 </td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<form name="searchform" action="book_desc.php" method="GET"><BR>
<input type="hidden" name="req_from" value="search">
<input type="text" name="search_word" size="10"><BR>
<font style="font-size:12px;" face="Helvetica" color="#ffffff"><B>Search
by</B></font><BR>
<select name="search_by">
<option value="author">By Author</option>
<option value="title">By Title</option>
</select><BR><BR>
<input type="submit" value="Search">
</form>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<font style="font-size:12px;" face="Helvetica" color="#ffffff"><B>Select Book
Category</B></font></td>
</tr>
</table>
</td>
</tr>
<?php
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_cat = mysql_query("select item_type from category", $db);
print "<tr><td align=\"center\"><form name=\"form1\" method=\"get\"
action=\"book_desc.php\"><table width=\"110\" border=\"0\" cellspacing=\"1\"
cellpadding=\"1\"><tr><td align=\"center\"><SELECT NAME=\"item_type\">";
while ($row_cat=mysql_fetch_array($result_cat))
{
   print "<option value=\"$row_cat[0]\">$row_cat[0]</option>";
```

```
   //print "<tr><td bgcolor=\"#FF3300\" align=\"center\"><a
   href='book_desc.php?item_type=$row_cat[0]'><font style=\"font-size:12px;\"
   face=\"Helvetica\"
   color=\"#ffffff\">$row_cat[0]</font></a></td></tr>";
}
print "</SELECT></td></tr><tr><td align=\"center\"><input
type=\"submit\" value=\"BOOK LIST\"></td></tr>";
print "</table></form></td> </tr>";
?>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<font style="font-size:12px;" face="Helvetica" color="#ffffff"><B>Your
Cart</B></font></td>
</tr>
</table>
</td>
</tr>
<?php
$date = date('Y-m-d');
$result_cart = mysql_query("select book.title from tmp,book where tmp.user_id='$user' and
tmp.sesid='$sesid' and tmp.date='$date' and book.item_no=tmp.item_no", $db);
print "<tr><td align=\"center\"><table width=\"110\" border=\"0\"
cellspacing=\"1\" cellpadding=\"1\">";
while ($row_cart=mysql_fetch_array($result_cart))
{
   print "<tr><td align=\"center\"><font style=\"font-size:12px;\"
   face=\"Helvetica\"
   color=\"#000000\">$row_cart[0]</font></a></td></tr>";
}
print "</table></td> </tr>";
?>
<tr>
<td>
<table width="150" border="0" cellspacing="0" cellpadding="0">
<tr>
<td bgcolor="#7272B1" align="center">
<?php
if (($login == "ok") || ($sesid))
print "<a href=\"checkout.php\"><font style=\"font-size:12px;\" face=\"Helvetica\"
color=\"#ffffff\"><B>Check Out</B></font></a></td>";
?>
</tr>
</table>
</td>
</tr>
</table>
</td>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height="1" width="1">
</td>
```

The above listing creates the middle section of the home page for the Client section.

Listing 9-24 shows the bottom section of the home page created using the bottomhtml.php file:

Listing 9-24: The bottomhtml.php File

```
<tr>
<td HEIGHT="1" BGCOLOR="#1E237B" width="600" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1></td>
</tr>
</table>
<!-- end table3-->
</td>
</tr>
<tr><td>
<table border=0 width=90%>
<tr>
<td align="center" colspan="3"><font face="arial" size="1"
color="#1E237B"> </font></td>
</tr>
</table>
</td></tr>
</table>
</body>
</html>
```

The above listing creates the bottom section of the home page for the online shopping cart application.

Figure 9-14 shows the home page of the online shopping cart application:

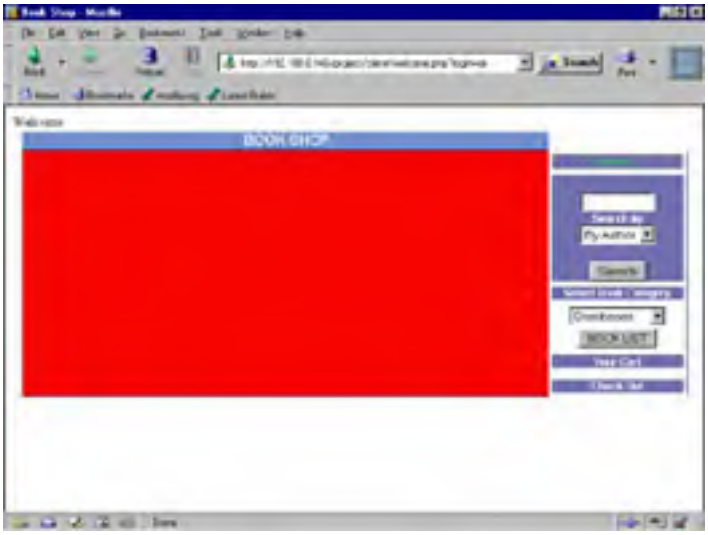


Figure 9-14: Home Page of the Client Section

Registering a New User

To initiate the registration process, end user needs to click the New User link, as shown in Figure 9-2. The new.php file is executed by the Web server that represents a Web page to accept data for a new end user.

Listing 9-25 shows the content of the new.php file:

Listing 9-25: The new.php File

```
<?php
include("tophtml.php");
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td>
<table width=100%>
<tr>
<td align="center">
<form name="newusr" action="new1.php" method="GET">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b>User Name *</b></font>
</td>
<td>
<input type="text" name="user_id" maxlength="20" value="<?php if ($user_id) echo
"$user_id"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b>Password *</b></font>
</td>
<td>
<input type="password" name="passwd" maxlength="10" value="<?php echo "$passwd";
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b>Confirm Password *</b></font>
</td>
<td>
<input type="password" name="cpasswd" maxlength="10" value="<?php if ($cpasswd) echo
"$cpasswd"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Name *</b></font>
</td>
<td>
<input type="text" name="name" maxlength="40" value="< ?php if ($name) echo "$name";
```

```
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Address1 *</b></font>
</td>
<td>
<input type="text" name="address_line1"  value="<?php if ($address_line1) echo
"$address_line1"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Address2</b></font>
</td>
<td>
<input type="text" name="address_line2" value="< ?php if ($address_line2) echo
"$address_line2";?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> City *</b></font>
</td>
<td>
<input type="text" name="city" value="<?php if ($city) echo "$city"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Country *</b></font>
</td>
<td>
<input type="text" name="country"  value="<?php if ($country) echo "$country"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Pin Code *</b></font>
</td>
<td>
<input type="text" name="pin"  value="<?php if ($pin) echo "$pin"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Email ID *</b></font>
</td>
<td>
<input type="text" name="email_id"  value="<?php if ($email_id) echo "$email_id";
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Phone Number *</b></font>
</td>
<td>
<input type="text" name="phone_number"  value="<?php if ($phone_number) echo
"$phone_number";?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Fax Number </b></font>
</td>
<td>
<input type="text" name="fax_number"  value="<?php if ($fax_number) echo "$fax_number";
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Card Number *</b></font>
</td>
<td>
<input type="text" name="card_no"  value="<?php if ($card_no) echo "$card_no"; ?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Expiry Date</b></font>
</td>
<td>
<input type="text" name="expiry_date"  value="<?php if ($expiry_date) echo "$expiry_date";
```

```
?>">
</td>
</tr>
<tr>
<td valign="top" align="right">
<font face="Verdana" size="1"><b> Card Type</b></font>
</td>
<td>
<select name="card_type">
<option value="American Express">American Express</option>
<option value="Master Card">Master Card</option>
<option value="Visa">Visa</option>
</select>
<!--
<input type="text" name="card_type" value="<?php if ($card_type) echo "$card_type";
?>">
-->
</td>
</tr>
<tr>
<td colspan="2" align="center"><input type="submit" value="Submit"></td>
</tr>
</table>
</form>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
<?php
include("bottumhtml.php");
?>
```

The above listing creates a Web page to accept information to register a new end user, as shown in Figure 9-15:

Figure 9-15: Web Page to Register New End User

When the end user clicks the Submit button, the new1.php file is processed to add the end user information to the database table.

Listing 9-26 show the content of the new1.php file:

Listing 9-26: The new1.php File

```
<?php
settype($error,"string");
settype($order_list,"string");
settype($user,"string");
settype($sesid,"string");
$name=$_GET['user'];
$user_id=$_GET['user_id'];
$passwd=$_GET['passwd'];
$address_line1=$_GET['address_line1'];
$address_line2=$_GET['address_line2'];
$city=$_GET['city'];
$country=$_GET['country'];
$pin=$_GET['pin'];
$email_id=$_GET['email_id'];
$phone_number=$_GET['phone_number'];
$card_no=$_GET['card_no'];
$expiry_date=$_GET['expiry_date'];
$card_type=$_GET['card_type'];
$fax_number=$_GET['fax_number'];
/*
echo "value ". $name;
echo $user_id;
echo $passwd;
echo $address_line1;
echo $address_line2;
echo $city;
echo $country;
echo $pin;
echo $email_id;
echo $phone_number;
echo $card_no;
echo $expiry_date;
echo $card_type;
echo $fax_number;
*/
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$query = mysql_query("insert into
user_profile(name,user_id,password,address_line1,address_line2,city,country,pin,email_id,
phone_number,card_no,expiry_date,card_type,fax_number)
values('$name','$user_id','$passwd','$address_line1','$address_line2','$city','$country',
'$pin','$email_id','$phone_number','$card_no','$expiry_date','$card_type','$fax_number')",
 $db) or ($error1=mysql_errno());
$num = mysql_affected_rows();
if ($num < 1)
$message="Agent ".$agentid." already exist.";
else
{
   $user=$user_id;
   session_save_path("/tmp");
   session_start();
   $sesid=session_id();
   session_register("sesid");
   session_register("order_list");
   session_register("user");
   $message = "You have registered successfully.";
}
include("tophtml.php");
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td>
<table width=100%>
<tr>
<td align="center">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td valign="top" align="center">
<font face="Verdana" size="2" color="#FF0000"><b><?php echo "$message"
```

```
?></b></font>
</td>
</tr>
<?php
if ($error1)
{
   print "
   <tr>
   <td valign=\"top\" align=\"center\">
   <form name=\"form1\" action=\"new.php\" method=\"post\">
   <input type=\"hidden\" name=\"name\" value=\"$name\">
   <input type=\"hidden\" name=\"user_id\" value=\"$user_id\">
   <input type=\"hidden\" name=\"address_line1\" value=\"$address_line1\">
   <input type=\"hidden\" name=\"address_line2\" value=\"$address_line2\">
   <input type=\"hidden\" name=\"city\" value=\"$city\">
   <input type=\"hidden\" name=\"country\" value=\"$country\">
   <input type=\"hidden\" name=\"pin\" value=\"$pin\"><input type=\"hidden\"
   name=\"email_id\" value=\"$email_id\">
   <input type=\"hidden\" name=\"phone_number\" value=\"$phone_number\">
   <input type=\"hidden\" name=\"fax_number\" value=\"$fax_number\">
   <input type=\"hidden\" name=\"card_no\" value=\"$card_no\">
   <input type=\"hidden\" name=\"expiry_date\" value=\"$expiry_date\">
   <input type=\"hidden\" name=\"card_type\" value=\"$card_type\">
   <input type=\"submit\" value=\"Try Again\">
   </td>
   </tr>";
}
?>
</table>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing inserts a new record in the user_profile database table. If the new end user is registered successfully, the message, You have registered successfully, is displayed to the end user; else, an error message is displayed.

Viewing Books Stored in a Category

When the end user clicks the BOOK LIST button after selecting a book category from the Select Book Category combo box, as shown in Figure 9-14, the book_desc.php file is processed. This file displays all the books present in the specified category.

Listing 9-27 shows the content of the book_desc.php file:

Listing 9-27: Displaying Books in a Specified Category

```
<?php
session_save_path("/tmp");
session_start();
include("tophtml.php");
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
```

```
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$req_from=$_GET['req_from'];
$title=$_GET['title'];
$search_by=$_GET['search_by'];
$search_word=$_GET['search_word'];
$item_type=$_GET['item_type'];
if ($req_from == "search")
{
   if ($search_by == "author")
   {
      $result1=mysql_query("select * from book where author like '%$search_word%'", $db);
   }
   elseif ($search_by == "title")
   {
      $result1=mysql_query("select * from book where title like '%$search_word%'", $db);
   }
}
else
{
   $result1=mysql_query("select * from book where item_type='$item_type'", $db);
}
$num = mysql_num_rows($result1);
if ($num > 0)
{
   while($row1=mysql_fetch_array($result1))
   {
      print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
      face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[0]</b></font></td><td
      align=\"center\"
      width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[1]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[2]</b></font></td><td
      align=\"center\"
      width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[3]</b></font></td><td
      align=\"center\"
      width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[4]</b></font></td><td
      align=\"center\"
      width=\"20%\"><form name=\"cart\" action=\"add_cart.php\"
      action=\"get\"><input
      type=\"hidden\" name=\"item_type\" value=\"$item_type\"><input type=\"hidden\"
      name=\"item_no\" value=\"$row1[0]\"><input type=\"hidden\" name=\"req_from\"
      value=\"$req_from\"><input type=\"hidden\" name=\"search_by\"
      value=\"$search_by\"><input type=\"hidden\" name=\"search_word\"
      value=\"$search_word\"><input type=\"submit\" value=\"ADD TO
      CART\"></form></td></tr>";
   }
}
else
{
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"
   colspan=\"5\"><font face=\"verdana\" size=1 color=\"#FF0000\"><b>No match for
   the $search_word found.</b></font></td></tr>";
}
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
```

The above listing retrieves the information about books for a specific category.

Figure 9-16 shows the Web page that displays the books present in the Database category:

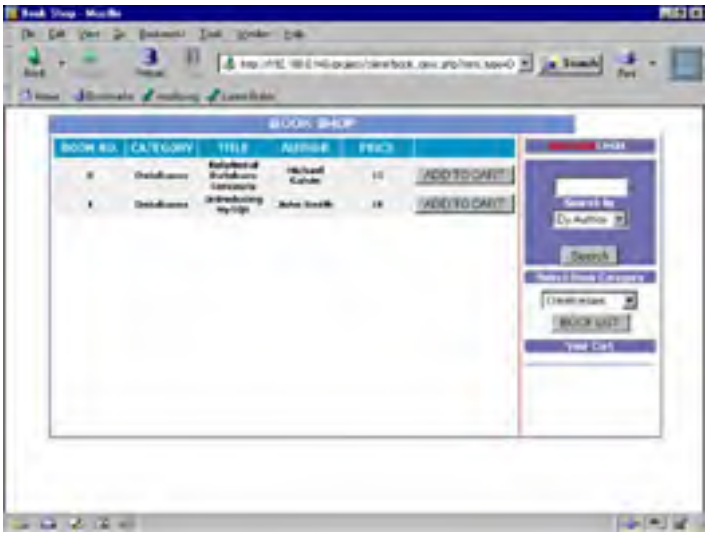


Figure 9-16: Books in the Database Category

Placing a Book in the Shopping Cart

When the end user clicks the ADD TO CART button, shown in Figure 9-16, the add_cart.php file is processed to add the selected book to the cart.

Listing 9-28 shows the content of the add_cart.php file:

Listing 9-28: The add_cart.php File

```
<?php
//include "sessioncheck.php";
session_start();
include("tophtml.php");
$date = date('Y-m-d');
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$search_by=$_GET['search_by'];
$search_word=$_GET['search_word'];
$item_type=$_GET['item_type'];
$item_no=$_GET['item_no'];
if ($user)
{
   if (!$resulr=mysql_query("insert into tmp
   values('NULL','$user','$item_no','$sesid','$date')", $db))
   {
      $e=mysql_error();
      echo "$e";
   }
}
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
```

```
color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
if ($req_from == "search")
{
   if ($search_by == "author")
   {
      $result1=mysql_query("select * from book where author like '%$search_word%'", $db);
   }
   elseif ($search_by == "title")
   {
      $result1=mysql_query("select * from book where title like '%$search_word%'", $db);
   }
}
else
{
   $result1=mysql_query("select * from book where item_type='$item_type'", $db);
}
while($row1=mysql_fetch_array($result1))
{
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
   face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[0]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[1]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[2]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[3]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row1[4]</b></font></td><td
   align=\"center\" width=\"20%\"><form name=\"cart\" action=\"add_cart.php\"
   action=\"post\"><input type=\"hidden\" name=\"item_type\"
   value=\"$item_type\"><input type=\"hidden\" name=\"item_no\"
   value=\"$row1[0]\"><input type=\"hidden\" name=\"req_from\"
   value=\"$req_from\"><input type=\"hidden\" name=\"search_by\"
   value=\"$search_by\"><input type=\"hidden\" name=\"search_word\"
   value=\"$search_word\"><input type=\"submit\" value=\"ADD TO
   CART\"></form></td></tr>";
}
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing adds a book to the shopping cart.

Figure 9-17 shows the output when the book, Relational Database Concepts, is added to the cart:

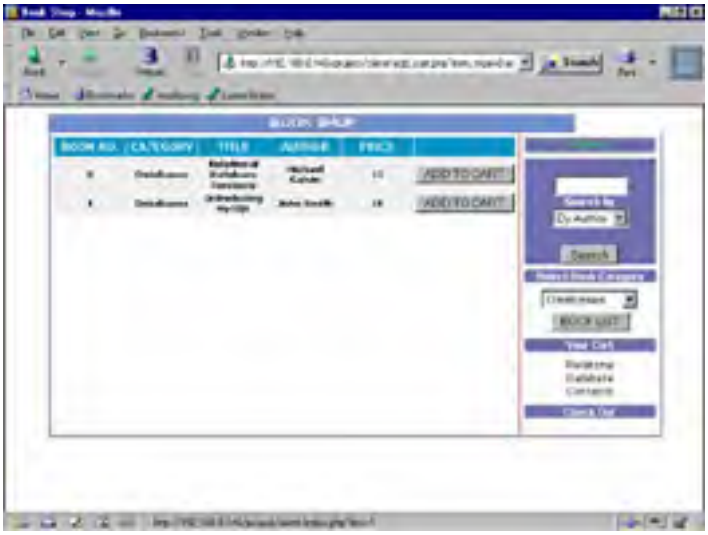


Figure 9-17: Adding a Book to the Cart

Searching for a Book

The end user can search for a book either by its title or the author name. To search a book by title, enter the title in the Search By text field, as shown in Figure 9-17, and click the Search button. The book_desc.php file is executed to retrieve the books from the database that match the specified title.

Listing 9-29 shows the content of the book_desc.php file:

Listing 9-29: Retrieving Books matching a Specified Title

```
<?php
session_save_path("/tmp");
session_start();
include("tophtml.php");
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$req_from=$_GET['req_from'];
$title=$_GET['title'];
$search_by=$_GET['search_by'];
$search_word=$_GET['search_word'];
$item_type=$_GET['item_type'];
if ($req_from == "search")
{
   if ($search_by == "author")
   {
      $result1=mysql_query("select * from book where author like '%$search_word%'", $db);
   }
   elseif ($search_by == "title")
   {
      $result1=mysql_query("select * from book where title like '%$search_word%'", $db);
```

```
    }
}
else
{
   $result1=mysql_query("select * from book where item_type='$item_type'", $db);
}
$num = mysql_num_rows($result1);
if ($num > 0)
{
   while($row1=mysql_fetch_array($result1))
   {
      print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
      face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[0]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[1]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[2]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[3]</b></font></td><td
      align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
      color=\"#000000\"><b>$row1[4]</b></font></td><td
      align=\"center\" width=\"20%\"><form name=\"cart\" action=\"add_cart.php\"
      action=\"get\"><input type=\"hidden\" name=\"item_type\"
      value=\"$item_type\"><input type=\"hidden\" name=\"item_no\"
      value=\"$row1[0]\"><input type=\"hidden\" name=\"req_from\"
      value=\"$req_from\"><input type=\"hidden\" name=\"search_by\"
      value=\"$search_by\"><input type=\"hidden\" name=\"search_word\"
      value=\"$search_word\"><input type=\"submit\" value=\"ADD TO
      CART\"></form></td></tr>";
   }
}
else
{
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"
   colspan=\"5\"><font face=\"verdana\" size=1 color=\"#FF0000\"><b>No match for
   the $search_word found.</b></font></td></tr>";
}
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

Figure 9-18 shows the output when search is performed on the basis of the author name, Galvin:

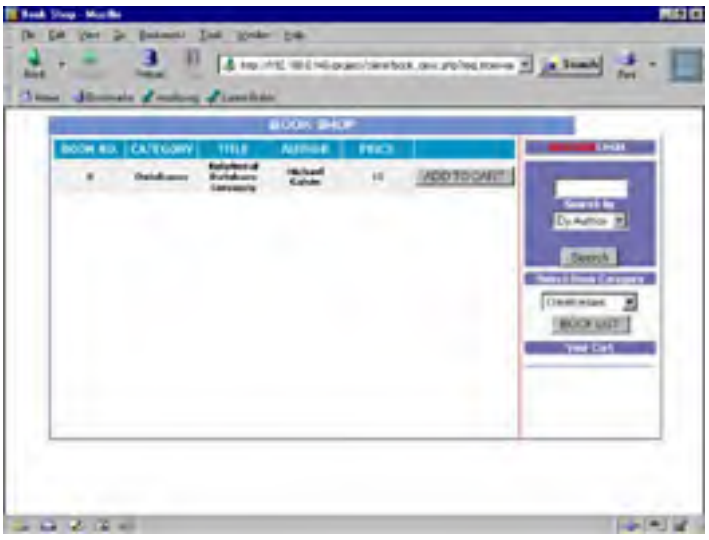


Figure 9-18: Searching Books Based on Author Name

Viewing Items Placed in a Shopping Cart

When the end user clicks the Check Out hyperlink present in the right section of the welcome.php Web page, the checkout.php file is executed. The checkout.php Web page creates a Web page that shows the books added to the shopping cart, as shown in Listing 9-30:

Listing 9-30: The checkout.php File

```
<?php
session_start();
include("tophtml.php");
$date = date('Y-m-d');
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>SNO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$sn=0;
$total=0;
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result = mysql_query("select book.*,tmp.order_no from tmp,book where tmp.user_id='$user' and
tmp.sesid='$sesid' and tmp.date='$date' and book.item_no=tmp.item_no", $db);
while($row_chack=mysql_fetch_array($result))
{
   $total=$total+$row_chack[4];
   $sn++;
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
   face=\"verdana\" size=1
   color=\"#000000\"><b>$sn</b></font></td><td align=\"center\"
   width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[0]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[1]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[2]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[3]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[4]</b></font></td><td
   align=\"center\" width=\"20%\"><form name=\"cart\" action=\"del_cart.php\"
   action=\"post\"><input type=\"submit\" value=\"DELETE\"><input type=\"hidden\"
   name=\"order_no\" value='$row_chack[5]'></form></td></tr>";
}
print "<tr bgcolor=\"#F2F2F2\"><td align=\"right\" width=\"100%\"
colspan=\"2\"><form name=\"order\" action=\"final_order.php\" method=\"post\"><input
type=\"submit\" value=\"POST YOUR ORDER\"></td><td align=\"right\" width=\"100%\"
colspan=\"2\"><font face=\"verdana\" size=1 color=\"#000000\"><b>TOTAL BOOKS:
  $sn</b></font></td><td align=\"right\" width=\"100%\"
```

```
colspan=\"2\"><font face=\"verdana\" size=1 color=\"#000000\"><b>TOTAL :
  </b></font></td><td align=\"left\" width=\"100%\"><font
face=\"verdana\" size=1
color=\"#000000\"><b>$total</b></font></td></td></tr>";
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing displays the titles of the books that are added to the shopping cart in a Web page, as shown in Figure 9-19 :

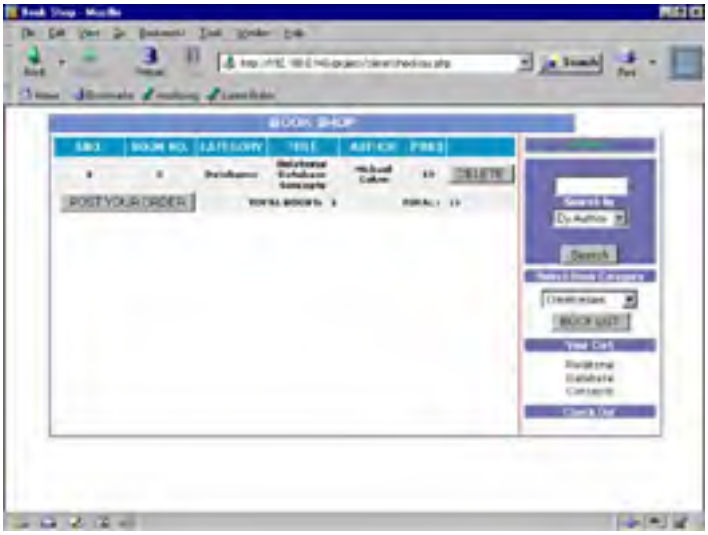


Figure 9-19: Displaying Books Added to the Shopping Cart

Removing a Book From the Shopping Cart

When the end user clicks the Delete button corresponding to a book name, as shown in Figure 9-19, the del_cart.php file is executed to remove the selected book from the shopping cart.

Listing 9-31 shows the content of the del_cart.php file:

Listing 9-31: The del_cart.php File

```
<?php
session_start();
include("tophtml.php");
$date = date('Y-m-d');
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$order_no=$_GET["order_no"];
$db=mysql_connect('localhost', 'root', '');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$resul_del=mysql_query("delete from tmp where order_no=$order_no", $db);
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
```

```
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="20%"><font face="Arial" size=2
color="#ffffff"><b>SNO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>BOOK NO.</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>CATEGORY</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>TITLE</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>AUTHOR</b></font></td>
<td align="center" width="20%"><font face="Arial" size=2 color="#ffffff"><b>PRICE</b></font></td>
<td align="center" width="20%"> </td>
</tr>
<?php
$sn=0;
$total=0;
$result_chack = mysql_query("select book.*,tmp.order_no from tmp,book where tmp.user_id='$user'
and tmp.sesid='$sesid' and tmp.date='$date' and book.item_no=tmp.item_no", $db);
while($row_chack=mysql_fetch_array($result_chack))
{
   $sn++;
   $total=$total+$row_chack[4];
   print "<tr bgcolor=\"#F2F2F2\"><td align=\"center\" width=\"20%\"><font
   face=\"verdana\" size=1
   color=\"#000000\"><b>$sn</b></font></td><td align=\"center\"
   width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[0]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[1]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[2]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[3]</b></font></td><td
   align=\"center\" width=\"20%\"><font face=\"verdana\" size=1
   color=\"#000000\"><b>$row_chack[4]</b></font></td><td
   align=\"center\" width=\"20%\"><form name=\"cart\" action=\"del_cart.php\"
   action=\"post\"><input type=\"submit\" value=\"DELETE\"><input type=\"hidden\"
   name=\"order_no\" value='$row_chack[5]'></form></td></tr>";
}
print "<tr bgcolor=\"#F2F2F2\"><td align=\"right\" width=\"100%\"
colspan=\"2\"><form name=\"order\" action=\"final_order.php\"
method=\"post\"><input type=\"submit\" value=\"POST YOUR ORDER\"></td><td
align=\"right\" width=\"100%\" colspan=\"2\"><font face=\"verdana\" size=1
color=\"#000000\"><b>TOTAL BOOKS:   $j</b></font></td><td
align=\"right\" width=\"100%\" colspan=\"2\"><font face=\"verdana\" size=1
color=\"#000000\"><b>TOTAL :   </b></font></td><td
align=\"left\" width=\"100%\"><font face=\"verdana\" size=1
color=\"#000000\"><b>$total</b></font></td></td></tr>";
?>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing removes the selected book from the shopping cart.

Confirming an Order

When the end user clicks the POST YOUR ORDER button, as shown in Figure 9-19, the final_order.php file is executed to confirm the order.

Listing 9-32 shows the content of the final_order.php file:

Listing 9-32: The final_order.php File

```
<?php
session_start();
include("tophtml.php");
$date = date('Y-m-d');
$user=$_SESSION["user"];
$sesid=$_SESSION["sesid"];
$order_no=$_GET["order_no"];
$db=mysql_connect('localhost','root','');
if (!$db)
{
   echo "Error When connecting to Database";
}
mysql_select_db("shop", $db);
$result_con=mysql_query("select user_id,item_no,date from tmp where sesid='$sesid' and
user_id='$user' and date='$date'", $db);
$tt=mysql_num_rows($result_con);
if ($tt > 0)
{
   $result_ins=mysql_query("insert into transaction(order_no,user_id,date,status)
   values('NULL','$user','$date','Pending')", $db);
   $result_order=mysql_query("select max(order_no) from transaction where user_id='$user' and
   date='$date'", $db);
   $row_no = mysql_fetch_array($result_order);
   $order_no = $row_no[0];
   while($row_con=mysql_fetch_array($result_con))
   {
      $result11 = mysql_query("insert into order1
      values('NULL','$order_no','$row_con[1]','$row_con[0]')", $db);
   }
   $result_del=mysql_query("delete from tmp where sesid='$sesid' and user_id='$user' and
   date='$date'", $db);
}
?>
<tr>
<td valign="top">
<table border="0" width="700" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif">
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="1" width="100%">
<tr>
<td>
<table width="100%" cellspacing="1" bgcolor="#FFFFFF">
<tr bgcolor="#0099CC">
<td align="center" width="100%"><font face="Arial" size=2 color="#000000"><b>
<?php
if ($order_no)
echo "Your order has been posted successfully. Your Order No. is $order_no";
else
echo "Please select books for order.";
?>
</b></font></td>
</tr>
</table>
</td>
</tr>
<tr>
<td>
</td>
</tr>
</table>
</td>
<?php
include("right.php");
?>
</tr>
</table>
</td>
</tr>
<?php
include("bottomhtml.php");
?>
```

The above listing confirms the order for the books added to the shopping cart.

Figure 9-20 shows the Web page that displays the order confirmation message to the end user:

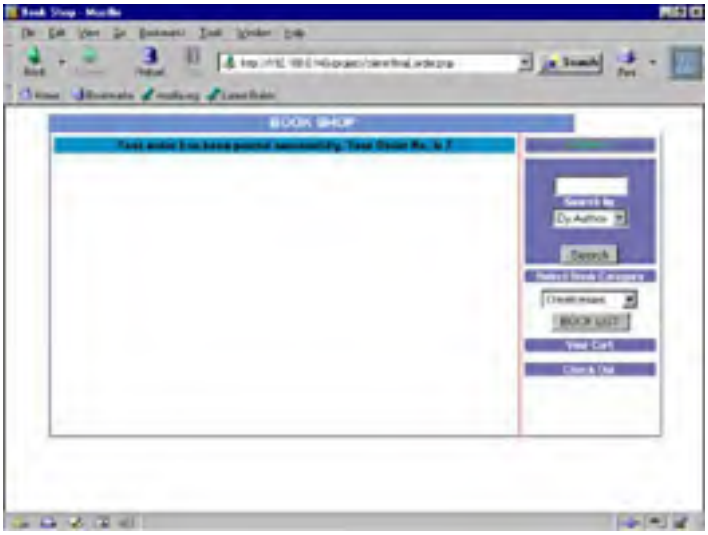


Figure 9-20: Web Page to Confirm the Order

Logging Off from the Shopping Cart Application

When the end user clicks the LOGOUT hyperlink present in the right section of welcome.php Web page, the logout.php file is executed, which logs the end user out of the application.

Listing 9-33 shows the content of the logout.php file:

Listing 9-33: The logout.php File for the End User

```
<?php
session_save_path("/tmp");
session_start();
$sesid=session_id();
session_unregister("sesid");
session_unregister("order_list");
session_unregister("user");
session_destroy();
$sesid="";
include("tophtml.php");
?>
<tr>
<td valign="top">
<table border="0" width="600" cellspacing="1" cellpadding="0" height="345">
<tr>
<td BGCOLOR="#1E237B" width="1" background="/images/spacer.gif"
<img SRC="/images/spacer.gif" ALT="-" BORDER=0 height=1 width=1>
</td>
<td width="540" valign="top">
<table border="0" cellpadding="2" cellspacing="2" width="100%">
<tr>
<td>
<table width=100%>
<tr>
<td align="center"><font face="arial" size=3 color="#FF0000"><b>
Logout successfully.
 </b></font></td>
</tr>
<?php
include "bottumhtml.php";
?>
```

The above listing logs out the end user from the online shopping cart application.

PREVIOUS NEXT

Appendix A: XML and XSLT Functions

This appendix describes various XML and XSLT functions.

XML Functions

Extensible Markup Language (XML) is a standard defined by the World Wide Web consortium (W3C) to represent data in a structured format.

You can use the following functions in Hypertext Preprocessor (PHP) to parse the XML documents:

- `utf8_decode()`: Transforms a string encoded in the UTF-8 format to the single-byte ISO-8859-1encoding format. The syntax of the utf8_decode() function is:

```
string utf8_decode (string data)
```

- `utf8_encode()`: Encodes a string with ISO-8859-1 encoding format to the UTF-8 format. The syntax of the utf8_encode() function is:

```
string utf8_encode (string data)
```

- `xml_error_string()`: Returns an XML parser string with a description of the error code, or returns the value, False, if no description is found. The syntax of the xml_error_string() function is:

```
string xml_error_string (int error_code)
```

- `xml_get_current_byte_index()`: Returns the current index of an XML parser. The syntax of the xml_get_current_byte_index() function is:

```
int xml_get_current_byte_index (parser)
```

In the above syntax, the xml_get_current_byte_index() function returns the value, False, if the specified parser is not a valid parser; or else, returns the current index of an XML parser.

- `xml_get_current_column_number()`: Returns the column number the parser is currently at. The syntax of the xml_get_current_column_number() function is:

```
int xml_get_current_column_number (parser)
```

- `xml_get_current_line_number()`: Returns the value of current line number the parser is currently at, in its data buffer. The syntax of the xml_get_current_line_number() function is:

```
int xml_get_current_line_number (parser)
```

- `xml_parse_into_struct()`: Parses the XML data into an array structure. The syntax of the xml_parse_into_struct() function is:

```
int xml_parse_into_struct (parser, string data, array &values [, array &index])
```

- `xml_parse()`: Parses an XML document. The syntax of the xml_parse() function is:

```
bool xml_parse (parserName, string data [, bool final])
```

- `xml_parser_create_ns()`: Creates a XML parser with XML namespace support. The syntax of the xml_parser_create_ns() function is:

```
resource xml_parser_create_ns ([string encoding [, string separator]])
```

- `xml_parser_create()`: Creates an XML parser. The syntax of the xml_parser_create() function is:

```
resource xml_parser_create ([string encoding])
```

- `xml_parser_free()`: Frees an XML parser and returns the value, False, if the parser is invalid; else, frees the specified parser and returns the value, True. The syntax of the xml_parser_free() function is:

```
bool xml_parser_free (resource parser)
```

- `xml_parser_get_option()`: Returns the value, False, if the specified parser is not a valid parser; else, returns the value of the option that is passed as argument to the xml_parser_get_option() function. The syntax of the xml_parser_get_option() function is:

```
mixed xml_parser_get_option (resource parser, int option)
```

- `xml_set_character_data_handler()`: Sets the character data handler for the specified XML parser. The syntax of the xml_set_character_data_handler() function is:

```
bool xml_set_character_data_handler (resource parser, callback handler)
```

- `xml_set_default_handler()`: Sets the default handler for the specified parser. The syntax of the xml_set_default_handler() is:

```
bool xml_set_default_handler (resource parser, callback handler)
```

- `xml_set_object()`: Lets you use the XML parser within an object. The syntax of the xml_set_object() function is:

```
void xml_set_object (resource parser, object object)
```

- `xml_set_processing_instruction_handler()`: Sets the processing instruction handler for the specified parser. The syntax of the xml_set_processing_instruction_handler() function is:

```
bool xml_set_processing_instruction_handler (resource parser, callback handler)
```

- `xml_set_start_namespace_decl_handler()`: Sets the start namespace declaration handler for the specified parser. The syntax of the xml_set_start_namespace_decl_handler() function is:

```
bool xml_set_start_namespace_decl_handler (resource parser, callback handler)
```

- `xml_set_unparsed_entity_decl_handler()`: Sets the unparsed entity declaration handler for the specified parser. The syntax of the xml_set_unparsed_entity_decl_handler() function is:

```
bool xml_set_unparsed_entity_decl_handler (resource parser, callback handler)
```

XSLT Functions

You use eXtensible Stylesheet Language Transformations (XSLT) to restructure or transform XML data into various formats, such as HTML. You can format XML data using PHP, in accordance with the template provided by XSLT.

The PHP extension provides a processor-independent API to perform XSLT transformations, and provides an interface to various XSLT processors, such as Sablotron and Xalan. PHP 4.1.1, the XSLT extension, supports only the Sablotron processor.

You can download Sablotron processor from http://www.gingerall.com/

The functions provided by the PHP extension to perform XSLT transformations are:

- `xslt_backend_info()`: Returns a string that contains information about the backend compilation settings. If no backend information is available, the xslt_backend_info() function returns an error string. The syntax of the xslt_backend_info() function is:

```
string xslt_backend_info (void)
```

- `xslt_backend_name()`: Returns the name of the backend processor. The syntax of the xslt_backend_name() function is:

```
string xslt_backend_name (void)
```

- `xslt_backend_version()`: Returns the version number of Sablotron. The syntax of the xslt_backend_version() function is:

```
string xslt_backend_version (void)
```

- `xslt_create()`: Creates a new XSLT processor. The syntax of the xslt_create() function is:

```
resource xslt_create (void)
```

- `xslt_errno()`: Returns an error number. The syntax of the xslt_errno() function is:

```
int xslt_errno (resource)
```

- `xslt_error()`: Returns an error message that describes the error generated while processing the XSLT processor. The syntax of the xslt_error() function is:

```
mixed xslt_error (resource)
```

- `xslt_process()`: Performs the XSLT transformation. The syntax of the xslt_process() function is:

```
mixed  xslt_process (resource , string xml_container, string xsl_container [, string
result_container [, array arguments [, array parameters]]])
```

In the above syntax, the containers refer to the filename containing the document to be processed. The result _container string refers to the filename for a transformed document.

- `xslt_set_base()`: Sets the base Uniform Resource Identifier (URI) for XSLT transformations. The syntax of the xslt_set_base() function is:

```
void xslt_set_base (resource x, string uri)
```

- `xslt_set_encoding()`: Sets the encoding for the parsing of XML documents. The xslt_set_encoding() function is used only when you compile the Sablotron processor as backend, with encoding support. The syntax of the xslt_set_encoding() function is:

```
void xslt_set_encoding (resource x, string encoding)
```

- `xslt_set_error_handler()`: Sets an error handler function for an XSLT processor. The syntax of the xslt_set_errror_handler () function is:

```
void xslt_set_error_handler (resource x, mixed handler)
```

- `xslt_set_log()`: Creates a file that stores the log messages. The log messages store information about the status of the XSLT processor. The syntax of the xslt_set_log() function is:

```
void xslt_set_log (resource x, mixed logparam)
```

In the above syntax, x is a variable that refers to the XSLT parameter, and logparam is a parameter that toggles logging on and off. You need to call the xslt_set_log() function with a Boolean parameter to enable message logging, as logging is disabled by default.

- `xslt_set_sax_handler()`: Sets the SAX handlers for the XSLT processor. The syntax of the xslt_set_sax_handler() function is:

```
void xslt_set_sax_handler ( resource x, array handlers)
```

- `xslt_set_scheme_handler()`: Sets the scheme handlers for an XSLT processor. The syntax of the xslt_set_scheme_handler () function is:

```
void xslt_set_scheme_handlers ( resource processor, array handlers)
```

Appendix B: Introducing eZXML

The eZXML class is an eXtensible Markup Language (XML) Document Object Model (DOM) parser written in Hypertext Preprocessor (PHP) language. The eZXML class provides an alternative approach to DOM implementation in PHP and lets you represent XML document in the form of nested objects. You do not require external libraries to support the eZXML class, because it is compatible with the libXml library.

The eZXML class generates a DOM tree after parsing the XML document, and generates an error message on encountering an incorrect XML document. The eZXML class creates a series of objects containing information about the XML nodes. Each object contains standard properties that enable you to traverse the DOM tree and access its attributes. You can develop PHP applications using the standalone libraries of the eZXML package.

This appendix describes how to create DOM tree of an XML document and how to convert an XML document into HTML using eZXML.

Working with eZXML

eZXML parses XML documents and creates a DOM tree representation from the XML document. eZXML creates objects to represent the XML document. Each object encapsulates properties that help traverse the DOM tree and access its elements and attributes. The elements of the eZXML class are defined in the eZXML.php file.

You need to include the eZXML.php file in the PHP script to parse the XML documents. eZXML.php is a procedural file that is defined in the eZXML class. The eZXML.php file defines the following members:

- NamespaceStack: Contains the namespaces.
- CurrentNameSpace: Specifies the current namespace.
- DomDocument: Specifies a reference to the DOM document.
- & eZXML::domTree ($xmlDoc, $params = array()): Returns a DOM object tree from the XML document.
- &eZXML::parseAttribute($attibuteString): Parses the attributes of the DOM tree and returns the value, False, if the string passed as argument does not contain any attributes.

Creating the DOM Tree of an XML Document using eZXML

The eZXML class contains the domTree() method that accepts XML strings as arguments and returns the DOM tree representation of an XML document. The object, which the domTree() method returns, contains information corresponding to the nodes of the XML document. You can use the information stored in the objects to create, modify, and format the XML document.

Listing B-1 shows how to create employee.xml file:

Listing B-1: Creating the employee.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<EMPLOYEEINFORMATION>
<EMPLOYEE>
<NAME ID="E001">George</NAME>
<AGE>35</AGE>
<DEPARTMENT>RESEARCH AND DEVELOPMENT</DEPARTMENT>
<DESIGNATION>BRANCH MANAGER</DESIGNATION>
</EMPLOYEE>
<EMPLOYEE>
<NAME ID="E002">John</NAME>
<AGE>45</AGE>
<DEPARTMENT>HUMAN RESOURCE</DEPARTMENT>
<DESIGNATION>MANAGER</DESIGNATION>
</EMPLOYEE>
</EMPLOYEEINFORMATION>
```

The above listing creates the employee.xml file that stores the employee information, such as name, ID, department, and designation. You can create the DOM object tree representation of the employee.xml file using eZXML, as shown in Listing B-2:

Listing B-2: Creating Object Representation of the employee.xml File

```
<?php
// include class definition
include_once( "./ezpublish-3.4.0/lib/ezutils/classes/ezdebug.php" );
include_once( "./ezpublish-3.4.0/lib/eZXML/classes/ezdomnode.php" );
include_once( "./ezpublish-3.4.0/lib/eZXML/classes/ezdomdocument.php" );
include_once( "./ezpublish-3.4.0/lib/eZXML/classes/eZXML.php" );
// XML file
$xmlFile = "employee.xml";
// parse XML file into single string
$xmlString = join("", file($xmlFile));
// create Document object
$doc = eZXML::domTree($xmlString, array("TrimWhiteSpace"=> "true"));
print_r($doc);
?>
```

The above listing creates a DOM tree using the eZXML class. The eZXML.php file is included in the script to parse the well-formed XML document, employee.xml. In the above listing, the object created by eZXML contains the node objects, which store information corresponding to the XML nodes.

Converting an XML Document into HTML using eZXML

eZXML converts XML documents into the HTML format. For example, you can create a file called customer.xml and convert into an HTML format.

Listing B-3 shows how to create the customer.xml file:

Listing B-3: Creating the customer.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<Shopping>
<customerInfo>
<custName>John</custName>
<address>1600 Pennsylvania Ave. NW, Washington, DC 20500</address>
<date>2001-09-15</date>
</customerInfo>
<items>
<item custid="3225">
<product>Air Conditioners</product>
<price>16235.00</price>
<quantity>1</quantity>
<subtotal>262235.00</subtotal>
</item>
<item custid="3226">
<product>Dryers</product>
<price>129.99</price>
<quantity>4</quantity>
<subtotal>139.96</subtotal>
</item>
<item custid="2345">
<product>Vacuum Cleaners</product>
<price>4139.99</price>
<quantity>2</quantity>
<subtotal>4139.99</subtotal>
</item>
<item custid="4576">
<product>Washers></product>
<price>349.99</price>
<quantity>2</quantity>
<subtotal>349.99</subtotal>
</item>
</items>
</Shopping>
```

The above listing creates the customer.xml file that stores billing information, such as product name, billing date, quantity, and product price.

You can obtain the HTML format of the customer.xml file after running the PHP script in your Web browser.

Listing B-4 shows how to convert the customer.xml file into the HTML format:

Listing B-4: Converting customer.xml into HTML Format

```
<?php
// Include class definition
include("eZXML.php");
// Arrays to associate XML elements with HTML output
$startTags = array
(
   'CUSTOMER'=> '<p> <b>Customer: </b>',
   'ADDRESS'=> '<p> <b>address: </b>',
   'DATE'=> '<p> <b>date: </b>',
   'ITEMS'=> '<p> <b>Details: </b> <table width="100%" border="1"
   cellspacing="0"
   cellpadding="3"><tr><td><b>Item
   description</b></td><td><b>Price </b></td>
   <td><b>Quantity</b></td><td><b>Sub-total </b></
   td></tr>',
   'ITEM'=> '<tr>',
   'PRODUCT'=> '<td>',
   'PRICE'=> '<td>',
   'QUANTITY'=> '<td>',
   'SUBTOTAL'=> '<td>',
   'TERMS'=> '<p> <b>Rules and Regulation: </b> <ul>',
   'TERM'=> '<li>'
);
$endTags = array
(
   'LINE'=> ', ',
   'ITEMS'=> '</table>',
```

```
    'ITEM'=> '</tr>',
    'PRODUCT'=> '</td>',
    'PRICE'=> '</td>',
    'QUANTITY'=> '</td>',
    'SUBTOTAL'=> '</td>',
    'TERMS'=> '</ul>',
    'TERM'=> '</li>'
);
// XML file
$xmlfile = "customer.xml";
$xmlString = join("", file($xmlFile));
// parse XML string and create object
$doc = eZXML::domTree($xmlString, array("TrimWhiteSpace" => "true"));
// start printing
print ($doc->children);
// Accepts an array of nodes recursively as argument.
// Iterates through the XML document.
// Constructs HTML markups.
// Displays the XML contents.
function print ($NODE)
{
   global $startTags, $endTags, $Totals;
   for ($t=0; $t<sizeof($NODE); $t++)
    {
      // How to handle elements
      if ($NODE[$t]->type == 1)
      {
         // Displays the opening tags
         echo $startTags[strtoupper($NODE[$t]->name)];
         // recurse
         print ($NODE[$t]->children);
         // Displays the closing tags
         echo $endTags[strtoupper($NODE[$t]->name)];
      }
      // How to handle text nodes
       if ($NODE[$t]->type == 3)
       {
         // Printing the text
         echo($NODE[$t]->content);
       }
   }
}
?>
```

The above listing converts the customer.xml file into the HTML format using the eZXML class. In the above listing, the eZXML.php file is included in the script to parse and manipulate the customer.xml file. The eZXML.php file contains functions, which represent XML document in a DOM tree format by creating nested objects.

In the above listing, the associative arrays, $startTags and $endTags, store the HTML markup, if the node is of element type. If the node is of text type, the content of the text node is displayed in the Web browser.

PREVIOUS NEXT

Appendix C: The PHP.XPath Class

PHP.XPath is a Hypertext Preprocessor (PHP) class that searches and retrieves data from an eXtensible Markup Language (XML) document using XPath. The PHP.XPath class uses the Document Object Model (DOM) Application Programming Interface (API) to modify an XML document.

This appendix introduces the PHP.XPath class, and describes the types of objects in this class. It also explains the public and private methods of the objects in the PHP.XPath class.

Introducting the PHP.XPath Class

The PHP.XPath class accepts XML data as its argument, parses the data, and creates an object that represents XML data. This object uses the methods defined in the PHP.XPath class to create XML nodesets on the basis of XPath expressions. You can add, delete, or edit the nodes from an XML document tree, and retrieve specific information from within an XML document.

If the PHP module on your server is not compiled with the XML library, you can use the PHP.XPath class to provide XML functionality. The PHP.XPath class consists of objects of three base classes, which are:

- XpathBase: Contains debugging functions.
- XpathEngine: Contains XML import and export methods. The XpathEngine base class is made up of the XPath specification.
- Xpath: Contains methods that modify an XML document.

Note You need not install the DOM XML PHP library to modify an XML document using the PHP.XPath class.

The XPathBase Class

The XPathBase class consists of public and private methods. A public method declared in a class is accessible to other classes. The public methods in the XPathBase class are:

- `XPathBase() method`: Represents the constructor of the XPathBase class.
- `reset() method`: Resets the XPathBase class object and enables the object to accept new XML data.
- `setVerbose() method`: Alters the details of error level reporting in an XML document. The syntax of the setVerbose() method is:

```
setVerbose($level=1)
```

In the above syntax, the $level parameter is set to the default value, 1. To turn off the error-level reporting, set the value of the $level parameter to 0. If the $level parameter has an integer value greater than 1, error-level reporting is turned on.

- `getLastError() method`: Returns the recent error message as a string value. This method returns a null value if there is no error message.

Note The higher the value of the $level parameter, the higher is the level of error.

A private method is not accessible outside the class definition. The private methods in the XPathBase class are:

- `_searchString() method`: Searches a string within another string in an XML document. The syntax of the _searchString() method is:

```
_searchString($term, $expression)
```

In the above syntax, the $expression parameter denotes the string to be searched; and the $term parameter indicates the string in which the _searchString() method searches the specified expression.

The _searchString() method returns an integer value. It returns the offset value at which the string is found. The _searchString() method returns –1, if the $expression parameter is not a substring of the string in the $term parameter.

- `_bracketsCheck() method`: Checks if there is an ending bracket corresponding to each starting bracket in an XPath expression. The syntax of the _bracketsCheck() method is:

```
_bracketsCheck($term)
```

In the above syntax, the $term parameter denotes the XPath string, in which the _bracketsCheck() method checks for a matching bracket.

- `_prestr() method`: Retrieves the substring positioned before a delimiter. The syntax of the _prestr() method is:

```
_prestr($string, $delimiter, $offset=0)
```

In the above syntax, the $string parameter denotes the string from which the _prestr() method retrieves a substring. The $delimiter parameter denotes the string that contains the delimiter for the _prestr() method. If the $offset parameter is set to 0, the method searches the $string string, starting from the first character, until it finds the delimiter for the first instant.

If the $offset parameter is set to a value other than 0, the _prestr() method searches the $string string from the specified offset value. The _prestr() method returns a substring that precedes the specified delimiter in the original string. An example of the _prestr() method is:

```
_prestr(Park:Town, ':', $offset=0)
```

In the above code, the _prestr() method extracts the Park substring from the Park:Town string because the Park string precedes the : delimiter.

- `_afterstr() method`: Retrieves the substring positioned after a delimiter. The syntax of the _afterstr() method is:

```
_afterstr($string, $delimiter, $offset=0)
```

In the above syntax, the $string parameter denotes the string from which the _afterstr() method retrieves a substring. The $delimiter parameter denotes the string that contains the delimiter. The _afterstr() method returns a substring that is positioned after the delimiter in the original string.

An example of the _afterstr() method is:

```
_afterstr(Park:Town, ':', $offset=0)
```

In the above code, the _afterstr() method extracts the Town substring from the Park:Town string because the Town string is positioned after the : delimiter.

- `_setLastError() method`: Generates an error message in textual form, and sets the error message to a variable. The syntax of the _setLastError() method is:

```
_setLastError($msg='', $line='-', $file='-')
```

In the above syntax, the $msg parameter represents an error message, the $line parameter represents the line number that contains the error, and the $file parameter represents the XML file name that is checked for error.

- `_displayError() method`: Displays an error message. The syntax of the _displayError() method is:

```
_displayError($msg, $line='-', $file='-', $terminate=TRUE)
```

In the above syntax, the $msg parameter represents the error message that the method displays, the $line parameter represents the line number that contains the error, and the $file parameter represents the XML file name that is checked for error. If the $terminate parameter, in the above syntax, is set to TRUE, it indicates that the file execution should stop.

The return type of the $msg and $file parameters is string, the return type of the $line parameter is an integer, and the return type of the $terminate parameter is boolean.

- `_beginDebugExecution() method`: Starts the debugging of the specified function. The syntax of the _beginDebugExecution() method is:

```
_beginDebugExecution($function)
```

In the above syntax, the $function parameter represents the name of the function that the _beginDebugExecution() method starts debugging. The return type of the _beginDebugExecution() method is an array value.

- `_closeDebugExecution() method`: Stops the debugging of the specified function. The syntax of the _closeDebugExecution() method is:

```
_closeDebugExecution($time, $return="")
```

In the above syntax, the $time parameter represents the time when the debugging starts, and the $return parameter represents the value that the debug function returns.

- `_printContext() method`: Displays an XPath context. The syntax for the _printContext() method is:

```
_printContext($context)
```

In the above syntax, the $context parameter represents the XPath context.

The XPathEngine Class

The XPathEngine class consists of import and export methods. It also handles XPath queries. The public methods of the XPathEngine class are:

- `XPathEngine() method`: Represents the constructor of the XPathEngine class. This method accepts a single optional argument that specifies the XML filename that you need to parse.

- `getProperties() method`: Returns the specified properties of the XPathEngine class. The syntax of the getProperties() method is:

```
getProperties($par=NULL)
```

In the above syntax, the $par parameter represents the property of the XPathEngine class whose value is to be retrieved. If the $par parameter is set to NULL, the getProperties() method retrieves all the properties of the XPathEngine class.

- `getNode() method`: Retrieves the nodes from an XML document tree. An XPath expression specifies the nodes that you need to retrieve. The syntax of the getNode() method is:

```
getNode($abs_XPath='')
```

In the above syntax, the $abs_XPath parameter is a string that represents the absolute path to be traversed for retrieving the specified nodes. The getNode() method returns the value, FALSE, if the specified node does not exist in the document tree.

- `exportToFile() method`: Generates an XML string that contains the content of the current document, and writes the XML string to the specified file. The syntax of the exportToFile() method is:

```
exportToFile($file, $abs_XPath='', $header=NULL)
```

In the above syntax, the $abs_XPath parameter represents the address of the node that you need to export. The $header parameter represents the string that should appear before the content in an XML file. If you specify the $header parameter as NULL, the header string is assigned the value of the XML header in the parsed XML file.

- `importFromFile() method`: Reads a Uniform Resource Locator (URL) or a file, and parses the XML data. The syntax of the importFromFile() method is:

```
importFromFile($file)
```

In the above syntax, the $file parameter represents the name of the XML file that you need to parse.

- `match() method`: Evaluates an XPath query by parsing it. The syntax of the match() method is:

```
match($query, $base='')
```

In the above syntax, the $query parameter represents the XPath expression that you need to evaluate, and the $base parameter represents the path of a document node from where the query executes.

- `equalNodes() method`: Compares two nodes to check if the nodes represent the same node in a document tree. The syntax of the equalNodes() method is:

```
equalNodes($node1, $node2)
```

In the above syntax, the $node1 and $node2 parameters represent either the absolute location paths to a specific node, or the node itself. The equalNodes() method returns TRUE if the two nodes are equal and FALSE if the nodes are not equal.

- `hasChildNodes() method`: Checks if the specified node contains child nodes. The syntax of the hasChildNodes() method is:

```
hasChildNodes($abs_XPath)
```

In the above syntax, the $abs_Xpath parameter represents the absolute path of the parent node. The hasChildNodes() method returns TRUE if the parent node exists and contains child nodes. If a parent node does not contain any child nodes, the hasChildNodes() method returns FALSE.

The private methods of the XPathEngine class are:

- `_InternalExport() method`: Exports an XML document, starting from the specified node in an XML document. The syntax of the _InternalExport() method is:

```
_InternalExport($node)
```

In the above syntax, the $node parameter represents the node in the XML document from where you need to start exporting the document.

- `_handleStartElement() method`: Handles the opening tags while parsing an XML document. The syntax of the _handleStartElement() method is:

```
_handleStartElement($parser, $node, $attributes)
```

In the above syntax, the $parser parameter represents the handler that accesses the XML parser, the $node parameter represents the opening tag in an XML document, and the $attributes parameter represents an associative array that contains a list of attributes of the opening tag.

- `_handleEndElement() method`: Handles the closing tags while parsing an XML document. The syntax of the _handleEndElement() method is:

```
_handleEndElement($parser, $node)
```

In the above syntax, the $parser parameter represents the handler for accessing the XML parser, and the $node parameter represents the closing tag in an XML document.

- `_handleCharacterData() method`: Handles character data while parsing an XML document. The syntax of the _handleCharacterData() method is:

```
_handleCharacterData($parser, $text)
```

In the above syntax, the $parser parameter represents the handler for accessing the XML parser and the $text parameter represents the character data in an XML document.

- `_checkPredicates() method`: Checks if a node matches the specified list of predicates. The syntax of the _checkPredicates() method is:

```
_checkPredicates($nodesets, $predicates)
```

In the above syntax, the $nodesets parameter represents an array that contains the location paths of the nodes that you need to match against the specified list of predicates. The $predicates parameter represents an array of predicates.

- `_getAxis() method`: Retrieves the axis name from an XPath query. The syntax of the _getAxis() method is:

```
_getAxis($step, $context)
```

In the above syntax, the $step parameter represents a string that contains the XPath query, and the $context parameter represents the context from where the method starts evaluating an XML document. The _getAxis() method returns an array that contains the axis name, and its nodetest.

- `_handleAxis_Child() method`: Handles the child axis of a document tree. The syntax of the _handleAxis_Child() method is:

```
_handleAxis_Child($axis, $context)
```

In the above syntax, the $axis parameter is an array that contains information about the axis of a node, and the $context parameter represents the path of the node of the axis that is processed.

The XPath Class

The XPath class also includes public and private methods. The public methods of the XPath class are:

- `XPath() method`: Represents the constructor of the XPath class.
- `nodeName() method`: Retrieves the names of the nodes from a document tree, along with the path of the nodes specified by the method parameter. The syntax of the nodeName() method is:

```
nodeName($query)
```

In the above syntax, the $query parameter represents the path from where you need to retrieve the nodes.

- `appendChild() method`: Adds child nodes to the existing child nodes in a document tree. The syntax of the appendChild() method is:

```
appendChild($query, $node, $afterText=FALSE, $autoReindex=TRUE)
```

In the above syntax, the $query parameter represents the path of the child node to which you append another child node. The $node parameter is either a string or an array. If the $afterText parameter is set to FALSE, the node is inserted after the text. If the $autoReindex parameter is set to TRUE, the XML document is reindexed to reflect the changes in the document. The appendChild() method returns the path of the appended child node.

The private methods of the XPath class are:

- `_title() method`: Generates a title line. The syntax of the _title() method is:

```
_title($title)
```

In the above syntax, the $title parameter represents the title that you add to an XML document.

- _xml2Document() method: Parses an XML document to a tree node. The syntax of the _xml2Document() method is:

```
_xml2Document($str)
```

In the above syntax, the $str parameter represents the string that is converted to a document node.

◄ PREVIOUS NEXT ►

Appendix D: XML-RPC for PHP

eXtensible Markup Language-Remote Procedure Calls (XML-RPC) is an XML-based protocol that lets you make remote procedure calls over the Internet. XML-RPC is encoded in XML, and uses HTTP to transfer the client requests and receive the responses from a server. You can transfer complex data over the Internet using XML-RPC.

Using XML-RPC, you can integrate and develop a Hypertext Preprocessor (PHP) Web application. XML-RPC involves the process in which the server provides procedures to call the clients. The calling program transfers a message to the remote program, which executes the procedure specified in the message and returns the result to the calling program. The XML-RPC implementation provides various basic RPC functions, such as xmlrpc_decode_request() and xmlrpc_encode_request(), which enables you to make procedure calls in HTTPS. XML-RPC also provides utility functions for the conversion of the PHP and XML-RPC data types.

This appendix describes various XML-RPC and utility functions.

XML-RPC Functions

Using XML-RPC functions, you can create XML-RPC clients and servers. The XML-RPC functions are:

- `xmlrpc_decode_request()`: Decodes XML strings into native PHP types. The syntax of the xmlrpc_decode_request () function is:

  ```
  array xmlrpc_decode_request (string xml, string method [, string encoding])
  ```

- `xmlrpc_encode_request()`: Generates the XML language for a method request. The syntax of the xmlrpc_encode_request() function is:

  ```
  string xmlrpc_encode_request (string method, mixed params)
  ```

- `xmlrpc_get_type()`: Retrieves the xmlrpc data type for a PHP value. The syntax of the xmlrpc_encode_request() function is:

  ```
  string xmlrpc_get_type (mixed value)
  ```

- `xmlrpc_parse_method_descriptions()`: Decodes the XML language into a list of method descriptions. The syntax of the xmlrpc_parse_method_descriptions function is:

  ```
  array xmlrpc_parse_method_descriptions ( string xml)
  ```

- `xmlrpc_server_call_method()`: Parses the XML requests. The syntax of the xmlrpc_server_call_method() function is:

  ```
  mixed xmlrpc_server_call_method (resource server, string xml, mixed user_data [, array out
  ```

- `xmlrpc_server_create()`: Creates an xmlrpc server. The syntax of the xmlrpc_server_create() function is:

  ```
  resource xmlrpc_server_create (void)
  ```

- `xmlrpc_server_destroy()`: Destroys the server resources. The syntax of the xmlrpc_server_destroy() function is:

  ```
  void xmlrpc_server_destroy (resource server)
  ```

- `xmlrpc_server_register_method()`: Registers a PHP function to a resource method matching the methodspecified as argument in the xmlrpc_server_register_method () function. The syntax of the xmlrpc_server_register_method() function is:

  ```
  bool xmlrpc_server_register_method (resource server, string methodName, string function)
  ```

- `xmlrpc_set_type()`: Specifies the xmlrpc data type, or datetime, for a PHP string value. The syntax of the xmlrpc_set_type() function is:

  ```
  bool xmlrpc_set_type (string value, string type)
  ```

Utility Functions

In addition to basic RPC functions, XML-RPC provides various utility functions to convert PHP data into XML-RPC compatible data types. Utility functions also provide a set of APIs to support HTTPS transactions. The utility functions are:

- `XMLRPC_prepare`: Transforms data into the appropriate data structure to be serialized into an XML-RPC message. For example, if you pass an associative array as an argument to the XMLRPC_prepare() function, the XMLRPC_prepare() function serializes it into an XML-RPC struct. The syntax of the XMLRPC_prepare() function is:

  ```
  XMLRPC_prepare($data, $type = NULL)
  ```

- `XMLRPC_request`: Connects to the site specified as argument to the XMLRPC_request() function, at the specified location, calls the method, and passes the parameters to the method. The syntax of the XMLRPC_request() function is:

  ```
  XMLRPC_request($siteName, $location, $methodName, $param = NULL, $useragent = NULL)
  ```

- `XMLRPC_response()`: Sends back an XML-RPC response to the server. The syntax of the XMLRPC_response() function is:

  ```
  XMLRPC_response($return_value, $server = NULL)
  ```

- `XMLRPC_error()`: Sends an XML-RPC error message. The syntax of the XMLRPC_error() function is:

  ```
  XMLRPC_error($fault_Code, $fault_String, $server = NULL)
  ```

- `XMLRPC_convert_timestamp_to_iso8601()`: Lets you set the formatted time string required by XML-RPC. The syntax of the XMLRPC_convert_timestamp_to_iso8601() is function

  ```
  XMLRPC_convert_timestamp_to_iso8601($timestamp)
  ```

- `count_numeric_items()`: Returns the number of numeric indices of an array. The syntax of the count_numeric_items() is function is:

  ```
  count_numeric_items($array)
  ```

- `XML_serialize()`: Accepts a data structure, and serializes it into XML format. The syntax of the XML_serialize() function is:

  ```
  & XML_serialize($data)
  ```

- `XMLRPC_debug_print`: Displays a table of debug messages and erases the debugging logs to ensure the error messages are not repeated. The syntax of the XML_serialize() function is:

  ```
  function XMLRPC_debug_print()
  ```

- `XMLRPC_debug()`: Logs the debugging messages. The syntax of the XMLRPC_debug() function is:

  ```
  XMLRPC_debug($function_name, $debug_message)
  ```

- `XMLRPC_adjustValue()`: Converts an XML-RPC data structure into a native PHP data structure. The syntax of the XMLRPC_adjustValue() function is:

  ```
  & XMLRPC_adjustValue(&$current_node)
  ```

Appendix E: Implementing SOAP using SOAPx4

Download CD Content

Simple Object Access Protocol (SOAP) is a W3C standard for exchanging information between various applications developed in different programming languages and running on various operating systems in a network. SOAPx4 is the PHP implementation of SOAP. SOAPx4 implements SOAP using the PHP server and client classes. SOAPx4 implements Web Services Description Language (WSDL) in Hypertext Pre-Processor (PHP).

This appendix explains how to implement SOAP requests and responses using SOAPx4. The appendix also explains how to create the SOAP server and SOAP client to process the requests of the clients in a distributed network.

Connecting a SOAP Client to a SOAP Server

A server accepts client requests in the form of SOAP request, and responds to the client in the form of SOAP response. The SOAP messages are transmitted in a distributed network in the form of header and body sections consisting of multiple soapval objects. You use the soapval object to specify a SOAP value. The syntax to use the soapval object is:

```
$var=new soapval(name, data_type, data)
```

The above syntax shows that the constructor of the soapval object accepts three parameters: name, data_type, and data. The name parameter indicates the name of the element, and the data_type parameter indicates the type of data to be stored in the element. The data parameter indicates the value of the element. The variable, $var, contains information stored in the soapval object.

The soapval and soapmsg objects create SOAP requests and SOAP responses. The soapmsg object of SOAPx4 contains a serialized message that is sent to the client and server of SOAP-based systems. The syntax to use the soapmsg object is:

```
$var=new soapmsg(procedure_name, array of soapval objects)
```

The above syntax shows that the constructor of the soapmsg object accepts two parameters, procedure_name and array of the soapval objects. The procedure_name parameter indicates the name of the procedure that is requested by the client, and the second parameter indicates the information to be sent to server. The $var variable indicates the complete SOAP message that is transmitted from the client to the server.

Creating a SOAP Request

You need to create a SOAP request that invokes the procedure of the SOAP server. In SOAPx4, you need to include the class.soap_client.php class to initiate the SOAP requests and invoke the required procedure.

Listing E-1 shows how to create a SOAP request using the soapval objects:

Listing E-1: Creating a SOAP Request

```
<?php
//Include the class definition of the php file that you can use to implement SOAP.
include("class.soap_client.php");
//Initialize the constructor of the soapval object that accepts three parameters.
$object=new soapval("student","string","John Williams");
//Create an object of the soapmsg object that contains data of the soapval object.
$message=new soapmsg("getStudentData", array($object));
//Create a SOAP packet that contains complete message to be transmitted using the serialize() functi
print $message->serialize();
?>
```

The above listing shows that the SOAP packet is a well-formed XML document, because it contains the envelope that indicates the encoding scheme used by the SOAP packets. The listing shows the envelope body that indicates the information to be sent.

The above listing shows that the class.soap_client.php file class is included in the SOAP request class to implement SOAP in PHP. The $object variable contains information that is passed to the procedure stored in the server class. The $message variable contains the complete SOAP information that is transmitted to the server.

> **Note** You can use the serializeval() function to serialize data into XML form. The command to display the serialized value of the soapval object is:
>
> ```
> print $object->serializeval();
> ```

The serialization process of SOAP creates SOAP packets in the XML form, which contains request that is transmitted from the client to the server.

Listing E-2 shows a generated SOAP packet:

Listing E-2: The Generated SOAP Packet

```
<xml version="1.0"?>
<SOAP-ENV: Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="httpd://www.w3.org/2001/XMLSchema"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:si=http://soapinterop.org/xsd
xmlns:ns6=http://testuri.org SOAP-ENV:
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
>
<SOAP-ENV:Body>
<ns6:getStudentData>
<student xsi:type="xsd:string">John Williams</student>
</ns6:getStudentData>
</SOAP-ENV-Body>
</SOAP-ENV:Envelope>
```

The above listing shows that the SOAP packet is a well-formed XML document, because it contains the envelope that indicates the encoding scheme used by the SOAP packets. The listing shows the envelope body that indicates the information to be sent.

Creating a SOAP Response

You need to create a SOAP response that transmits message to the SOAP request.

Listing E-3 shows how to create a SOAP response:

Listing E-3: Creating a SOAP Response

```
<?php
include("class_soap_client.php");
$object=new soapval("data","string","Age:15, Standard: 10th");
$message=new soapmsg("getStudentData", array($object));
print $msg->serialize();
?>
```

The above listing shows that the soapval object contains information stored in the getStudentData procedure. The $message variable contains the complete SOAP packet that is transmitted from the SOAP response to the SOAP request.

You can also serialize the response, sent to the client, using the serialize() function.

Listing E-4 shows a generated response in the form of SOAP packets:

Listing E-4: The Generated SOAP Response

```
<xml version="1.0"?>
<SOAP-ENV: Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsd="httpd://www.w3.org/2001/XMLSchema"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xmlns:SOAP-ENC=http://schemas.xmlsoap.org/soap/encoding/
xmlns:si=http://soapinterop.org/xsd
xmlns:ns6=http://testuri.org SOAP-ENV:
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
>
<SOAP-ENV:Body>
<ns6:getStudentData>
<data xsi:type="xsd:string">Age:15, Standard: 10th</data>
</ns6:getStudentData>
</SOAP-ENV-Body>
</SOAP-ENV:Envelope>
```

The above listing shows that the SOAP packet is a well-formed XML document, because it contains the envelope that indicates the encoding scheme to be used by SOAP packets. The listing also shows the envelope body that indicates the information to be sent as the SOAP response to the SOAP request.

Creating a SOAP Server

You create a server class in PHP to implement the SOAP responses. PHP provides the server class to implement SOAP response in SOAPx4. You need to include the class.soap_server.php and class.soap_client.php classes to initiate a SOAP server in SOAPx4. The add_to_map() and service() functions register the required procedures with the newly created server, and send the responses to the clients.

The add_to_map() function adds the requested procedure to the list of procedures stored in the server. This function accepts three parameters: procedure_name, array_of_parameters, and return_type. The procedure_name parameter specifies the name of the procedure that is to be registered with the server. The second parameter specifies the array of the parameters passed to the procedure. The return_type parameter specifies the type of data returned by the procedure. The syntax to use the add_to_map() function is:

```
$server_object->add_to_map("procedure_name", "array_of_parameters", "return_type");
```

In the above syntax, the $server_object variable indicates the reference of the server object, which invokes the add_to_map() function to register the procedure.

The service() function processes the SOAP requests and returns the SOAP packets as a response to the client.

Listing E-5 shows how to create a SOAP server:

Listing E-5: Creating a SOAP Server

```
<?php
include("class.soap_client.php");
include("class.soap_server.php");
// Initialize a SOAP server.
$server=new soap_server;
$server->add_to_map("retrievePOP3Messages", array("SOAPStruct"), array("int"):
$server->service($HTTP_RAW_POST_DATA);
// The retrievePOP3Message function is a user-defined function that returns the number of
messages stored in the mailbox.
function retrievePOP3Messages($structure)
{
   $box=imap_open("{".$structure["pop_host"]."/pop3:110}", $structure["pop_user"],
   $structure["pop_pass"]);
   //If the connection is successfully established.
   if($box)
   {
    //Retrieve the number of messages.
    $totalMessage=imap_num_msg($box);
    imap_close($box);
    return $totalMessage;
   }
   else
   {
      // Generate an error.
      $parameter=array("errorCode"=>"75", "errorString"=>"Connection Not Established",
      "detail"=>"Not able to connect POP3 Server");
      $errMsg=new soapmsg("Errror", $parameter, http://schemas.xmlsoap.org/soap/envelope/
      );
   }
}
?>
```

In the above listing, the SOAP server establishes a connection with the mail server, POP3, and returns the number of messages stored in the mailbox. In the above code:

- The retrievePOP3Messages function is a user-defined function that takes the parameters passed by the clients.
- The imap_open() function establishes a connection with the POP3 mail server by passing three parameters: user name, password, and the host name of the mail server.
- The reference of the mail server is stored in the $box variable.
- The imap_num_msg() function retrieves the number of messages stored in the mailbox if the connection is established with the mail server.
- The imap_num_msg() function returns the value of the $totalMessage variable that contains the number of messages stored in the mailbox.
- The retrievePOP3Message() function generates an error if the connection with the mail server is not established.

Creating a SOAP Client

You need to include the class.soap_client.php class to initiate a SOAP client in SOAPx4. The SOAP client generates a SOAP request, and translates the SOAP packet returned as a response by the requested procedure.

The soapclient object lets you initiate a SOAP client with SOAPx4. The constructor of the soapclient object contains the location of the SOAP server as a parameter. The call() function of the soapclient object sends a request to the SOAP server and retrieves the SOAP packets as a response from the server.

This function accepts four parameters: procedure name, array of the soapval object, namespace, and SOAPAction parameter. The syntax to use the call() function is:

```
$client_object->call(procedure_name, array_of_soapval_object, namespace, SOAPAction)
```

In the above syntax, the procedure_name parameter indicates the name of the procedure to be invoked. The array_of_soapval_object parameter indicates the parameters passed by the SOAP client to the procedure. The urn:soapserver is the value for both namespace and SOAPAction parameters.

You can create a soap client code in any client-side script, such as Hypertext Markup Language (HTML) or Active Server Pages (ASP).

Listing E-6 shows how to code a SOAP client in HTML:

Listing E-6: Creating a SOAP Client

```
<html>
<head>
<basefont face="Times New Roman">
</head>
<body>
<?php
if(!$_POST['submit'])
{
   ?>
   <table border="0" cellspacing="5" cellpadding="5">
   <form action="<? echo $_SERVER['PHP_SELF']; ?>" method="POST">
   <tr>
   <td><b>Username:</b></td>
   <td><input type="text" name="user_name"></td>
   </tr>
   <tr>
   <td><b>Password:</b></td>
   <td><input type="password" name="user_pass"></td>
   </tr>
   <tr>
   <td><b>POP Server:</b></td>
   <td><input type="text" name="host_name"></td>
   </tr>
   <tr>
   <td colspan="2"  align="center"><input type="submit" name="submit" value="Retrieve
   Total Messages"></td>
   </tr>
   </form>
   </table>
   <?php
}
else
{
   include("class_soap_client.php");
   $server=http://mail.service/rpc/server.php;
   $parameter=array("user_name"=>$_POST['user_name'],
   "user_pass"=>$_POST['pop_pass'], "host_name"=>$POST['host_name']);
   $client=new soapclient($server);
   $object=new soapval("parameters","SOAPStruct",$parameter);
   echo $client->call("retrievePOP3Messages", array($object), "urn:soapserver",
   "urn:soapserver");
}
?>
</body>
</html>
```

The above listing shows that the HTML page accepts the user name, password, and the name of the mail server from the client. This information is passed to the retrievePOP3Messages procedure of the SOAP server, by clicking the Submit button. The SOAP server returns the number of messages stored in the mailbox.

Note You can display the client debug messages using the command:

```
$client->debug_flag=true;
```

Appendix F: PHPXML Classes

The PHPXML class is a collection of classes that process XML documents using PHP. PHPXML classes let you access XML documents using eXtensible Markup Language Path (XPath) language.

This appendix explains the packages of PHPXML classes. This appendix also explains the various classes present in the packages.

Classes in PHPXML

A package is a collection of classes that you can use to combine related classes. PHPXML consists of the following classes used for querying an XML document:

- Xml_check
- RDQL_query_document
- RDF_iterator
- RDQL_query
- RDF_document_iterator
- RDQL_db
- RDQL_query_db
- XqueryLite
- class_xindice

PHPXML classes contain the following classes used for validating an XML document:

- Path_parser
- RDDL_parser
- RSS_parser
- RDF_parser
- AbstractSAXParser
- AbstractFilter
- ExpatParser
- FilterOutput
- class_schematron
- class_xslt

The XML_check Class

The Xml_check class validates whether an XML document is well-formed or not. If the XML document is well-formed, the Xml_check class returns various information, such as the number of elements, attributes, and text section. The Xml_check class also returns the line number and column number of the XML document where an error is encountered. The XML_check class contains various functions, such as:

- `get_xml_elements()`: Provides the number of elements in an XML document.
- `get_xml_attributes()`: Provides the number of attributes in an XML document.
- `get_xml_size()`: Specifies the size of an XML document.

Resource Description Format Data Query Language

Resource Description Format (RDF) Data Query Language (RDQL) is a language that you use to query RDF documents from local file systems or URLs. RDQL is similar to Structured Query Language (SQL). You can implement RDQL using the following four classes:

- `RDQL_query_document`
- `RDF_iterator`
- `RDQL_query`
- `RDF_document_iterator`

The RDQL_query_document class implements the RDQL engine to query the RDF documents based on a URL and pathname. This class contains the rdql_query_url(string $query) function that queries the URL or string specified by the $query parameter. The rdql_query_url() function returns an associative array that contains the result of the RDQL query.

The RDQL engine uses the RDF_iterator class to query RDF documents from various sources, such as files and databases. The iterator is an object that specifies how to access and parse RDF documents. This class contains two functions, which are:

- `tuple_match(array $condition, array $row)`: Returns the value, true, if a row satisfies a condition.
- `find_tuples(array $condition, array $row)`: Returns the rows that satisfy a condition.

The RDQL_query class retrieves the RDQL rows from the RDF document using the RDQL engine. This class contains various functions, such as:

- `RDQL_query(RDF_iterator $iterator)`: Accesses the RDF rows in the RDF documents. It is a constructor of the class that does not return any value.
- `parse_query(string $query)`: Returns the results of the RDQL query.
- `tokenize(string $expression)`: Returns an array that contains various sections, such as SELECT, FROM, and WHERE, of the RDQL expression.
- `parse_select(string $expression)`: Returns an array that contains the SELECT section of the RDQL expression.

The RDF_document_iterator class contains an iterator that creates RDQL queries. This class does not contain any function.

Note RDF is a language that provides the accessed resources in the form of URLs.

The RDQL_db Class

The RDQL Database (DB) package is implemented using the class_rdql_db.php class. The RDQL DB package contains two classes, RDQL_db and RDQL_query_db. The RDQL_db class retrieves, stores, and deletes RDF documents from the MySQL database. Using the key of an RDF document, you can retrieve, store, and delete documents from the database. The primary key of a database is also called key. This class contains various functions, such as:

- `get_rdf_document(string $key)`: Returns a PHP string that is accessed by the key of the RDF document.
- `store_rdf_document(string $url, string $key)`: Returns the value, true, if the RDF document is stored in the database with the specified key. The $url variable provides a URL or file-path of the RDF document.
- `remove_rdf_document(string $key)`: Deletes an RDF document from the database, based on the key of the document, which is specified as an argument of the function.

You can query multiple documents from the database using the RDQL_query_db class, by specifying the name of documents in the FROM section of the query. You can use asterisk, *, to query all the documents stored in the database. This class contains the rdql_query_db(string $query) function that returns the result of the RDQL query.

The Path_parser Class

The Path_parser class implements the event-driven parsers, such as Expat. This class invokes the handlers when an element is encountered in an XML document. This class contains various functions to parse the XML documents, such as:

- `init()`: Parses the documents using a single object.
- `parse_file(string $url)`: Returns the value, true, if the document indicated by the $url variable is successfully parsed.
- `set_handler(string $url, string $handler)`: Processes XML elements when a specified handler is invoked.

The RDDL_parser Class

Resource Directory Description Language (RDDL) contains the RDDL_parser class that parses RDDL documents. This class contains a function that returns information pertaining to RDDL resources in an associative PHP array. Each element of an associative array contains various resources, such as role, href, type, title, and id. You can parse an RDDL document from URLs or files. The RDDL_parser class contains various functions, such as:

- `rddl_parse(string $url)`: Returns the value, true, if the RDDL document indicated by the $url variable is successfully parsed.
- `get_resources(string $rddl)`: Returns an associative array that contains resources found in an RDDL document.
- `get_error()`: Returns an error message when the rddl_parser() function returns the value, false.

Note An eXtensible Hyper Text Markup Language (XHTML) document that contains the <resource> element is known as an RDDL document.

The RSS_parser Class

Really Simple Syndication (RSS) contains the RSS_parser class that implements the RSS1.0 parser. This class contains a

function that parses RSS documents and returns an associative array. Each element of an associative array contains information about channels, such as channel data, channel_image, channel_textinput, and channel_items, in the RSS document. The RSS_parser class contains various functions, such as:

- `rss_parse(string $url)`: Returns the value, true, if the RSS document specified by the $url variable is successfully parsed.
- `get_channel_data(string $url)`: Returns an associative array that contains information about the channels, such as link, description, image, and text input.
- `get_items_data(string $url)`: Returns an associative array that contains information about the items stored in the RSS document.

Note RSS is an XML-based language that displays information about Web sites, such as hyperlinks and the content of the Web sites.

The Rdf_parser Class

Resource Description Framework (RDF) contains the Rdf_parser class that parses the RDF document with the RDF specifications. The Rdf_parser class contains various functions, such as:

- `rdf_parser_create(string $encoding_scheme)`: Returns the value, true, if a new parser is created. The $encoding_scheme parameter of the function, which specifies the encoding scheme to parse a document, is optional.
- `rdf_set_element_handler(string $start, string $end)`: Invokes the handlers when the start and end elements of the non-RDF elements are encountered in the RDF document.
- `rdf_Parse(string $str, $length, $is_final)`: Returns the value, true, if the RDF document is successfully parsed. The $str parameter indicates the chunk of data to be parsed, the $length parameter indicates the length of data to be parsed, and the $is_final parameter indicates the final data to be parsed.

Note RDF is a language that you can use to represent information on the Internet.

The XqueryLite Class

The XqueryLite class queries XML documents. This class lets you run queries in the Xquery 1.0 language, using files and PHP strings. The constructor of the XqueryLite class does not accept any parameter, and does not generate any result. The init() function initializes a query. The XqueryLite class contains various functions, such as:

- `evaluate_xqueryl(string $query)`: Runs an Xquery Lite 1.0 query.
- `_get_root_name(object $node)`: Returns the name of the root element of the XML document.
- `_parse_query(string $query)`: Returns the result of the Xquery Lite expression.

Note XqueryLite is a language that queries XML documents.

The class_sax_filters.php Class

The Simple Application Programming Interface (API) for XML (SAX) filters package contains the class_sax_filters.php class. This class contains a set of classes that implements the SAX parser filters. The Expat parser is an example of the SAX parser. Filters modify, query, update, and transform XML documents.

The class_sax_filters.php class contains four classes, which are:

- `AbstractSAXParser`
- `AbstractFilter`
- `ExpatParser`
- `FilterOutput`

The class_schematron Class

Schematron is a validation language that contains the class_schematron class to validate XML documents. You can validate XML documents using files, PHP strings and URLs. The class_schematron class contains various functions, such as:

- `compile_schematron_from_file(string $file)`: Returns an XSLT stylesheet after compiling the Schematron script.
- `validate_mem_using_file(string $xml_str, string $validation_file)`: Checks the string of the XML document specified in the $xml_str variable from the file specified in the $validation_file parameter.
- `validate_mem_using_mem(string $xml_str, string $validation_str)`: Checks an XML document using the specified PHP string in the $validation_str parameter.

Note Schematron is a structured schema language that creates a tree structure for validating XML documents. You can use this class to perform validations in PHP.

The class_xslt Class

eXtensible Style sheet Language Transformation (XSLT) contains the class_xslt class to implement the XSLT processor. This

class supports XSLT transformations and XML documents. This class contains functions that set the XSLT style sheet to be used from PHP strings and URLs. The class_xslt class contains various functions, such as:

- `getOutput()`: Provides the output of the XSLT transformation.
- `transform(string $url)`: Processes an XSLT transformation.
- `setXslString(string $xsl)`: Assigns the string to be used for the XSLT stylesheet.
- `setXsl(string $url)`: Assigns the URL to be used for the XSLT stylesheet.

The class_xindice Class

The class_xindice class accesses a Xindice 1.0 XML database using PHP script. The constructor of the class accepts two parameters, the URL of the Xindice server and the port of the server. A function of the class returns the number of documents present in the collection of databases. The class_xindice contains various functions, such as:

- `createCollection(string $base, string $collection)`: Builds a collection of databases with the name specified in the $collection variable.
- `getDocumentCount(string $collectionLocation)`: Provides the number of documents that exist in the specified location of a collection.
- `InsertDocument(string $collection, string $id, $xmldoc)`: Sets a new document, specified by the $xmldoc variable, in the collection.
- `getDocument(string $collection, string $id)`: Accesses a document from the database.
- `removeDocument(string $collection, string $id)`: Deletes a document from a collection.

Note You need to install the Xindice XML-RPC plugin because the class_xindice class uses it.

PREVIOUS NEXT

Appendix G: PHP and Extensible Stylesheet Language for Transformation

 Download CD Content

Hypertext Preprocessor (PHP) is a server-side scripting language that creates Web applications and processes eXtensible Markup Language (XML) documents using eXtensible Stylesheet Language for Transformation (XSLT). XSLT transforms XML documents into other formats, such as Hypertext Markup Language (HTML), ASCII, and Wireless Markup Language (WML). PHP uses the Sablotron XSLT processor to perform server-side transformation of XML documents.

This appendix explains how to use PHP with XSLT to transform XML documents.

Implementing XSLT with PHP

The XSLT processor combines PHP and XSLT to perform server-side XSLT transformations that convert the XML files into HTML files. PHP includes an XSLT extension that supports various XSLT processors, such as Sablotron and Xalan. The XSLT extension contains an Application Programming Interface (API) that lets you work with other processors. An API contains functions that are compatible with other XSLT engines. You can use these functions to change the XSLT processor without modifying the PHP code.

The PHP XSLT extension uses the Sablotron XSLT processor as the default processor. You need to install the Sablotron XSLT processor explicitly because it is not automatically installed with the Linux operating system. You can enable the Sablotron XSLT processor by reconfiguring and installing PHP.

To perform an XSL transformation with PHP:

1. Include the xml and xsl files in the PHP script.
2. Create an object of the XSLT processor, using the xslt_create() function of PHP, as shown in the following code:

   ```
   $xp=xslt_create();
   ```

   The above code shows that the XSLT processor is referenced by the $xp variable, and used by other XSLT functions, such as xslt_process().
3. Process the XML source document and stylesheet using the xslt_process() function of PHP. The xslt_process() function accepts three mandatory parameters: the reference of the XSLT processor, the XML document, and the stylesheet.

   Listing G-1 shows how to use the xslt_process() function:

   Listing G-1: Using the xslt_process() Function

   ```
   if($res=xslt_process($xp, $xfile, $xsltfile))
   {
      echo $result;
   }
   else
   {
      echo "Error";
   }
   ```

   In the above code:

   - The $xp parameter specifies the reference of the XSLT processor.
   - The $xfile parameter specifies the name of the XML document.
   - The $xsltfile parameter specifies the name of the stylesheet.
   - The xslt_process() function reads and validates the XML document against the rules specified in the stylesheet, and then generates a result tree.
   - The $res variable contains the result of the transformation.
   - The statement, echo $result, displays the result tree if the transformation is successful; otherwise, displays an error message.
4. Release the memory occupied by the XSLT processor using the xslt_free() function. The xslt_free() function accepts a parameter that specifies the reference of the XSLT processor. The code to use the xslt_free() function is:

   ```
   xslt_free($xp);
   ```

Note The xslt_process() function accepts six parameters, out of which, three parameters are optional and three are mandatory. The three optional parameters are:

- Name of the file that stores the result tree.
- Name of an associative array that contains buffers.

- Name of an associative array that contains XSLT parameters.

Handling Errors

PHP XSLT extension provides functions for handling and displaying errors generated at the time of the transformation of XML documents. The xslt_error() function displays the error message, and the xslt_errno() function returns the error code. You can define a user-defined function for handling errors, using the xslt_set_error_handler() function. The syntax of the xslt_error() function is:

```
xslt_error($xslt_processor);
```

In the above syntax, the $xslt_processor parameter indicates the reference of the XSLT processor. The xslt_error() function accepts only a single parameter, and generates a system-defined error message. If you do not provide a parameter to the function, it displays the last error message generated in the transformation process.

The syntax of the xslt_errno() function is:

```
xslt_errno($xslt_processor);
```

In the above syntax, the $xslt_processor parameter indicates the reference of the XSLT processor. The xslt_errno() function accepts only a single parameter, and returns the error code corresponding to the generated error. If you do not provide a parameter to the function, the function returns the error code of the last error generated in the transformation process.

The xslt_set_error_handler() function lets you define a user-defined function for handling errors. This function accepts two parameters, the reference of the XSLT processor and the name of the user-defined function to be invoked.

The syntax of the xslt_set_error_handler() function is:

```
xslt_set_error_handler($xslt_processor, "user_defined_function");
```

In the above syntax, the $xslt_processor parameter specifies the reference of the XSLT processor. The second parameter specifies the name of the function to be invoked when an error occurs in the transformation process.

The user-defined function for handling error accepts four parameters, which are:

- The reference of the XSLT processor.
- The error level.
- The error code.
- The associative array containing error information.

Logging Processor Messages

Error messages are stored in a log file, which specifies the description of errors generated in the transformation process. The xslt_set_log() file lets you initiate the logging process. You need to specify the name of the log file in the xslt_set_log() function, and invoke the xslt_set_log() function twice. The first invocation of the function specifies the initiation of the logging process, and the second invocation specifies the name of the file that stores the logging information. When you invoke the xslt_set_log() function for the first time, it accepts two parameters, which are:

- The reference of the XSLT processor.
- The Boolean value, true.

Enter the following code to invoke the xslt_set_log() function for the first time:

```
xsl_set_log($xp, true);
```

When you invoke the xslt_set_log() function for the second time, it accepts two parameters, which are:

- The reference of the XSLT processor.
- The name of a file that stores the log messages.

Enter the following code to invoke the xslt_set_log() function for the second time:

```
xsl_set_log($xp, "/xfile.log");
```

In the above code:

- The $xp parameter indicates the reference of the XSLT processor.
- The xfile.log file stores the logging information.
- The Web browser displays the logging information if you do not provide the second parameter.
- The logging information is appended at the end of the log file if the log file exists.

Using Named Buffers

Named buffers are the references that help store XML and XSLT documents in memory. You can provide the named buffers in the fifth parameter of the xslt_process() function. You can use the named buffers of the XML and XSLT strings in place of the XML and XSLT files specified in the second and third parameters of the xslt_process() function. Enter the following code to create and use the named buffers of the XML and XSLT strings:

```
$xml_str=join('',file($xfile);
$xslt_str=join('', file($xsltfile));
$parameter_buff=array("/xml"=>$xml_str, "/xslt"=>$xslt_str);
$xp=xslt_create();
$res=xslt_process($xp, "arg:/xml", "arg:/xslt", NULL, $parameter_buff);
```

In the above code:

- The file() function of PHP reads the contents of the file and stores it in an array.
- The join() function stores the text of a file in a single string.
- The $xml_str variable contains the named buffer of the XML document.
- The $xslt_str variable contains the named buffer of the XSLT document.
- The $parameter_buff parameter is an associative array that consists of two key-value pairs.
    - The /xml key specifies the named buffer of the XML document.
    - The /xslt key specifies the named buffer of the XSLT document.

If you use the Sablotron processor, you need to prefix the named buffers with the arg: text in the xslt_process() function.

PREVIOUS NEXT

Index

A-C

Index

D-E

Index

H-I

Index

J-M

Index

O-P

Index

R-S

Index

U-W

Index

X

List of Figures

Chapter 1: Introducing PHP and XML

Chapter 2: PHP and Simple Application Programming Interface for XML

Chapter 3: PHP and Document Object Model

Chapter 5: PHP and XPath

Chapter 6: PHP and XML-Remote Procedure Calls

Chapter 7: PHP and Web Distributed Data Exchange

Chapter 8: Working with Databases using PHP and XML

Chapter 9: Creating an Online Shopping Cart Application

PREVIOUS NEXT

List of Tables

List of Listings

Chapter 1: Introducing PHP and XML

Chapter 2: PHP and Simple Application Programming Interface for XML

Chapter 3: PHP and Document Object Model

CD Content

Following are select files from this book's Companion CD-ROM. These files are copyright protected by the publisher, author, and/or other third parties. Unauthorized use, reproduction, or distribution is strictly prohibited.

Click on the link(s) below to download the files to your computer:

File	Description	Size
All CD Content	Integrating PHP and XML	80,276
Chapter 1:	Introducing PHP and XML	6,991
Chapter 2:	PHP and Simple Application Programming Interface for XML	7,201
Chapter 3:	PHP and Document Object Model	7,742
Chapter 4:	Understanding Extensible Stylesheet Language for Transformation	2,511
Chapter 5:	PHP and XPath	4,726
Chapter 6:	PHP and XML-Remote Procedure Calls	4,613
Chapter 7:	PHP and Web Distributed Data Exchange	4,095
Chapter 8:	Working with Databases using PHP and XML	8,582
Chapter 9:	Creating an Online Shopping Cart Application	30,574
Appendix E:	Implementing SOAP using SOAPx4	3,161
Appendix G:	PHP and Extensible Stylesheet Language for Transformation	300

Integrating PHP and XML

SkillSoft Press © 2004

Learn how to use SAX, XSLT, and XPath to manipulate XML documents, as well as use of XML-RPC protocol for accessing procedures on a remote computer, and much more.

Table of Contents