SRINIVASAN COLLEGE OF ARTS & SCIENCE
(Affiliated to Bharathidasan University, Trichy)
PERAMBALUR – 621 212.

DEPARTMENT OF COMPUTER APPLICATIONS

Course: MCA Year:**II** Semester:**IV**

Course Material on:

WEB TECHNOLOGIES

Prepared by:

Ms.S.Lavanya, MCA., M.Phil.,

Month & Year: MARCH – 2020

PHP NOTES

Unit-1

PHP Syntax:

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with `<?php` and ends with `?>`:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is "`.php`".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "`echo`" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

Note: PHP statements end with a semicolon (`;`).

PHP Case Sensitivity

In PHP, NO keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are case-sensitive.

In the example below, all three echo statements below are equal and legal:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
0.EcHo "Hello World!<br>";
?>

</body>
</html>
```

Note: However; all variable names are case-sensitive!

Look at the example below; only the first statement will display the value of the $color variable! This is because $color, $COLOR, and $coLOR are treated as three different variables:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```

Exercise:

Insert the missing part of the code below to output "Hello World".

□ "Hello World"

PHP Comments:

Comments in PHP

A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand your code
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports several ways of commenting:

Example

Syntax for single-line comments:

```
<!DOCTYPE html>
<html>
<body>

<?php
// This is a single-line comment

# This is also a single-line comment
?>

</body>
</html>
```

Example

Syntax for multiple-line comments:

```
<!DOCTYPE html>
<html>
```

```
<body>

<?php
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
?>

</body>
</html>
```

Example

Using comments to leave out parts of the code:

```
<!DOCTYPE html>
<html>
<body>

<?php
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>

</body>
</html>
```

PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

```
<?php
$txt = "Hello world!";
$x = 5;
```

```
$y = 10.5;
?>
```

After the execution of the statements above, the variable $txt will hold the value Hello world!, the variable $x will hold the value 5, and the variable $y will hold the value 10.5.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Think of variables as containers for storing data.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

Remember that PHP variable names are case-sensitive!

Output Variables

The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
<?php
$txt = "W3Schools.com";
echo "I love $txt!";
?>
```

The following example will produce the same output as the example above:

Example

```php
<?php
$txt = "W3Schools.com";
echo "I love " . $txt . "!";
?>
```

The following example will output the sum of two variables:

Example

```php
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

Note: You will learn more about the echo statement and how to output data to the screen in the next chapter.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.In PHP 7, type declarations were added. This gives an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.You will learn more about strict and non-strict requirements, and data type declarations in the PHP Functions chapter.

PHP Variables Scope:

In PHP, variables can be declared anywhere in the script.The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope:

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

Variable with global scope:

```
<?php
$x = 5; // global scope

function myTest() {
    // using x inside this function will generate an error
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example

Variable with local scope:

```
<?php
function myTest() {
    $x = 5; // local scope
    echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

PHP The global Keyword

The `global` keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
   global $x, $y;
   $y = $x + $y;
}

myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
   $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

Example

```
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}

myTest();
myTest();
myTest();
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

Exercise:

Create a variable named txt and assign the value "Hello".

□ = "□";

PHP echo and print Statements

With PHP, there are two basic ways to get output: echo and print.

In this tutorial we use echo or print in almost every example. So, this chapter contains a little more info about those two output statements.

PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

The PHP echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

The PHP print Statement

The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Display Variables

The following example shows how to output text and variables with the print statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
print $x + $y;
?>
```

Learn PHP
Study PHP at W3Schools.com
9

PHP Data Types

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object

- NULL
- Resource

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

Example

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>
```

```
Hello world!
Hello world!
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$x = 5985;
var_dump($x);
?>
```

```
int(5985)
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$x = 10.365;
var_dump($x);
?>
```

```
float(10.365)
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

An array stores multiple values in one single variable.In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

Example

```
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }You will learn a lot more about arrays in later chapters of this tutorial.

PHP Object

An object is a data type which stores data and information on how to process that data.In PHP, an object must be explicitly declared.First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
    function Car() {
        $this->model = "VW";
    }
}

// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

```
VW
```

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

Example

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

```
NULL
```

PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.A common example of using the resource data type is a database call.We will not talk about the resource type here, since it is an advanced topic.

Flow control in PHP

PHP if statement

The if statement has the following general form:

```
if (expression)
    statement
```

The if keyword is used to check if an expression is true. If it is true, a statement is then executed. The statement can be a single statement or a compound statement. A compound statement consists of multiple statements enclosed by curly brackets.

ifstatement.php

```
<?php

$num = 31;

if ($num > 0)
    echo "\$num variable is positive\n";
```

We have a $num variable. It is assigned value 31. The if keyword checks for a boolean expression. The expression is put between square brackets. The expression 31 > 0 is true, so the next statement is executed. Curly brackets are optional if there is only one statement to execute.

```
$ php ifstatement.php
$num variable is positive
```

This is the output of the example.

ifstatement2.php

```
<?php

$num = 31;

if ($num > 0) {
    echo "\$num variable is positive\n";
    echo "\$num variable equals to $num\n";
}
```

If we intend to execute more than one statement, we have to put them inside square brackets. If we did not use them, only the first statement would be executed. Curly brackets form the *body* of the if statement.

We can use the else keyword to create a simple branch. If the expression inside the square brackets following the if keyword evaluates to false, the statement inside the else body is automatically executed.

boyorgirl.php

```
<?php

$sex = "female";
```

```
if ($sex == "male") {
  echo "It is a boy\n";
} else {
  echo "It is a girl\n";
}
```

We have a $sex variable. It has "female" string. The boolean expression evaluates to false and we get "It is a girl" in the console.

```
$ php boyorgirl.php
It is a girl
```

This is the output of the example.

We can create multiple branches using the elseif keyword. The elseif keyword tests for another condition if and only if the previous condition was not met. Note that we can use multiple elseif keywords in our tests.

ifelsestm.php

```
<?php

echo "Enter a number: ";

$a = intval(fgets(STDIN));

if ($a < 0) {
   printf("%d is a negative number\n", $a);
} elseif ($a == 0) {
   printf("%d is a zero\n", $a);
} elseif ($a > 0) {
   printf("%d is a positive number\n", $a);
}
```

We read a value from the user using the fgets() function. The value is tested if it is a negative number or positive or if it equals to zero.

```
$ php ifelsestm.php
Enter a number: 4
4 is a positive number
$ php ifelsestm.php
Enter a number: -3
-3 is a negative number
```

Sample output of the program.

PHP switch statement

The switch statement is a selection control flow statement. It allows the value of a variable or expression to control the flow of program execution via a multiway branch. It creates multiple branches in a simpler way than using the if, elseif statements.

The switch statement works with two other keywords: case and break. The case keyword is used to test a label against a value from the round brackets. If the label equals to the value, the statement following the case is executed. The break keyword is used to jump out of the switch statement. There is an optional default statement. If none of the labels equals the value, the default statement is executed.

domains.php

```
<?php

$domain = 'sk';

switch ($domain) {

    case 'us':
        echo "United States\n";
    break;
    case 'de':
        echo "Germany\n";
    break;
    case 'sk':
        echo "Slovakia\n";
    break;
    case 'hu':
        echo "Hungary\n";
    break;
    default:
        echo "Unknown\n";
    break;
}
```

In our script, we have a *$domains* variable. It has the 'sk' string. We use the switch statement to test for the value of the variable. There are several options. If the value equals to 'us' the 'United States' string is printed to the console.

```
$ php domains.php
Slovakia
```

We get 'Slovakia'. If we changed the *$domains* variable to 'rr', we would get 'Unknown' string.

PHP while loop

The while is a control flow statement that allows code to be executed repeatedly based on a given boolean condition.

This is the general form of the while loop:

```
while (expression):
    statement
```

The while loop executes the statement when the expression is evaluated to true. The statement is a simple statement terminated by a semicolon or a compound statement enclosed in curly brackets.

whilestm.php

```
<?php

$i = 0;

while ($i < 5) {
    echo "PHP language\n";
    $i++;
}
```

In the code example, we repeatedly print "PHP language" string to the console.

The while loop has three parts: initialization, testing, and updating. Each execution of the statement is called a cycle.

```
$i = 0;
```

We initiate the $i variable. It is used as a counter in our script.

```
while ($i < 5) {
   ...
}
```

The expression inside the square brackets is the second phase, the testing. The while loop executes the statements in the body until the expression is evaluated to false.

```
$i++;
```

The last, third phase of the while loop is the updating; a counter is incremented. Note that improper handling of the while loop may lead to endless cycles.

```
$ php whilestm.php
PHP language
PHP language
PHP language
PHP language
PHP language
```

The program prints a message five times to the console.

The do while loop is a version of the while loop. The difference is that this version is guaranteed to run at least once.

dowhile.php

```
<?php

$count = 0;

do {
    echo "$count\n";
} while ($count != 0)
```

First the iteration is executed and then the truth expression is evaluated.

The while loop is often used with the list() and each() functions.

seasons.php

```
<?php

$seasons = ["Spring", "Summer", "Autumn", "Winter"];

while (list($idx , $val) = each($seasons)) {
    echo "$val\n";
}
```

We have four seasons in a $seasons array. We go through all the values and print them to the console. The each() function returns the current key and value pair from an array and advances the array cursor. When the function reaches the end of the array, it returns false and the loop is terminated. The each() function returns an array. There must be an array on the left side of the assignment too. We use the list() function to create an array from two variables.

```
$ php seasons.php
Spring
Summer
Autumn
Winter
```

This is the output of the seasons.php script.

PHP for keyword

The for loop does the same thing as the while loop. Only it puts all three phases, initialization, testing and updating into one place, between the round brackets. It is mainly used when the number of iteration is know before entering the loop.

Let's have an example with the for loop.

forloop.php

```
<?php

$days = [ "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
          "Saturday", "Sunday" ];

$len = count($days);

for ($i = 0; $i < $len; $i++) {
    echo $days[$i], "\n";
}
```

We have an array of days of a week. We want to print all these days from this array.

```
$len = count($days);
```

Or we can programmatically figure out the number of items in an array.

```
for ($i = 0; $i < $len; $i++) {
   echo $days[$i], "\n";
}
```

Here we have the for loop construct. The three phases are divided by semicolons. First, the $i counter is initiated. The initiation part takes place only once. Next, the test is conducted. If the result of the test is true, the statement is executed. Finally, the counter is incremented. This is one cycle. The for loop iterates until the test expression is false.

```
$ php forloop.php
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

This is the output of the forloop.php script.

PHP foreach statement

The foreach construct simplifies traversing over collections of data. It has no explicit counter. The foreach statement goes through the array one by one and the current value is copied to a variable defined in the construct. In PHP, we can use it to traverse over an array.

foreachstm.php

```
<?php

$planets = [ "Mercury", "Venus", "Earth", "Mars", "Jupiter",
             "Saturn", "Uranus", "Neptune" ];

foreach ($planets as $item) {
    echo "$item ";
}

echo "\n";
```

In this example, we use the foreach statement to go through an array of planets.

```
foreach ($planets as $item) {
    echo "$item ";
}
```

The usage of the foreach statement is straightforward. The $planets is the array that we iterate through. The $item is the temporary variable that has the current value from the array.

The foreach statement goes through all the planets and prints them to the console.

```
$ php foreachstm.php
Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune
```

Running the above PHP script gives this output.

There is another syntax of the foreach statement. It is used with maps.

foreachstm2.php

```
<?php

$benelux =  [ 'be' => 'Belgium',
          'lu' => 'Luxembourgh',
          'nl' => 'Netherlands' ];

foreach ($benelux as $key => $value) {
    echo "$key is $value\n";
}
```

In our script, we have a $benelux map. It contains domain names mapped to the benelux states. We traverse the array and print both keys and their values to the console.

```
$ php foreachstm2.php
be is Belgium
lu is Luxembourgh
nl is Netherlands
```

This is the outcome of the script.

PHP break, continue statements

The break statement is used to terminate the loop. The continue statement is used to skip a part of the loop and continue with the next iteration of the loop.

testbreak.php

```
<?php

while (true) {
    $val = rand(1, 30);
    echo $val, " ";
    if ($val == 22) break;
}

echo "\n";
```

We define an endless while loop. There is only one way to jump out of a such loop—using the break statement. We choose a random value from 1 to 30 and print it. If the value equals to 22, we finish the endless while loop.

```
$ php testbreak.php
6 11 13 5 5 21 9 1 21 22
```

We might get something like this.

In the following example, we print a list of numbers that cannot be divided by 2 without a remainder.

testcontinue.php

```
<?php

$num = 0;

while ($num < 1000) {

    $num++;
    if (($num % 2) == 0) continue;

    echo "$num ";

}

echo "\n";
```

We iterate through numbers 1..999 with the while loop.

```
if (($num % 2) == 0) continue;
```

If the expression $num % 2 returns 0, the number in question can be divided by 2. The continue statement is executed and the rest of the cycle is skipped. In our case, the last statement of the loop is skipped and the number is not printed to the console. The next iteration is started.

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

- Conditional assignment operators

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x – y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	**Name**	**Example**	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.The PHP decrement operators are used to decrement a variable's value.

Operator	**Name**	**Description**
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
And	And	$x and $y	True if both $x and $y are true
Or	Or	$x or $y	True if either $x or $y is true
Xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types

!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

PHP Strings

A string is a sequence of characters, like "Hello world!".

PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

strlen() - Return the Length of a String

The PHP strlen() function returns the length of a string.

Example

Return the length of the string "Hello world!":

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

str_word_count() - Count Words in a String

The PHP str_word_count() function counts the number of words in a string.

Example

Count the number of word in the string "Hello world!":

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

strrev() - Reverse a String

The PHP strrev() function reverses a string.

Example

Reverse the string "Hello world!":

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

strpos() - Search For a Text Within a String

The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

Example

Search for the text "world" in the string "Hello world!":

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

Tip: The first character position in a string is 0 (not 1).

str_replace() - Replace Text Within a String

The PHP str_replace() function replaces some characters with some other characters in a string.

Example

Replace the text "world" with "Dolly":

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

PHP Arrays

An array stores multiple values in one single variable:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the `array()` function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

Get The Length of an Array - The count() Function

The `count()` function is used to return the length (the number of elements) of an array:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

Exercise:

Use the correct function to output the number of items in an array.

```
$fruits = array("Apple", "Banana", "Orange");
echo □;
```

PHP Functions

The real power of PHP comes from its functions.PHP has more than 1000 built-in functions, and in addition you can create your own custom functions

PHP Built-in Functions

PHP has over 1000 built-in functions that can be called directly, from within a script, to perform a specific task.Please check out our PHP reference for a complete overview of the PHP built-in functions.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function

Create a User Defined Function in PHP

A user-defined function declaration starts with the word `function`:

Syntax

```
function functionName() {
  code to be executed;
}
```

Note: A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Tip: Give the function a name that reflects what the function does!

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code, and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name followed by brackets ():

Example

```
<?php
function writeMsg() {
  echo "Hello world!";
}

writeMsg(); // call the function
?>
```

```
Hello world!
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

```
Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.
```

The following example has a function with two arguments ($fname and $year):

Example

```
<?php
function familyName($fname, $year) {
   echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983");
?>
```

```
Hege Refsnes. Born in 1975
Stale Refsnes. Born in 1978
Kai Jim Refsnes. Born in 1983
```

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatch.

In the following example we try to send both a number and a string to the function without using strict:

Example

```
<?php
function addNumbers(int $a, int $b) {
   return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to int(5), and it will return 10
?>
```

```
10
```

To specify strict we need to set declare(strict_types=1);. This must be on the very first line of the PHP file.

In the following example we try to send both a number and a string to the function, but here we have added the strict declaration:

Example

```
<?php declare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
    return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

The strict declaration forces things to be used in the intended way.

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php declare(strict_types=1); // strict requirement
function setHeight(int $minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

PHP Functions - Returning values

To let a function return a value, use the return statement:

Example

```
<?php declare(strict_types=1); // strict requirement
function sum(int $x, int $y) {
   $z = $x + $y;
   return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

```
5 + 10 = 15
7 + 13 = 20
2 + 4 = 6
```

PHP Return Type Declarations

PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

To declare a type for the function return, add a colon (:) and the type right before the opening curly ({)bracket when declaring the function.

In the following example we specify the return type for the function:

Example

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
   return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

```
6.4
```

You can specify a different return type, than the argument types, but make sure the return is the correct type:

Example

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
  return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

6

Exercise:

Create a function named myFunction.

```
□ ` {
 echo "Hello World!";
}
```

PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!

When you are manipulating files you must be very careful.

You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The PHP code to read the file and write it to the output buffer is as follows
(the readfile() function returns the number of bytes read on success):

Example

```php
<?php
echo readfile("webdictionary.txt");
?>
```

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper
Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language
SVG = Scalable Vector Graphics XML = EXtensible Markup Language236
```

The readfile() function is useful if all you want to do is open up a file and read its contents.

The next chapters will teach you more about file handling.

Unit 2-PHP

What is OOP?

From PHP5, you can also write PHP code in an object-oriented style.

Object-Oriented programming is faster and easier to execute.

PHP What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

PHP - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

class

Fruit

objects

Apple

Banana

Mango

Another example:

class

Car

objects

Volvo

Audi

Toyota

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Look at the next chapters to learn more about OOP.

Classes and Objects

A class is a template for objects, and an object is an instance of class.

OOP Case

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Define a Class

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods goes inside the braces:

Syntax

```
<?php
class Fruit {
  // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Define Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

Output

```
Apple
Banana
```

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

Output

```
Name: Apple
Color: Red
```

PHP - The $this Keyword

The $this keyword refers to the current object, and is only available inside methods.

Look at the following example:

Example

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

So, where can we change the value of the $name property? There are two ways:

1. Inside the class (by adding a set_name() method and use $this):

Example

```
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

2.Outside the class (by directly change the property value):

Example

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
$apple->name = "Apple";
?>
```

PHP - instanceof

You can use the `instanceof` keyword to check if an object belongs to a specific class:

Example

```
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

```
bool(true)
```

Constructor

PHP - The __construct Function

A constructor allows you to initialize an object's properties upon creation of the object.

If you create a __construct() function, PHP will automatically call this function when you create an object from a class.

Notice that the construct function starts with two underscores (__)!

We see in the example below, that using a constructor saves us from calling the set_name() method which reduces the amount of code:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

```
Apple
```

Another example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

```
Apple
red
```

Destructor

PHP - The __destruct Function

A destructor is called when the object is destructed or the script is stopped or exited.

If you create a `__destruct()` function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (__)!

The example below has a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

The fruit is Apple.

Another example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function __destruct() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

$apple = new Fruit("Apple", "red");
?>
```

The fruit is Apple and the color is red.

Access Modifiers

PHP - Access Modifiers

Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

- public - the property or method can be accessed from everywhere. This is default
- protected - the property or method can be accessed within the class and by classes derived from that class
- private - the property or method can ONLY be accessed within the class

In the following example we have added three different access modifiers to the three properties. Here, if you try to set the name property it will work fine (because the name property is public). However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private):

Example

```
<?php
class Fruit {
  public $name;
  protected $color;
  private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

In the next example we have added access modifiers to two methods. Here, if you try to call the set_color() or the set_weight() function it will result in a fatal error (because the two functions are considered protected and private), even if all the properties are public:

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public $weight;
  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

Inheritance

PHP - What is Inheritance?

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the extends keyword.

Let's look at an example:

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

```
Am I a fruit or a berry? The fruit is Strawberry and the color is red.
```

Example Explained

The Strawberry class is inherited from the Fruit class.

This means that the Strawberry class can use the public $name and $color properties as well as the public __construct() and intro() methods from the Fruit class because of inheritance.

The Strawberry class also has its own method: message().

Class Constants

PHP - Class Constants

Constants cannot be changed once it is declared.

Class constants can be useful if you need to define some constant data within a class.

A class constant is declared inside a class with the const keyword.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (::) followed by the constant name, like here:

Example

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

```
Thank you for visiting W3Schools.com!
```

Or, we can access a constant from inside the class by using the self keyword followed by the scope resolution operator (::) followed by the constant name, like here:

Example

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

```
Thank you for visiting W3Schools.com!
```

Abstract Classes

PHP - What are Abstract Classes and Methods?

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the abstract keyword:

Syntax

```
<?php
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name, $color);
  abstract public function someMethod3() : string;
```

```php
}
?>
```

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same. However, the child class may has optional arguments in addition

Let's look at an example:

Example

```php
<?php
// Parent class
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "Choose German quality! I'm an $this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "Proud to be Swedish! I'm a $this->name!";
  }
}
```

```
class Citroen extends Car {
  public function intro() : string {
    return "French extravagance! I'm a $this->name!";
  }
}

// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>
```

```
Choose German quality! I'm an Audi!
Proud to be Swedish! I'm a Volvo!
French extravagance! I'm a Citroen!
```

Example Explained

The Audi, Volvo, and Citroen classes are inherited from the Car class. This means that the Audi, Volvo, and Citroen classes can use the public $name property as well as the public __construct() method from the Car class because of inheritance.

But, intro() is an abstract method that should be defined in all the child classes and they should return a string.

Traits

PHP - What are Traits?

PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the trait keyword:

Syntax

```
<?php
trait TraitName {
  // some code...
}
?>
```

To use a trait in a class, use the use keyword:

Syntax

```
<?php
class MyClass {
  use TraitName;
}
?>
```

Let's look at an example:

Example

```
<?php
trait message1 {
public function msg1() {
    echo "OOP is fun! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

```
OOP is fun!
```

Example Explained

Here, we declare one trait: message1. Then, we create a class: Welcome. The class uses the trait, and all the methods in the trait will be available in the class.

If other classes need to use the msg1() function, simply use the message1 trait in those classes. This reduces code duplication, because there is no need to redeclare the same method over and over again.

Static Methods

PHP - Static Methods

Static methods can be called directly - without creating an instance of a class.

Static methods are declared with the static keyword:

Syntax

```
<?php
class ClassName {
  public static function staticMethod() {
    echo "Hello World!";
  }
}
?>
```

To access a static method use the class name, double colon (::), and the method name:

Syntax

```
ClassName::staticMethod();
```

Let's look at an example:

Example

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

// Call static method
```

```php
greeting::welcome();
?>
```

```
Hello World!
```

Example Explained

Here, we declare a static method: welcome(). Then, we call the static method by using the class name, double colon (::), and the method name (without creating a class first).

PHP - More on Static Methods

A class can have both static and non-static methods. A static method can be accessed from a method in the same class using the self keyword and double colon (::):

Example

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }

  public function __construct() {
    self::welcome();
  }
}

new greeting();
?>
```

```
Hello World!
```

Static methods can also be called from methods in other classes. To do this, the static method should be public:

Example

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}
```

```
class SomeOtherClass {
  public function message() {
    greeting::welcome();
  }
}
?>
```

To call a static method from a child class, use the parent keyword inside the child class. Here, the static method can be public or protected.

Example

```
<?php
class domain {
  protected static function getWebsiteName() {
    return "W3Schools.com";
  }
}

class domainW3 extends domain {
  public $websiteName;
  public function __construct() {
    $this->websiteName = parent::getWebsiteName();
  }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

```
W3Schools.com
```

Static Properties

PHP - Static Properties

Static properties can be called directly - without creating an instance of a class.

Static properties are declared with the static keyword:

Syntax

```
<?php
class ClassName {
  public static $staticProp = "W3Schools";
}
?>
```

To access a static property use the class name, double colon (::), and the property name:

Syntax

```
ClassName::staticProp();
```

Let's look at an example:

Example

```
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

```
3.14159
```

Example Explained

Here, we declare a static property: $value. Then, we echo the value of the static property by using the class name, double colon (::), and the property name (without creating a class first).

PHP - More on Static Properties

A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the self keyword and double colon (::):

Example

```
<?php
class pi {
  public static $value=3.14159;
  public function staticValue() {
    return self::$value;
  }
}

$pi = new pi();
echo $pi->staticValue();
?>
```

```
3.14159
```

To call a static property from a child class, use the parent keyword inside the child class:

Example

```
<?php
class pi {
  public static $value=3.14159;
}

class x extends pi {
  public function xStatic() {
    return parent::$value;
  }
}

// Get value of static property directly via child class
echo x::$value;

// or get value of static property via xStatic() method
$x = new x();
echo $x->xStatic();
?>
```

```
3.141593.14159
```

Unit IV

PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!

When you are manipulating files you must be very careful.

You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The PHP code to read the file and write it to the output buffer is as follows
(the readfile() function returns the number of bytes read on success):

Example

```php
<?php
echo readfile("webdictionary.txt");
?>
```

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper
Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language
SVG = Scalable Vector Graphics XML = EXtensible Markup Language236
```

The readfile() function is useful if all you want to do is open up a file and read its contents.

The next chapters will teach you more about file handling.

PHP Exercises

Exercise:

Assume we have a file named "webdict.txt", write the correct syntax to open and read the file content.

```
echo □;
```

File Open/Read/Close

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```php
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper
Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language
SVG = Scalable Vector Graphics XML = EXtensible Markup Language
```

Tip: The fread() and the fclose() functions will be explained below.

The file may be opened in one of the following modes:

Modes	**Description**
r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist.
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist.
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

AJAX = Asynchronous JavaScript and XML

Note: After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP Exercises

Exercise:

Open a file, and write the correct syntax to output one character at the time, until end-of-file.

```
$myfile = fopen("webdict.txt", "r");
while(!feof($myfile)) {
  echo fgetc($myfile);
}
```

File Create/Write

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

```
John Doe
Jane Doe
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```php
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

```
Mickey Mouse
Minnie Mouse
```

File Open/Read/Close

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language

Tip: The fread() and the fclose() functions will be explained below.

The file may be opened in one of the following modes:

Modes	**Description**
r	**Open a file for read only**. File pointer starts at the beginning of the file

w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist.
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist.
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

AJAX = Asynchronous JavaScript and XML

Note: After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper
Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language
SVG = Scalable Vector Graphics XML = EXtensible Markup Language
```

Complete PHP Filesystem Reference

For a complete reference of filesystem functions, go to our complete PHP Filesystem Reference.

PHP Exercises

Exercise:

Open a file, and write the correct syntax to output one character at the time, until end-of-file.

```
$myfile = fopen("webdict.txt", "r");
while(!feof($myfile)) {
  echo fgetc($myfile);
}
```

File Create/Write

In this chapter we will teach you how to create and write to a file on the server.

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

```
John Doe
Jane Doe
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

```
Mickey Mouse
Minnie Mouse
```

File Upload

With PHP, it is easy to upload files to the server.

However, with ease comes danger, so always be careful when allowing file uploads!

Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file_uploads directive, and set it to On:

```
file_uploads = On
```

Create The HTML Form

Next, create an HTML form that allow users to choose the image file they want to upload:

```
<!DOCTYPE html>
<html>
<body>

<form action="upload.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"
- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next.

Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
```

```
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
   $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
   if($check !== false) {
      echo "File is an image - " . $check["mime"] . ".";
      $uploadOk = 1;
   } else {
      echo "File is not an image.";
      $uploadOk = 0;
   }
}
?>
```

PHP script explained:

- $target_dir = "uploads/" - specifies the directory where the file is going to be placed
- $target_file specifies the path of the file to be uploaded
- $uploadOk=1 is not used yet (will be used later)
- $imageFileType holds the file extension of the file (in lower case)
- Next, check if the image file is an actual image or a fake image

Note: You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

Check if File Already Exists

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
   echo "Sorry, file already exists.";
   $uploadOk = 0;
}
```

Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500KB, an error message is displayed, and $uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

Complete Upload File PHP Script

The complete "upload.php" file now looks like this:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

Cookies

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
```

```
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
```

```
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

heck if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Sessions

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the `session_start()` function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
```

```
</body>
</html>
```

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

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

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

Example

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

```
<?php
print_r($_SESSION);
?>

</body>
</html>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

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

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example

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

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

Filters

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

The PHP Filter Extension

PHP filters are used to validate and sanitize external input.

The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

The filter_list() function can be used to list what the PHP filter extension offers:

Example

```
<table>
  <tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
  </tr>
  <?php
  foreach (filter_list() as $id =>$filter) {
      echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) . '</td></tr>';
  }
  ?>
</table>
```

Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form
- Cookies
- Web services data
- Server variables
- Database query results

You should always validate external data!
Invalid submitted data can lead to security problems and break your webpage!
By using PHP filters you can be sure your application gets the correct input!

PHP filter_var() Function

The filter_var() function both validate and sanitize data.

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check
- The type of check to use

Sanitize a String

The following example uses the filter_var() function to remove all HTML tags from a string:

Example

```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Try it Yourself »

Validate an Integer

The following example uses the filter_var() function to check if the variable $int is an integer. If $int is an integer, the output of the code below will be: "Integer is valid". If $int is not an integer, the output will be: "Integer is not valid":

Example

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
   echo("Integer is valid");
} else {
   echo("Integer is not valid");
}
?>
```

Try it Yourself »

Tip: filter_var() and Problem With 0

In the example above, if $int was set to 0, the function above will return "Integer is not valid". To solve this problem, use the code below:

Example

```
<?php
$int = 0;
```

```
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int,
FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Try it Yourself »

Validate an IP Address

The following example uses the filter_var() function to check if the variable $ip is a valid IP address:

Example

```
<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
    echo("$ip is a valid IP address");
} else {
    echo("$ip is not a valid IP address");
}
?>
```

Try it Yourself »

Sanitize and Validate an Email Address

The following example uses the filter_var() function to first remove all illegal characters from the $email variable, then check if it is a valid email address:

Example

```
<?php
$email = "john.doe@example.com";
```

```
// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

Try it Yourself »

Sanitize and Validate a URL

The following example uses the filter_var() function to first remove all illegal characters from a URL, then check if $url is a valid URL:

Example

```
<?php
$url = "https://www.w3schools.com";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

// Validate url
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>
```

Try it Yourself »

Complete PHP Filter Reference

For a complete reference of all filter functions, go to our complete PHP Filter Reference. Check each filter to see what options and flags are available.

The reference contains a brief description, and examples of use, for each function!

Filters Advanced

Validate an Integer Within a Range

The following example uses the filter_var() function to check if a variable is both of type INT, and between 1 and 200:

Example

```
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int,
FILTER_VALIDATE_INT, array("options" => array("min_range"=>$min, "max_range"=>$ma
x))) === false) {
   echo("Variable value is not within the legal range");
} else {
   echo("Variable value is within the legal range");
}
?>
```

Validate IPv6 Address

The following example uses the filter_var() function to check if the variable $ip is a valid IPv6 address:

Example

```
<?php
$ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) === false) {
   echo("$ip is a valid IPv6 address");
} else {
   echo("$ip is not a valid IPv6 address");
}
?>
```

Try it Yourself »

Validate URL - Must Contain QueryString

The following example uses the filter_var() function to check if the variable $url is a URL with a querystring:

Example

```
<?php
$url = "https://www.w3schools.com";

if (!filter_var($url, FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED) ===
false) {
   echo("$url is a valid URL with a query string");
} else {
   echo("$url is not a valid URL with a query string");
}
?>
```

Try it Yourself »

Remove Characters With ASCII Value > 127

The following example uses the filter_var() function to sanitize a string. It will both remove all HTML tags, and all characters with ASCII value > 127, from the string:

Example

```
<?php
$str = "<h1>Hello WorldÆØÅ!</h1>";

$newstr = filter_var($str, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_HIGH);
```

```
echo $newstr;
?>
```

Try it Yourself »

Complete PHP Filter Reference

For a complete reference of all filter functions, go to our complete PHP Filter Reference. Check each filter to see what options and flags are available.

The reference contains a brief description, and examples of use, for each function!

PHP and JSON

What is JSON?

JSON stands for JavaScript Object Notation, and is a syntax for storing and exchanging data.

Since the JSON format is a text-based format, it can easily be sent to and from a server, and used as a data format by any programming language.

PHP and JSON

PHP has some built-in functions to handle JSON.

First, we will look at the following two functions:

- json_encode()
- json_decode()

PHP - json_encode()

The json_encode() function is used to encode a value to JSON format.

Example

This example shows how to encode an associative array into a JSON object:

```
<?php
$age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);

echo json_encode($age);
?>
```

Run Example »

Example

This example shows how to encode an indexed array into a JSON array:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");

echo json_encode($cars);
?>
```

Run Example »

PHP - json_decode()

The json_decode() function is used to decode a JSON object into a PHP object or an associative array.

Example

This example decodes JSON data into a PHP object:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj));
?>
```

Run Example »

The json_decode() function returns an object by default. The json_decode() function has a second parameter, and when set to true, JSON objects are decoded into associative arrays.

Example

This example decodes JSON data into a PHP associative array:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj, true));
?>
```

Run Example »

PHP - Accessing the Decoded Values

Here are two examples of how to access the decoded values from an object and from an associative array:

Example

This example shows how to access the values from a PHP object:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$obj = json_decode($jsonobj);

echo $obj->Peter;
echo $obj->Ben;
echo $obj->Joe;
?>
```

Run Example »

Example

This example shows how to access the values from a PHP associative array:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);
```

```
echo $arr["Peter"];
echo $arr["Ben"];
echo $arr["Joe"];
?>
```

Run Example »

PHP - Looping Through the Values

You can also loop through the values with a foreach() loop:

Example

This example shows how to loop through the values of a PHP object:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$obj = json_decode($jsonobj);

foreach($obj as $key => $value) {
  echo $key . " => " . $value . "<br>";
}
?>
```

Run Example »

Example

This example shows how to loop through the values of a PHP associative array:

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);

foreach($arr as $key => $value) {
  echo $key . " => " . $value . "<br>";
}
?>
```

PHP MySQL Database

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

PHP + MySQL Database System

- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Database Queries

A query is a question or a request.

We can query a database for specific information and have a recordset returned.

Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

The query above selects all the data in the "LastName" column from the "Employees" table.

To learn more about SQL, please visit our SQL tutorial.

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download it for free here: http://www.mysql.com

Facts About MySQL Database

MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

Another great thing about MySQL is that it can be scaled down to support embedded database applications.

Look at http://www.mysql.com/customers/ for an overview of companies using MySQL.

PHP – AJAX

What is AJAX?

AJAX = Asynchronous JavaScript and XML.

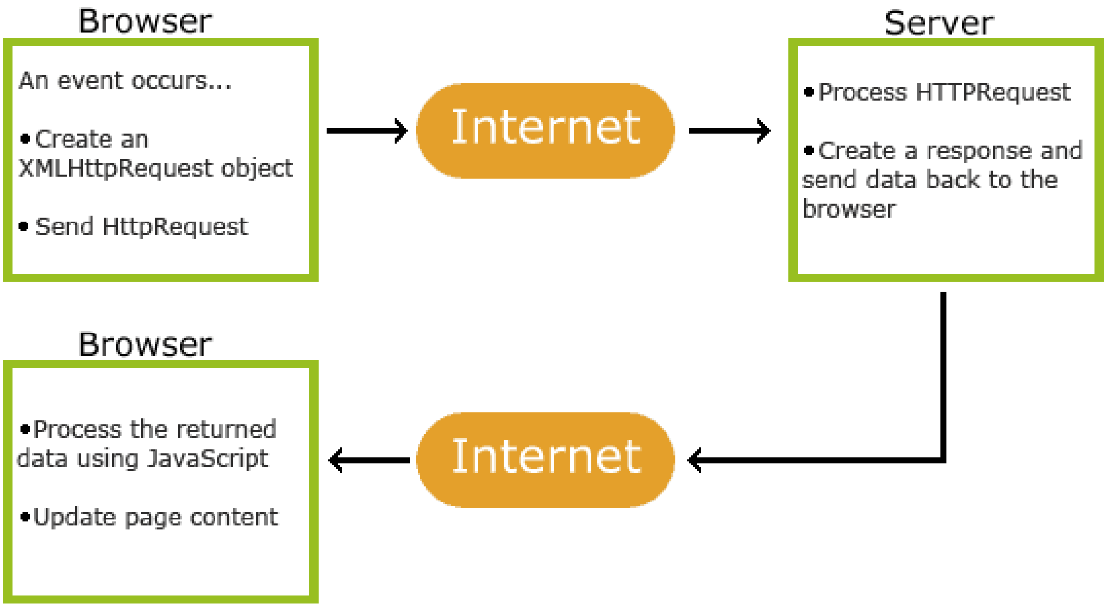
AJAX is a technique for creating fast and dynamic web pages.

AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.

Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

How AJAX Works


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

AJAX is Based on Internet Standards

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

AJAX applications are browser- and platform-independent!

Google Suggest

AJAX was made popular in 2005 by Google, with Google Suggest.

Google Suggest is using AJAX to create a very dynamic web interface: When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.

Start Using AJAX Today

In our PHP tutorial, we will demonstrate how AJAX can update parts of a web page, without reloading the whole page. The server script will be written in PHP.

If you want to learn more about AJAX, visit our AJAX tutorial.

AJAX is used to create more interactive applications.

AJAX PHP Example

The following example will demonstrate how a web page can communicate with a web server while a user type characters in an input field:

Example

Start typing a name in the input field below:

First name:

Suggestions:

Example Explained

In the example above, when a user types a character in the input field, a function called "showHint()" is executed.

The function is triggered by the onkeyup event.

Here is the HTML code:

Example

```
<html>
<head>
<script>
function showHint(str) {
  if (str.length == 0) {
    document.getElementById("txtHint").innerHTML = "";
    return;
  } else {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("txtHint").innerHTML = this.responseText;
      }
    };
    xmlhttp.open("GET", "gethint.php?q=" + str, true);
    xmlhttp.send();
  }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Start typing a name in the input field below:

First name: □

Suggestions:

Code explanation:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.

However, if the input field is not empty, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
- Notice that q parameter is added to the url (gethint.php?q="+str)

- And the str variable holds the content of the input field

The PHP File - "gethint.php"

The PHP file checks an array of names, and returns the corresponding name(s) to the browser:

```
<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
```

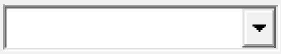
```
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";

// lookup all hints from array if $q is different from ""
if ($q !== "") {
   $q = strtolower($q);
   $len=strlen($q);
   foreach($a as $name) {
      if (stristr($q, substr($name, 0, $len))) {
         if ($hint === "") {
            $hint = $name;
         } else {
            $hint .= ", $name";
         }
      }
   }
}

// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;
?>
```

AJAX and MySQL

AJAX Database Example

The following example will demonstrate how a web page can fetch information from a database with AJAX:

Example



Person info will be listed here...

Example Explained - The MySQL Database

The database table we use in the example above looks like this:

id	FirstName	LastName	Age	Hometown
1	Peter	Griffin	41	Quahog
2	Lois	Griffin	40	Newport
3	Joseph	Swanson	39	Quahog
4	Glenn	Quagmire	41	Quahog

Example Explained

In the example above, when a user selects a person in the dropdown list above, a function called "showUser()" is executed.

The function is triggered by the onchange event.

Here is the HTML code:

Example

```
<html>
<head>
<script>
function showUser(str) {
    if (str == "") {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        if (window.XMLHttpRequest) {
            // code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
        } else {
            // code for IE6, IE5
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET","getuser.php?q="+str,true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<form>
<select name="users" onchange="showUser(this.value)">
  <option value="">Select a person:</option>
  <option value="1">Peter Griffin</option>
  <option value="2">Lois Griffin</option>
  <option value="3">Joseph Swanson</option>
  <option value="4">Glenn Quagmire</option>
  </select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here...</b></div>

</body>
</html>
```

Run example »

Code explanation:

First, check if person is selected. If no person is selected (str == ""), clear the content of txtHint and exit the function. If a person is selected, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getuser.php".

The source code in "getuser.php" runs a query against a MySQL database, and returns the result in an HTML table:

```
<!DOCTYPE html>
<html>
<head>
<style>
table {
    width: 100%;
    border-collapse: collapse;
}

table, td, th {
    border: 1px solid black;
    padding: 5px;
}

th {text-align: left;}
</style>
</head>
<body>

<?php
$q = intval($_GET['q']);

$con = mysqli_connect('localhost','peter','abc123','my_db');
```

```
if (!$con) {
  die('Could not connect: ' . mysqli_error($con));
}

mysqli_select_db($con,"ajax_demo");
$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysqli_query($con,$sql);

echo "<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
while($row = mysqli_fetch_array($result)) {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
  echo "<td>" . $row['Hometown'] . "</td>";
  echo "<td>" . $row['Job'] . "</td>";
  echo "</tr>";
}
echo "</table>";
mysqli_close($con);
?>
</body>
</html>
```

Explanation: When the query is sent from the JavaScript to the PHP file, the following happens:

1. PHP opens a connection to a MySQL server
2. The correct person is found
3. An HTML table is created, filled with data, and sent back to the "txtHint" placeholder

AJAX and XML

AJAX can be used for interactive communication with an XML file.

AJAX XML Example

The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

Example

CD info will be listed here...

Example Explained - The HTML Page

When a user selects a CD in the dropdown list above, a function called "showCD()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showCD(str) {
  if (str=="") {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {
      document.getElementById("txtHint").innerHTML=this.responseText;
    }
  }
  xmlhttp.open("GET","getcd.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
<option value="">Select a CD:</option>
<option value="Bob Dylan">Bob Dylan</option>
<option value="Bee Gees">Bee Gees</option>
<option value="Cat Stevens">Cat Stevens</option>
</select>
</form>
<div id="txtHint"><b>CD info will be listed here...</b></div>

</body>
</html>
```

The showCD() function does the following:

- Check if a CD is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getcd.php".

The PHP script loads an XML document, "cd_catalog.xml", runs a query against the XML file, and returns the result as HTML:

```
<?php
$q=$_GET["q"];
```

```
$xmlDoc = new DOMDocument();
$xmlDoc->load("cd_catalog.xml");

$x=$xmlDoc->getElementsByTagName('ARTIST');

for ($i=0; $i<=$x->length-1; $i++) {
  //Process only element nodes
  if ($x->item($i)->nodeType==1) {
    if ($x->item($i)->childNodes->item(0)->nodeValue == $q) {
      $y=($x->item($i)->parentNode);
    }
  }
}

$cd=($y->childNodes);

for ($i=0;$i<$cd->length;$i++) {
  //Process only element nodes
  if ($cd->item($i)->nodeType==1) {
    echo("<b>" . $cd->item($i)->nodeName . ":</b> ");
    echo($cd->item($i)->childNodes->item(0)->nodeValue);
    echo("<br>");
  }
}
?>
```

When the CD query is sent from the JavaScript to the PHP page, the following happens:

1. PHP creates an XML DOM object
2. Find all <artist> elements that matches the name sent from the JavaScript
3. Output the album information (send to the "txtHint" placeholder)

AJAX Live Search

AJAX Live Search

The following example will demonstrate a live search, where you get search results while you type.

Live search has many benefits compared to traditional searching:

- Results are shown as you type
- Results narrow as you continue typing

- If results become too narrow, remove characters to see a broader result

Search for a W3Schools page in the input field below:

The results in the example above are found in an XML file (links.xml). To make this example small and simple, only six results are available.

Example Explained - The HTML Page

When a user types a character in the input field above, the function "showResult()" is executed. The function is triggered by the "onkeyup" event:

```
<html>
<head>
<script>
function showResult(str) {
  if (str.length==0) {
    document.getElementById("livesearch").innerHTML="";
    document.getElementById("livesearch").style.border="0px";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {
      document.getElementById("livesearch").innerHTML=this.responseText;
      document.getElementById("livesearch").style.border="1px solid #A5ACB2";
    }
  }
  xmlhttp.open("GET","livesearch.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
```

```
<body>

<form>
<input type="text" size="30" onkeyup="showResult(this.value)">
<div id="livesearch"></div>
</form>

</body>
</html>
```

Source code explanation:

If the input field is empty (str.length==0), the function clears the content of the livesearch placeholder and exits the function.

If the input field is not empty, the showResult() function executes the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the input field)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "livesearch.php".

The source code in "livesearch.php" searches an XML file for titles matching the search string and returns the result:

```
<?php
$xmlDoc=new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');

//get the q parameter from URL
$q=$_GET["q"];

//lookup all links from the xml file if length of q>0
```

```
if (strlen($q)>0) {
  $hint="";
  for($i=0; $i<($x->length); $i++) {
    $y=$x->item($i)->getElementsByTagName('title');
    $z=$x->item($i)->getElementsByTagName('url');
    if ($y->item(0)->nodeType==1) {
      //find a link matching the search text
      if (stristr($y->item(0)->childNodes->item(0)->nodeValue,$q)) {
        if ($hint=="") {
          $hint="<a href='" .
          $z->item(0)->childNodes->item(0)->nodeValue .
          "' target='_blank'>" .
          $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
        } else {
          $hint=$hint . "<br /><a href='" .
          $z->item(0)->childNodes->item(0)->nodeValue .
          "' target='_blank'>" .
          $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
        }
      }
    }
  }
}

// Set output to "no suggestion" if no hint was found
// or to the correct values
if ($hint=="") {
  $response="no suggestion";
} else {
  $response=$hint;
}

//output the response
echo $response;
?>
```

If there is any text sent from the JavaScript (strlen($q) > 0), the following happens:

- Load an XML file into a new XML DOM object
- Loop through all <title> elements to find matches from the text sent from the JavaScript

- Sets the correct url and title in the "$response" variable. If more than one match is found, all matches are added to the variable
- If no matches are found, the $response variable is set to "no suggestion"

AJAX Poll

AJAX Poll

The following example will demonstrate a poll where the result is shown without reloading.

Do you like PHP and AJAX so far?

Yes: ○

No: ○

Example Explained - The HTML Page

When a user chooses an option above, a function called "getVote()" is executed. The function is triggered by the "onclick" event:

```
<html>
<head>
<script>
function getVote(int) {
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {
      document.getElementById("poll").innerHTML=this.responseText;
    }
  }
  xmlhttp.open("GET","poll_vote.php?vote="+int,true);
  xmlhttp.send();
}
</script>
</head>
```

```
<body>

<div id="poll">
<h3>Do you like PHP and AJAX so far?</h3>
<form>
Yes:
<input type="radio" name="vote" value="0" onclick="getVote(this.value)">
<br>No:
<input type="radio" name="vote" value="1" onclick="getVote(this.value)">
</form>
</div>

</body>
</html>
```

The getVote() function does the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (vote) is added to the URL (with the value of the yes or no option)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "poll_vote.php":

```
<?php
$vote = $_REQUEST['vote'];

//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);

//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];

if ($vote == 0) {
  $yes = $yes + 1;
}
```

```
if ($vote == 1) {
  $no = $no + 1;
}

//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>

<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($yes/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($no/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

The value is sent from the JavaScript, and the following happens:

1. Get the content of the "poll_result.txt" file
2. Put the content of the file in variables and add one to the selected variable
3. Write the result to the "poll_result.txt" file
4. Output a graphical representation of the poll result

The Text File

The text file (poll_result.txt) is where we store the data from the poll.

It is stored like this:

0||0

The first number represents the "Yes" votes, the second number represents the "No" votes.

Note: Remember to allow your web server to edit the text file. Do **NOT** give everyone access, just the web server (PHP).

UNIT-III

Introduction to web service:

A **web service** makes software application resources available over networks using standard **technologies**. Because **web services** are based on standard interfaces, they can communicate even if they are running on different operating systems and are written in different language.

A web service supports communication among numerous apps with HTML, XML, WSDL, SOAP, and other open standards. XML tags the data, SOAP transfers the message, and WSDL describes the service's accessibility.

SOAP WSDL UDDI:

SOAP defines a uniform way of passing XML-encoded data.

♦ WSDL allows service providers to specify what a web service can do, where it resides, and how to invoke it.

♦ UDDI provides a mechanism for clients to dynamically find other Web services.

In SOAP messages, the name of the service request and the input parameters take the form of XML elements.

WSDL describes the data types and structures for Web services, and tells how to map them into the messages that are exchanged.

UDDI provides a repository for Web-services descriptions. An UDDI registry can be searched on various criteria to find all kinds of services offered by businesses.

WSDL:

- WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
- WSDL definitions describe how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.

- WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.
- WSDL is the languagethat UDDI uses.

Example:

```
<definitionsname="HelloService"
targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   ................................................
</definitions>
```

UDDI:

UDDI is an XML-based standard for describing, publishing, and finding **web** services. **UDDI** stands for Universal Description, Discovery, and Integration. ... **UDDI** is an open industry initiative, enabling businesses to discover each other and define how they interact over the **Internet**.

UDDI has two sections −

- A registry of all web service's metadata, including a pointer to the WSDL description of a service.
- A set of WSDL port type definitions for manipulating and searching that registry.

Example:

```
<businessEntitybusinessKey="uuid:C0E6D5A8-C446-4f01-99DA-
70E212685A40"
operator="http://www.ibm.com"authorizedName="John Doe">
<name>Acme Company</name>
<description>
      We create cool Web services
</description>

<contacts>
```

```
<contactuseType="general info">
<description>General Information</description>
<personName>John Doe</personName>
<phone>(123) 123-1234</phone>
<email>jdoe@acme.com</email>
</contact>
</contacts>

<businessServices>
    ...
</businessServices>
<identifierBag>
<keyedReferencetModelKey="UUID:8609C81E-EE1F-4D5A-B202-
3EB13AD01823"
name="D-U-N-S"value="123456789"/>
</identifierBag>

<categoryBag>
<keyedReferencetModelKey="UUID:C0B9FE13-179F-413D-8A5B-
5004DB8E5BB2"
name="NAICS"value="111336"/>
</categoryBag>
</businessEntity>
```

Why web service is important:

Web services technology represents an **important** way for businesses to communicate with each other and with clients as well. ... Instead, **Web services** share business logic, data and processes through a programmatic interface across a network. The applications interface with each other, not with the users.

The evaluation of web applications:

A web application is software **application** that runs on a remote server. In most cases, **Web** browsers are used to access **Web applications**, over a network, such as the Internet. Some **web applications** are used in intranets, in companies and schools.

– Global approach. Web applications are published centralized in one place and the whole world can see them. Any user who has access to the Internet can access Web applications from a home computer.

– Simultaneous work of a large number of users. In general, traditional desktop applications are used by one user at a certain time,

Ability to work on multiple platforms. Most clients of Web applications are Web readers who play the role of a universal interface between the user and the system for displaying data of any format and can be run on any computer with the Internet access

Web services and enterprise:

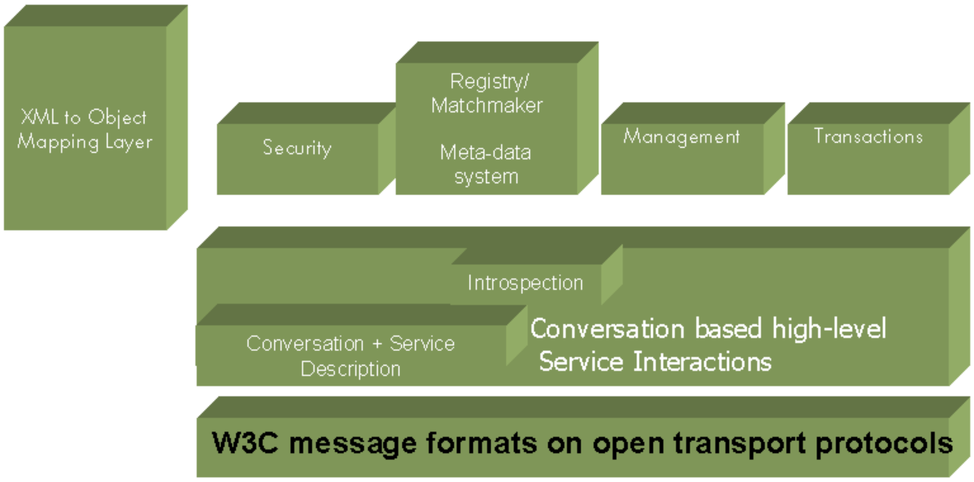
Enterprise web services can be considered to be a set of consistent, persistent, secure and REST full web services that allow developers to integrate and link applications and web pages that contain essential business systems. These services include application and software development that are serving businesses.

These APIs allow systems from across the UW to access data on demand in a scalable, real-time, and highly available architecture. The primary purpose for providing this data via APIs is to reduce duplicate shadow systems and the proliferation of sensitive business data.

The lingua franca of web services:

XML is a standard for data mark-up backed by the World Wide Web Consortium, which has been branded “the universal format for structured documents and data on the Web.”[1] The entire XML suite of standards, models, and processing technologies have been under development since 1998 with the initial XML specification, and has since been augmented by several additional supporting standards and notes that have brought XML to its current rich state. In fact, though XML is undeniably a richly specified technology, it has retained its simplicity and the entire XML platform can be profiled as follows.

Web Services Infrastructure Stack



XML to Object Mapping Layer

Security

Registry/ Matchmaker

Meta-data system

Management

Transactions

Introspection

Conversation + Service Description

Conversation based high-level Service Interactions

W3C message formats on open transport protocols

XML:

XML stands for eXtensible Markup Language,XML is a markup language much like HTML it was designed to store and transport data.XML was designed to be self-descriptive is a W3C Recommendation.

Syntax:

```
<note>
<date>2015-09-01</date>

<hour>08:30</hour>
<to>Tove</to>
<from>Jani</from>
```

```
<body>Don't forget me this weekend!</body>
</note>
```

XML file:

In the XML file structure, the first line is the **declaration** that contains the document information like version information, character set, etc.Declaration must come first in the document.Every opening tag should have a closing tag.Each XML document has a **root element**.

Syntax:

```
<?xmlversion="1.0"encoding="UTF-8"?>

<note>

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
  </note>
```

Example:

```
<?xml version = “1.0” encoding=”UTF-8”?> ----Declaration

<users> --------------------------------Root Element
     <user> ----------------------------Child Element
         <firstname>Brad</firstname>---Sub
         <lastname>Traversy</lastname>
         <email>techguyinfo@gmail.com</email>---Child
              <phone>555-555-5555</phone> ------  Elements
        </user>
        <user gender=”male”>
                <firstname>John</firstname>
                <lastname>Doe</lastname>
                <email>jdoe@gmail.com</email>
                <phone>444-444-4444</phone>
        </user>
</users>
```

XML Document:

An **XML document** is a basic unit of **XML** information composed of elements and other markup in an orderly package. An **XML document** can contains wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

XML Document Example:

```
<?xml version ="1.0"?>
<contact-info>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

XML namespace:

XML namespace. **XML namespaces** are used for providing uniquely named elements and attributes in an **XML** document. They are defined in a W3C recommendation. An **XML** instance may contain element or attribute names from more than one **XML** vocabulary.

Example

Employee document:

```
<employee>
<personInfo>
<empID>E0000001</empID>
<secID>S001</secID>
<name>John Smith</name>
</personInfo>
<personInfo>0
<empID>E0000002</empID>
<secID>S002</secID>
<name>Ichiro Tanaka</name>
</personInfo>
```

```
.
.
.
</employee>
```

Explicit and default namespace:

URIs are used simply because they are globally unique across the **Internet**. **Namespaces** can be declared either **explicitly** or by **default**. With an **explicit** declaration, you define a shorthand, or prefix, to substitute for the full name of the **namespace**. You use this prefix to qualify elements belonging to that **namespace**.

Example:

```
<BOOKS>

<bk:BOOK xmlns:bk="urn:example.microsoft.com:BookInfo"
         xmlns:money="urn:Finance:Money">
<bk:TITLE>Creepy Crawlies</bk:TITLE>
<bk:PRICE money:currency="US Dollar">22.95</bk:PRICE>
</bk:BOOK>
</BOOKS>
```

Attributes and namespace:

An **attribute** is a characteristic. In HTML, an **attribute** is a characteristic of a page element, such as font size or color. **Attributes** are used to amplify a tag. When a **Web** browser interprets an HTML tag, it will also look for its **attributes** so that it can display the **Web** page's elements properly.

XML schema:

XML Schema is commonly known as **XML Schema** Definition (**XSD**). It is used to describe and validate the structure and the content of **XML** data. **XML schema** defines the elements, attributes and data types. ... It is similar to a database **schema** that describes the data in a database.

Example :

```
<?xml version ="1.0" encoding ="UTF-8"?>
<xs:schemaxmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:elementname="contact">
<xs:complexType>
<xs:sequence>
<xs:elementname="name"type="xs:string"/>
<xs:elementname="company"type="xs:string"/>
<xs:elementname="phone"type="xs:int"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

XML schema and namespace:

When using prefixes in **XML**, a **namespace** for the prefix must be defined. The **namespace** can be defined by an xmlns attribute in the start tag of an element. ... The xmlns attribute in the second <table> element gives the f: prefix a qualified **namespace**.

```
<root>
<h:tablexmlns:h="http://www.w3.org/TR/html4/">
<h:tr>
<h:td>Apples</h:td>
<h:td>Bananas</h:td>
</h:tr>
</h:table>
<f:tablexmlns:f="https://www.w3schools.com/furniture">
<f:name>AfricanCoffeeTable</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
</root>
```

Implementing xml schema types:

The elements contained within a document are processed and the type and instance information from the document is exposed to the consuming software agent. After validation, the information is contained in an XML Document that's called a *post schema-validation Infoset* (usually referred to as just *Infoset*).

Infosets make it possible to reflect over the logical contents of a document, just as in some object-oriented programming languages, revealing the power of XML Schema as a platform-independent type system. To demonstrate, let's start to build some types and see how the (logical) type system works with the (physical) document.

Example:

```
<simpleType name="PostcodeType"
<restriction base="string">
<pattern value="(GIR 0AA)|((([A-Z][0-9][0-9]?)
   ➡        |(([A-Z][A-HJ-Y][0-9][0-9]?)|(([A-Z][0-9][A-Z])|
   ➡        ➡[A-Z][A-HJ-Y][0-9]?[A-Z])))) [0-9][A-Z]{2})"/>
     </restriction>
     </simpleType>
```

The any element:

Inheritance:

The inheritance features in XML Schema allow you to create type hierarchies that capture the data models of the underlying software systems that XML is designed to support.

Syntax:

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

Example:

```
xs:complexType name="MonitorType"

<xs:simpleContent>

<xs:extension base="xs:string">

<xs:attribute name="flatscreen" type="xs:boolean"/>

</xs:extension>
```

</xs:simpleContent>

</xs:complexType>

Substitution group:

A substitution group is a feature of XML schema that allows you to specify elements that can replace another element in documents generated from that schema. The replaceable element is called the head element and must be defined in the schema's global scope. The elements of the substitution group must be of the same type as the head element or a type that is derived from the head element's type.

Syntax:

```
<element name="widget" type="xsd:string" />
<element name="woodWidget" type="xsd:string"
substitutionGroup="widget" />
```

Example

```
<complexType name="widgetType">
<sequence>
<element name="shape" type="xsd:string" />
<element name="color" type="xsd:string" />
</sequence>
</complexType>
<complexType name="woodWidgetType">
<complexContent>
<extension base="widgetType">
<sequence>
<element name="woodType" type="xsd:string" />
</sequence>
</extension>
</complexContent>
```

```
</complexType>
<complexType name="plasticWidgetType">
<complexContent>
<extension base="widgetType">
<sequence>
<element name="moldProcess" type="xsd:string" />
</sequence>
</extension>
</complexContent>
</complexType>
<element name="widget" type="widgetType" />
<element name="woodWidget" type="woodWidgetType"
substitutionGroup="widget" />
<element name="plasticWidget" type="plasticWidgetType"
substitutionGroup="widget" />
<complexType name="partType">
<sequence>
<element ref="widget" />
</sequence>
</complexType>
<element name="part" type="partType" />
```

Global and local type declaration:

As with classes in object-oriented programming, you need to create instances of XML Schema types to do real work such as moving XML-encoded messages between web services. In this section, we'll examine two means of creating instances of types: using global types and declaring local types

Syntax:

<xs:element name="credit-card" type="CardType

minOccurs="0" maxOccurs="unbounded"/>

example:

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

```
<!-- A Global Type -->
<xs:complexType name="CardType">
<xs:sequence>
<xs:element name="card-type">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="Visa"/>
<xs:enumeration value="MasterCard"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
<xs:element name="expiry">
<xs:simpleType>
```

```
<xs:restriction base="xs:string">

<xs:pattern value="[0-9]{2}-[0-9]{2}"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="number">

<xs:simpleType name="CardNumberType">

<xs:restriction base="xs:string">

<xs:pattern

      value="[0-9]{4} [0-9]{4} [0-9]{4} [0-9]{4}"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="holder" type="xs:string"/>

</xs:sequence>

</xs:complexType>
```

```
<!-- A local type -->

<xs:element name="debit-card">

<xs:complexType>

<xs:complexContent>

<xs:extension base="CardType">

<xs:attribute name="issue"

      type="xs:positiveInteger"/>

</xs:extension>

</xs:complexContent>

</xs:complexType>

</xs:element>

<!-- Another local type -->

<xs:element name="wallet">

<xs:complexType>

<xs:sequence>

<xs:element name="credit-card" type="CardType"
```

```
    minOccurs="0" maxOccurs="unbounded"/>

<xs:element name="debit-card" ref="debit-card"

    minOccurs="0" maxOccurs="unbounded"/>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>
```

Managing schema:

A **schema** is a collection of database objects. A schema is owned by a database user and shares the same name as the user. **Schema objects** are logical structures created by users. Some objects, such as tables or indexes, hold data. Other objects, such as views or synonyms, consist of a definition only.

Example

```
CREATE SCHEMA AUTHORIZATION scott

   CREATE TABLE dept (

      deptno NUMBER(3,0) PRIMARY KEY,

      dname VARCHAR2(15),

      loc VARCHAR2(25))

CREATE TABLE emp (
      empno NUMBER(5,0) PRIMARY KEY,

      ename VARCHAR2(15) NOT NULL,

      job VARCHAR2(10),

      mgr NUMBER(5,0),

      hiredate DATE DEFAULT (sysdate),

      sal NUMBER(7,2),

      comm NUMBER(7,2),

      deptno NUMBER(3,0) NOT NULL

      CONSTRAINT dept_fkey REFERENCES dept)

    CREATE VIEW sales_staff AS

      SELECT empno, ename, sal, comm

      FROM emp

      WHERE deptno = 30

      WITH CHECK OPTION CONSTRAINT sales_staff_cnst

      GRANT SELECT ON sales_staff TO human_resources;
```

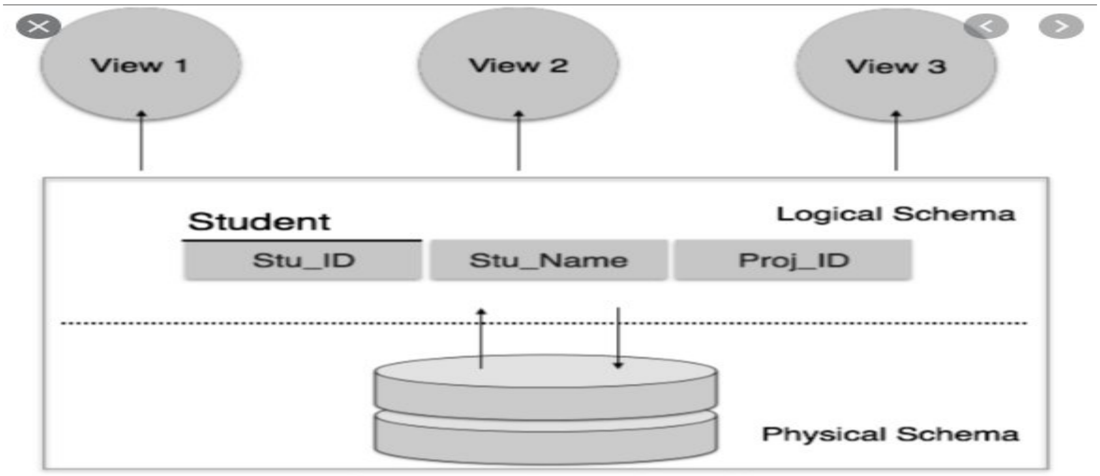
Schemas and instance:

The **Schema and Instance** are the essential terms related to databases. The major difference between **schema and instance** lies within their definition where **Schema** is the formal description of the structure of database whereas **Instance** is the set of information currently stored in a database at a specific time

- The **physical schema** is the lowest level of a schema which describes how the data stored on the disk or the physical storage.
- The **logical schema** is the intermediate level of a schema which describes the structure of the database to the database designers. It also specifies what relationship exists between the data.

- The **external schema** or **subschema** is the highest level of a schema which defines the views for the end users.

An **instance** is the information collected in a database at some specific moment, and it is also known as **state** or **extension**. It is a snapshot where the current state or occurrence of a database is framed at that moment.



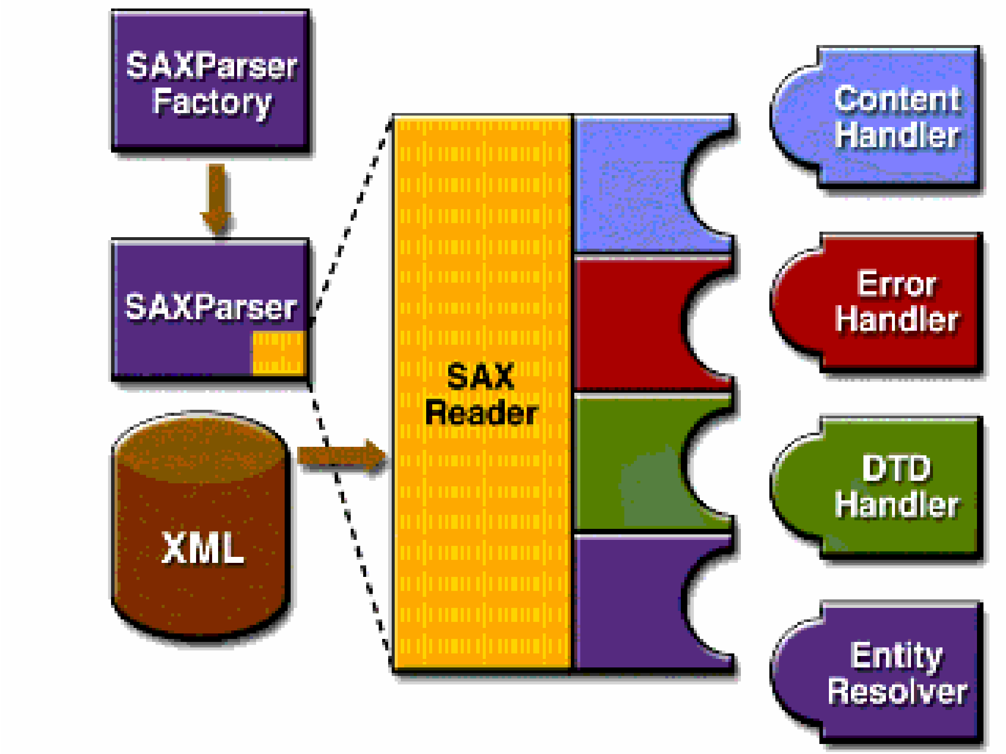
[*Image: Database Schemas*]

Processing XML SAX:

SAX is a streaming interface for **XML**, which means that applications using **SAX** receive event notifications about the **XML** document being **processed** an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element.

- Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document.
- Tokens are processed in the same order that they appear in the document.
- Reports the application program the nature of tokens that the parser has encountered as they occur.
- The application program provides an "event" handler that must be registered with the parser.

- As the tokens are identified, callback methods in the handler are invoked with the relevant information.



Syntax:

```
ublic class SAXLocalNameCount {
   static public void main(String[] args) {
      // ...
   }
}
```

Example:

```
package sax;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

import java.util.*;
import java.io.*;

public class SAXLocalNameCount {
// ...
}
```

SAX:

SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.

SAX is an *event-driven* interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XML parser.

SAX local name:

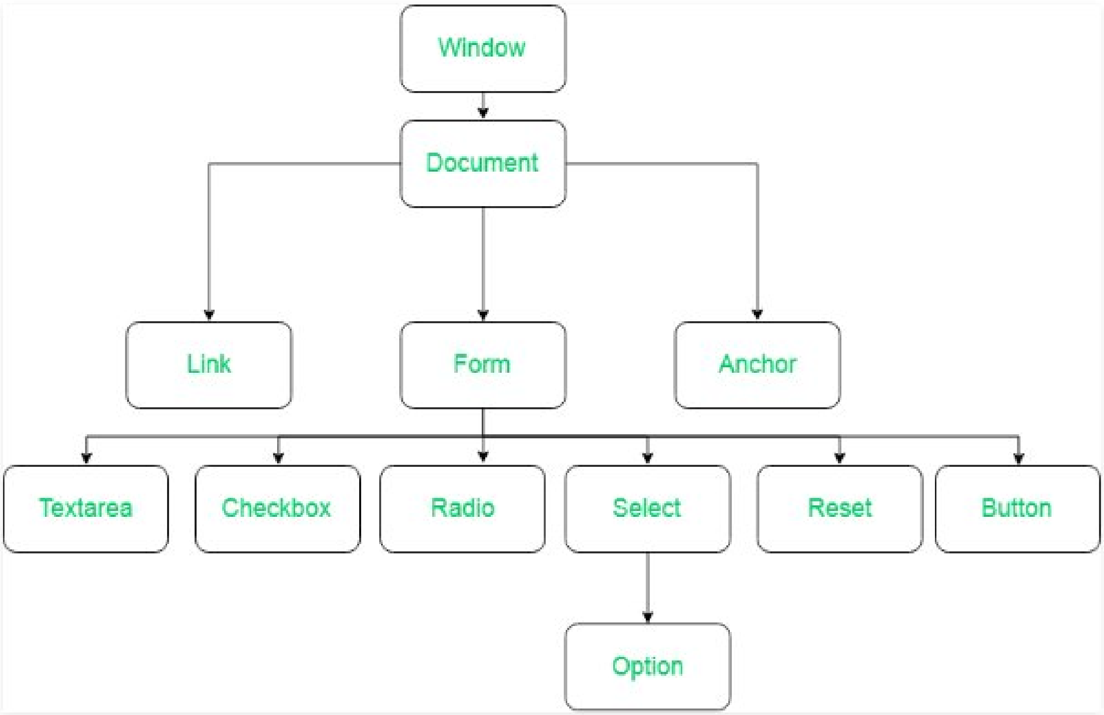
```
static public void main(String[] args) throws Exception
 {
   String filename = null;

   for (int i = 0; i < args.length; i++) {
      filename = args[i];
      if (i != args.length - 1) {
         usage();
      }
   }

   if (filename == null) {
      usage();
   }
}
```

```
public class SAXLocalNameCount {
   private static String convertToFileURL(String filename) {
      String path = new File(filename).getAbsolutePath();
      if (File.separatorChar != '/') {
         path = path.replace(File.separatorChar, '/');
      }

      if (!path.startsWith("/")) {
         path = "/" + path;
      }
      return "file:" + path;
   }

   // ...
}
private static void usage() {
   System.err.println("Usage: SAXLocalNameCount <file.xml>");
   System.err.println("       -usage or -help = this message");
   System.exit(1);
}
```

Document object model:

The **Document Object Model** (**DOM**) is a cross-platformand language-independent interface that treats an XML or HTML document as a tree structure wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. Nodes can have event handlers attached to them. Once an event is triggered, the event handlers get executed.

To render a document such as a HTML page, most web browsers use an internal model similar to the **DOM**. The nodes of every document are organized in a tree structure, called the *DOM tree*, with the topmost node named as "Document object". When an HTML page is rendered in browsers, the browser downloads the HTML into local memory and automatically parses it to display the page on screen



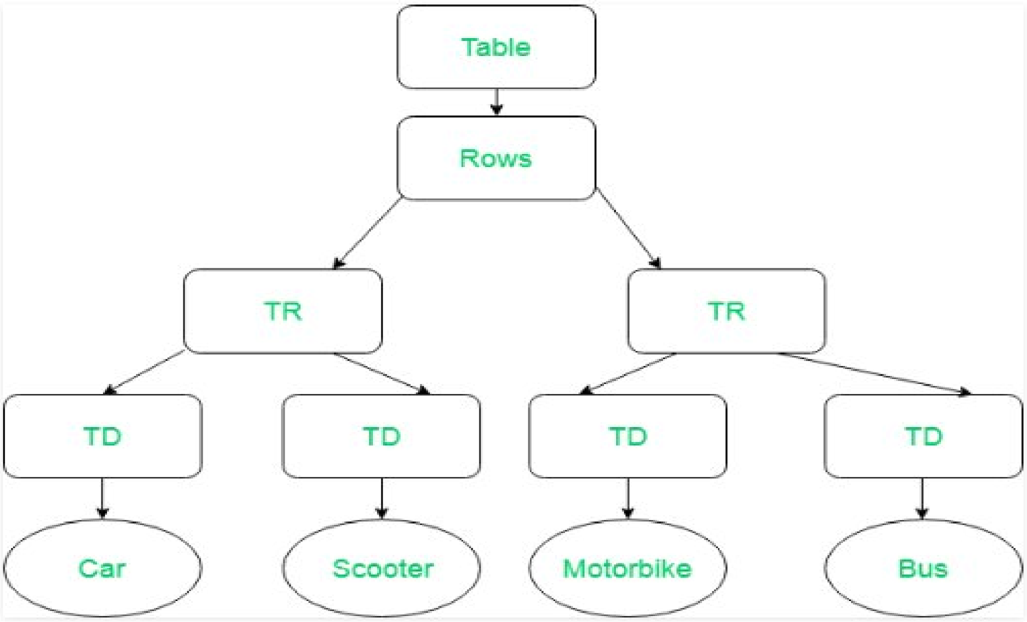
1. **Window Object:** Window Object is at always at top of hierarchy.
2. **Document object:** When HTML document is loaded into a window, it becomes a document object.
3. **Form Object:** It is represented by ***form*** tags.
4. **Link Objects:** It is represented by ***link*** tags.
5. **Anchor Objects:** It is represented by ***a href*** tags.
6. **Form Control Elements:**: Form can have many control elements such as text fields, buttons, radio buttons, and checkboxes, etc.

Methods of document object:

1. **write(“string”):** writes the given string on the document.
2. **getElementById():** returns the element having the given id value.
3. **getElementsByName():** returns all the elements having the given name value.
4. **getElementsByTagName():** returns all the elements having the given tag name.
5. **getElementsByClassName():** returns all the elements having the given class name.

Example:

```
<Table>

  <ROWS>
    <TR>
      <TD>Car</TD>
      <TD>Scooter</TD>
    </TR>
    <TR>
      <TD>MotorBike</TD>
      <TD>Bus</TD>
    </TR>
  </ROWS>
</Table>
```



Example:

```
<html>
<head>
<title>DOM Tests</title>
<script>
functionsetBodyAttr(attr, value){
if(document.body) document.body[attr]= value;
elsethrownewError("no support");
}
</script>
</head>
<body>
<div style="margin:.5in;height:400px;">
<p><b><tt>text</tt></b></p>
<form>
<select onChange="setBodyAttr('text',
        this.options[this.selectedIndex].value);">
<option value="black">black</option>
<option value="red">red</option>
```

```
</select>
<p><b><tt>bgColor</tt></b></p>
```

```
<select onChange="setBodyAttr('bgColor',
        this.options[this.selectedIndex].value);">
<option value="white">white</option>
<option value="lightgrey">gray</option>
</select>
<p><b><tt>link</tt></b></p>
<select onChange="setBodyAttr('link',
        this.options[this.selectedIndex].value);">
<option value="blue">blue</option>
<option value="green">green</option>
</select>
<small>
<a href="http://some.website.tld/page.html"id="sample">
          (sample link)
</a>
</small><br />
<input type="button"value="version"onclick="ver()"/>
</form>
</div>
</body>
</html>
```

XSLT:

XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents, or other formats such as HTML for web pages, plain text or XSL Formatting Objects,

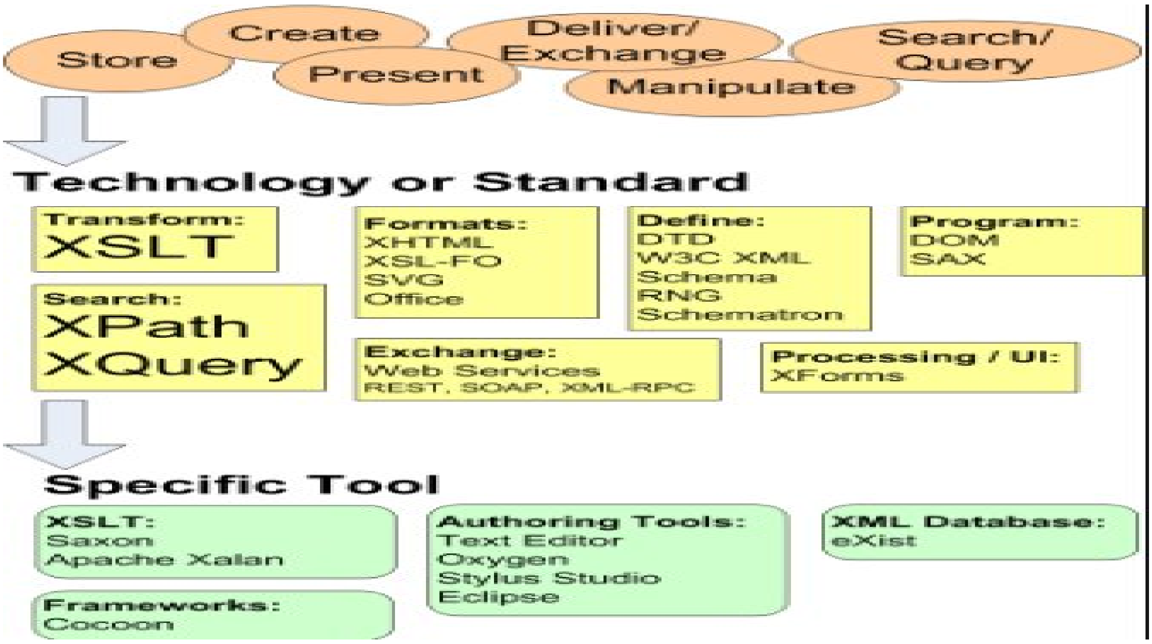
Example:

```
<?xml version="1.0"?>

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

<xsl:template match="/">
  <html>
  <body>
    <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
        <tr>
          <td><xsl:value-of select="title"/></td>
          <td><xsl:value-of select="artist"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

XPATH:

XPath is a W3C Recommendation that is used for identifying elements, attributes, text and other nodes within an XML document. ... In **XSLT**, **XPath** is used to match nodes from the source document for output templates via the match attribute of the xsl:template tag

XPath expressions can also be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.



Syntax:

<xsl:template match="SOME_XPATH">

XPath is also used in the xsl:value-of tag to specify elements, attributes and text to decide what to output.

<xsl:value-of select="SOME_XPATH"/>

Example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="XPaths.xsl"?>
<BusinessLetter>
    <Head>
        <SendDate>November 29, 2011</SendDate>
        <Recipient>
            <Name Title="Mr.">
                <FirstName>Joshua</FirstName>
                <LastName>Lockwood</LastName>
            </Name>
            <Company>Lockwood &amp; Lockwood</Company>
            <Address>
                <Street>291 Broadway Ave.</Street>
                <City>New York</City>
                <State>NY</State>
                <Zip>10007</Zip>
                <Country>United States</Country>
            </Address>
        </Recipient>
    </Head>
    <Body>
        <List>
            <Heading>
```

```
                                        Along with this letter, I have enclosed the
following items:
                              </Heading>
                              <ListItem>
                                        two original, execution copies of the Webucator
                                        Master Services Agreement
                              </ListItem>
                              <ListItem>
                                        two original, execution copies of the Webucator
Premier Support for
                                        Developers Services Description between
                                        Lockwood &amp; Lockwood and Webucator, Inc.
                              </ListItem>
                    </List>
                    <Para>Please sign and return all four original, execution copies to
me at
                    your earliest convenience.  Upon receipt of the executed copies,
we will
                    immediately return a fully executed, original copy of both
agreements
                    to you.</Para>
                    <Para>
          Please send all four original execution copies to my attention as follows:

          <Person>
                                        <Name>
                                                  <FirstName>Bill</FirstName>
                                                  <LastName>Smith</LastName>
```

```
                </Name>
                <Address>
                    <Company>Webucator, Inc.</Company>
                    <Street>4933 Jamesville Rd.</Street>
                    <City>Jamesville</City>
                    <State>NY</State>
                    <Zip>13078</Zip>
                    <Country>USA</Country>
                </Address>
            </Person>
        </Para>
        <Para>If you have any questions, feel free to call me at
        <Phone>800-555-1000 x123</Phone> or e-mail me at
        <Email>bsmith@webucator.com</Email>.</Para>
    </Body>
    <Foot>
        <Closing>
            <Name>
                <FirstName>Bill</FirstName>
                <LastName>Smith</LastName>
            </Name>
            <JobTitle>VP of Operations</JobTitle>
        </Closing>
    </Foot>
</BusinessLetter>
```

UNIT-IV

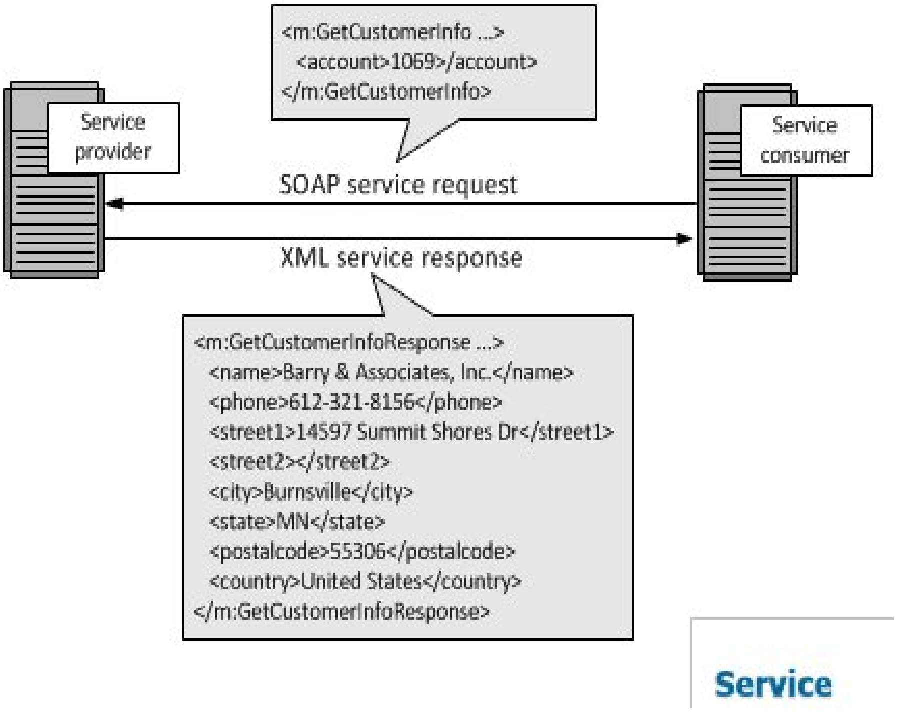
SOAP:

SOAP (Simple **Object Access Protocol**) is a message protocol that allows distributed elements of an application to communicate.SOAP can be carried over a variety of lower-level protocols, including the web-related Hypertext Transfer Protocol (HTTP).

WSDL:

WSDL stands for Web Services Description Language. It is the standard format for describing a web service. WSDL is a language for describing how to interface with XML-based services the language that UDDI uses.

The SOAP Model:



```
<m:GetCustomerInfo ...>
  <account>1069>/account>
</m:GetCustomerInfo>
```

```
<m:GetCustomerInfoResponse ...>
  <name>Barry & Associates, Inc.</name>
  <phone>612-321-8156</phone>
  <street1>14597 Summit Shores Dr</street1>
  <street2></street2>
  <city>Burnsville</city>
  <state>MN</state>
  <postalcode>55306</postalcode>
  <country>United States</country>
</m:GetCustomerInfoResponse>
```

SOAP Message:

SOAP provides the Messaging Protocol layer of a **web** services protocol stack for **web** services. It is an XML-based protocol consisting of three parts: an **envelope**, which defines the **message** structure and how to process it. a set of encoding rules for expressing instances of application-defined datatypes.

Example:

```
<?xml version ="1.0"?>
<SOAP-ENV:Envelopexmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-
envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<SOAP-ENV:Header>
    ...
    ...
</SOAP-ENV:Header>
<SOAP-ENV:Body>
```

```
      ...
      ...
<SOAP-ENV:Fault>
        ...
        ...
</SOAP-ENV:Fault>
      ...
</SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

SOAP Envelope:

The **SOAP envelope** contains two parts: An optional header providing information on authentication, encoding of data, or how a recipient of a **SOAP message** should process the **message**. The body that contains the **message**.

Example

```
<?xmlversion="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
...
Message information goes here
...
</soap:Envelope>
```

SOAP Header:

The **SOAP** <**Header**> is an optional element in a **SOAP** message. It is used to pass application-related information that is to be processed by **SOAP** nodes along the message path. The immediate child elements of the <**Header**> element are called **header** blocks.

Example:

```
<?xml version ="1.0"?>
```

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=" http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">

<SOAP-ENV:Header>
<t:Transaction
xmlns:t="http://www.tutorialspoint.com/transaction/"
SOAP-ENV:mustUnderstand="true">5
</t:Transaction>
</SOAP-ENV:Header>
   ...
   ...
</SOAP-ENV:Envelope>
```

SOAP Body:

The **SOAP body** is a mandatory element that contains the application-defined XML data being exchanged in the **SOAP message**. ... The **body** is defined as a child element of the envelope, and the semantics for the **body** are defined in the associated **SOAP** schema.

Example

```
?xml version ="1.0"?>
<SOAP-ENV:Envelope>
   ........
<SOAP-ENV:Body>
<m:GetQuotationxmlns:m="http://www.tp.com/Quotation">
<m:Item>Computers</m:Item>
</m:GetQuotation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Faults:

The optional **SOAP Fault** element is used to indicate error messages. The **SOAP Fault** element holds errors and status information for a **SOAP** message. If a **Fault** element is present, it must appear as a child element of the Body element. A **Fault** element can only appear once in a **SOAP** message.

Example:

```
<?xmlversion="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
                                                                                                                        ...
</soap:Fault>
</soap:Body>

</soap:Envelope>
```

SOAP ENCODNG:

SOAP Encoding is an extension of the **SOAP** framework specification that defines how a data value should be **encoded** in an XML format. **SOAP** Data Model is defined as an adjunct in **SOAP** 1.2 specification. **SOAP encoding** offers the following rules to convert any data value defined in **SOAP** data model into XML format.

Example:

```
<?xml version ='1.0' encoding ='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>
<ns1:getPriceResponse
xmlns:ns1="urn:examples:priceservice"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<returnxsi:type="xsd:double">54.99</return>
</ns1:getPriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP RPC :

RPC stands for Remote Procedure Call. XML-RPC is among the simplest and most foolproof web service approaches that makes it easy for computers to call procedures on other computers.

XML-RPC consists of three relatively small parts:

- **XML-RPC data model** : A set of types for use in passing parameters, return values, and faults (error messages).
- **XML-RPC request structures** : An HTTP POST request containing method and parameter information.
- **XML-RPC response structures** : An HTTP response that contains return values or fault information.

```
POST /sayHello HTTP/1.1
POST: api.example.com
Content-Type: application/json
{"name": "Racey McRacerson"}
```

Using alternative SOAP encoding :

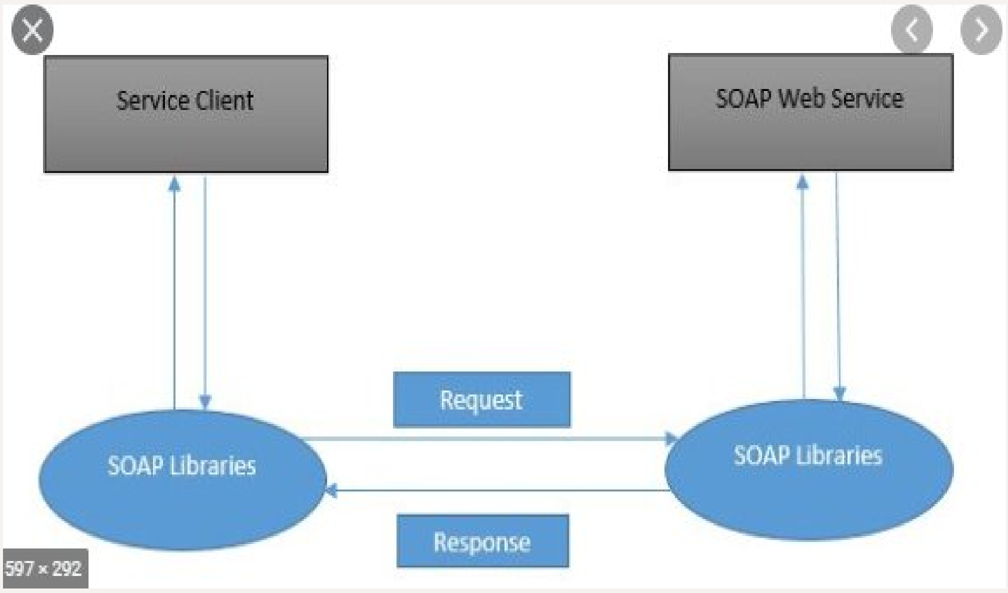
The way in which alternative SOAP encodings are handled is straightforward. Instead of encoding header or body content according to the SOAP Data Model, we simply encode according to the rules and constraints of our data model and schema. In essence, we can just slide our own XML documents into a SOAP message, providing we remember to specify the encodingStyle attribute (and of course ensuring that the intended recipients of the message can understand it). This style of SOAP encoding is known as literal style and naturally suits the interchange of business-level documents based on their existing schemas.

Example

```
xml version ='1.0' encoding ='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>
<ns1:getPriceResponse
xmlns:ns1="urn:examples:priceservice"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<returnxsi:type="xsd:double">54.99</return>
</ns1:getPriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Web service:

SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.SOAP is a W3C recommendation for communication between two applications.SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

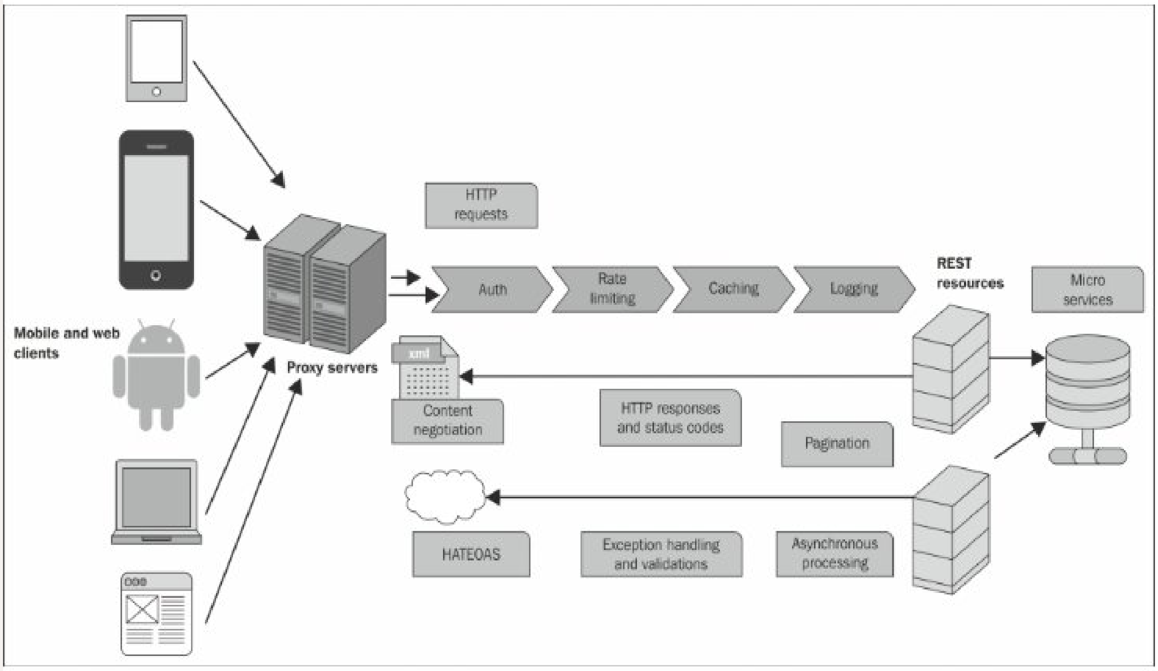


REST Architecture:

Representational state transfer (**REST**) is a software **architectural** style that defines a set of constraints to be used for creating **Web** services. **Web** services that conform to the **REST architectural** style, called **RESTful Web** services, provide interoperability between computer systems on the **Internet**.

A Restful system consists of a:

- client who requests for the resources.
- server who has the resources.



Difference between soap 1.1 and soap 1.2:

The SOAP 1.2 specification introduces several changes to SOAP 1.1. This information is not intended to be an in-depth description of all the new or changed features for SOAP 1.1 and SOAP 1.2. Instead, this information highlights some of the more important differences between the current versions of SOAP.

SOAP 1.1 request:

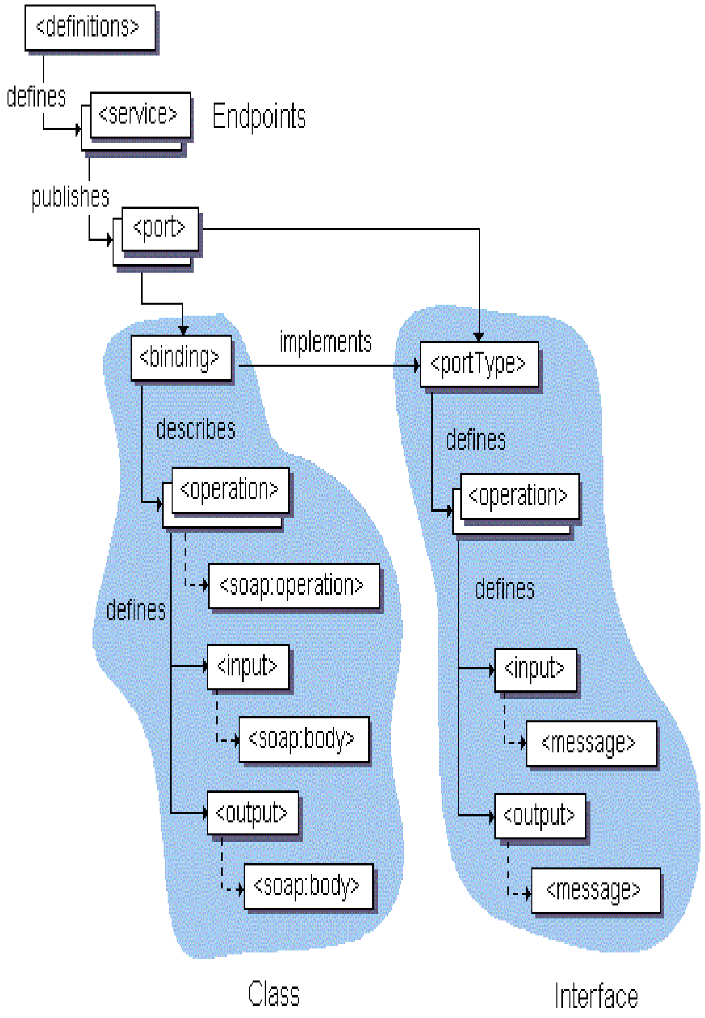
```
POST /WSShakespeare.asmx HTTP/1.1
Host: www.xmlme.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://xmlme.com/WebServices/GetSpeech"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<GetSpeech xmlns="http://xmlme.com/WebServices">
<Request>string</Request>
</GetSpeech>
</soap:Body>
</soap:Envelope>

SOAP 1.2 request:

POST /WSShakespeare.asmx HTTP/1.1
Host: www.xmlme.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
<soap12:Body>
<GetSpeech xmlns="http://xmlme.com/WebServices">
<Request>string</Request>
</GetSpeech>
</soap12:Body>
</soap12:Envelope>
```

WSDL Structure:

A **WSDL** document defines services as collections of network endpoints, or ports . In **WSDL**, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. ... Binding – a concrete protocol and data format specification for a particular port type.

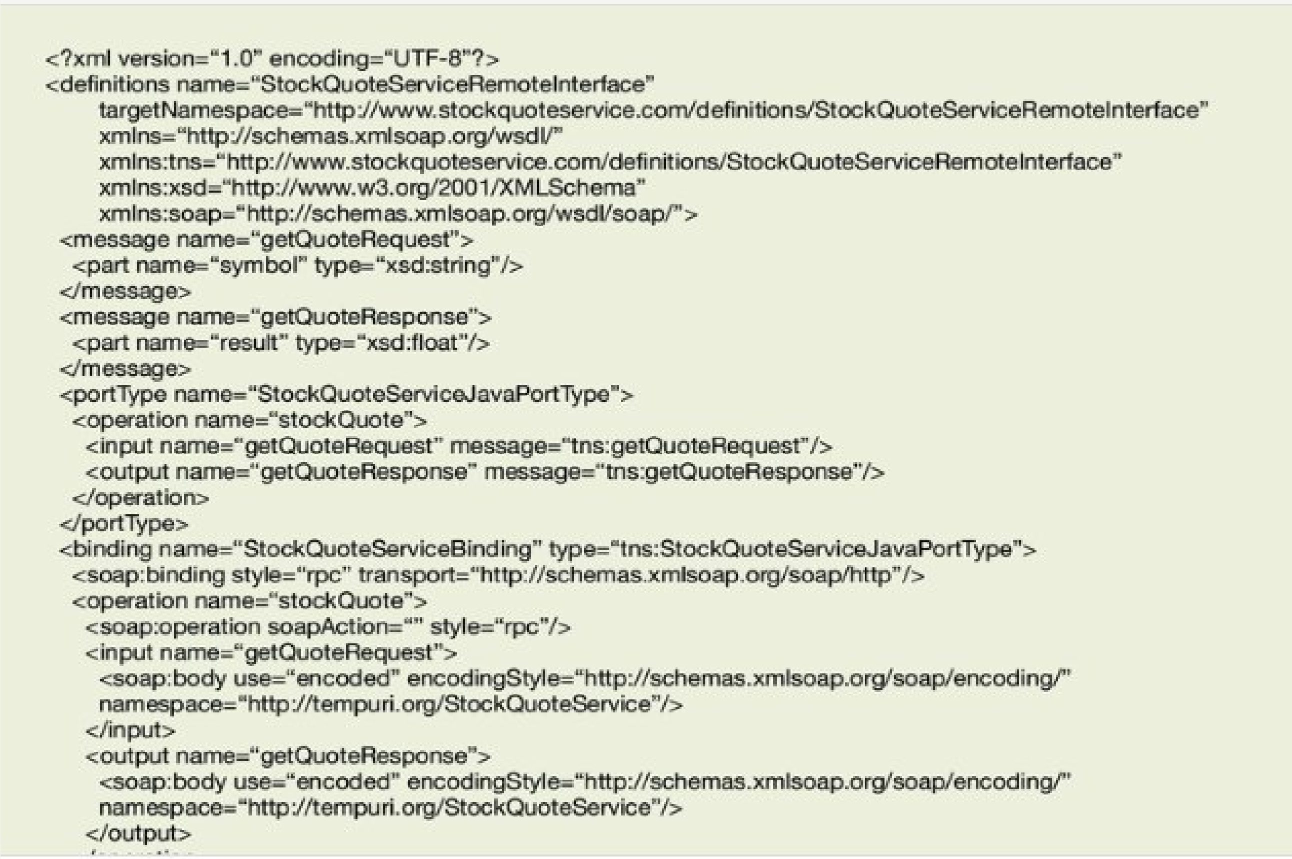
Example:



Example:

```
<!-- WSDL definition structure -->
<definitions
                    name="Guru99Service"
      targetNamespace=http://example.org/math/
      xmlns=http://schemas.xmlsoap.org/wsdl/>
          <!-- abstract definitions -->
                    <types> ...
                              <message> ...
                              <portType> ...

<!-- concrete definitions -->
                    <binding> ...
                    <service> ...
</definition>
```

The stock quote WSDL interface:



```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="StockQuoteServiceRemoteInterface"
        targetNamespace="http://www.stockquoteservice.com/definitions/StockQuoteServiceRemoteInterface"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:tns="http://www.stockquoteservice.com/definitions/StockQuoteServiceRemoteInterface"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <message name="getQuoteRequest">
    <part name="symbol" type="xsd:string"/>
  </message>
  <message name="getQuoteResponse">
    <part name="result" type="xsd:float"/>
  </message>
  <portType name="StockQuoteServiceJavaPortType">
    <operation name="stockQuote">
      <input name="getQuoteRequest" message="tns:getQuoteRequest"/>
      <output name="getQuoteResponse" message="tns:getQuoteResponse"/>
    </operation>
  </portType>
  <binding name="StockQuoteServiceBinding" type="tns:StockQuoteServiceJavaPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="stockQuote">
      <soap:operation soapAction="" style="rpc"/>
      <input name="getQuoteRequest">
        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://tempuri.org/StockQuoteService"/>
      </input>
      <output name="getQuoteResponse">
        <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://tempuri.org/StockQuoteService"/>
      </output>
```

The types:

Element

- Binding
- Service
- Managing WSDL description
- Extending WSDL

ELEMENT:

The HTML <content> **element**—an obsolete part of the **Web** Components suite of **technologies**—was used inside of Shadow DOM as an insertion point, and wasn't meant to be used in ordinary HTML. ... The HTML <frameset> **element** is used to contain <frame> **elements**.

Example:

```
<!DOCTYPE html>
<html>

<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

BINDING:

Binding is a process that allows an Internet user to manipulate Web page elements using a Web browser. It employs dynamic HTML (hypertext markup language) and does not require complex scripting or programming.

Syntax:

```
<ul data-bind="foreach: photos">

<li><img data-bind="attr: {src: $data.get('url')}"/></li>
</ul>
```

Example:

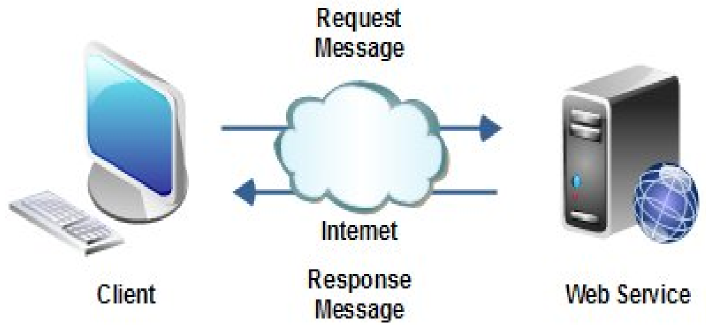
```
module.exports = classEditCompanyViewextendsBackbone.View
```

```
template: require "./templates/companies/edit"

  initialize: () ->
     @view = new EditCompanyViewModel()
     @render()

  render: () ->
     $(@el).html(@template())
     ko.applyBindings(@view, @el)

  setData: (company) =>
     @view.company(company)

classEditCompanyViewModel
constructor: ->
# let’s instantiate it with default company for bindings to work
     @company = ko.observable(new Company())
     @companyName = ko.computed
        read: => @company.get('name')
        write: (value) => @company.set('name': value)
     @companyState = ko.computed
        read: => @company.get('state')
        write: (value) => @company.set('state': value)
     @companyWebsite = ko.computed
        read: => @company.get('url')
        write: (value) => @company.set('url': value)
        @saveCompany = () =>
            @company.save()
```

SERVICE:

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction.

A **web service** is a collection of open protocols and standards used for exchanging data between applications or systems.



A clients sends a request message to a web service, and receives a response message.

Managing WSDL description:

WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service **can** read the **WSDL** to determine what functions are available on the server. Any special datatypes used are embedded in the **WSDL file** in the form of XML Schema.

Example:

```
<!-- WSDL definition structure -->
<definitions
                 name="Guru99Service"
     targetNamespace=http://example.org/math/
     xmlns=http://schemas.xmlsoap.org/wsdl/>
        <!-- abstract definitions -->
                 <types> ...
                          <message> ...
                          <portType> ...

<!-- concrete definitions -->
                 <binding> ...
                 <service> ...
</definition>
```

Extending WSDL:

Web services have received significant attention recently as many companies including Microsoft are pushing this technology. Web services have many attractive features such as using standard protocols including SOAP,WSDL and UDDI.

Example:

```
        class formatDate extends Date {


 getFormattedDate() {
   const months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
             'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
   return                    `${this.getDate()}-${months[this.getMonth()]}-
${this.getFullYear()}`;
 }


}


console.log(new        formatDate('August        19,        1975
23:15:30').getFormattedDate());
// expected output: "19-Aug-1975"
```

Using SOAP and WSDL:

It is used to access Web pages or to surf the Net. HTTP ensures that you do not have to worry about what kind of Web server -- whether Apache or IIS or any other -- serves you the pages you are viewing or whether the pages you view were created in ASP.NET or HTML.

SOAP Request:

```
Host: www.bookshop.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
```

```
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.bookshop.org/prices">
  <m:GetBookPrice>
    <m:BookName>The Fleamarket</m:BookName>
  </m:GetBookPrice>
</soap:Body>
</soap:Envelope>
```

SOAP response:

```
Host: www.bookshop.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.bookshop.org/prices">
  <m:GetBookPriceResponse>
    <m: Price>10.95</m: Price>
  </m:GetBookPriceResponse>
</soap:Body>
</soap:Envelope>
```

WSDL:

WSDL is a document that describes a Web service and also tells you how to access and use its method

Example

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

```
<definitions  name ="DayOfWeek"
```

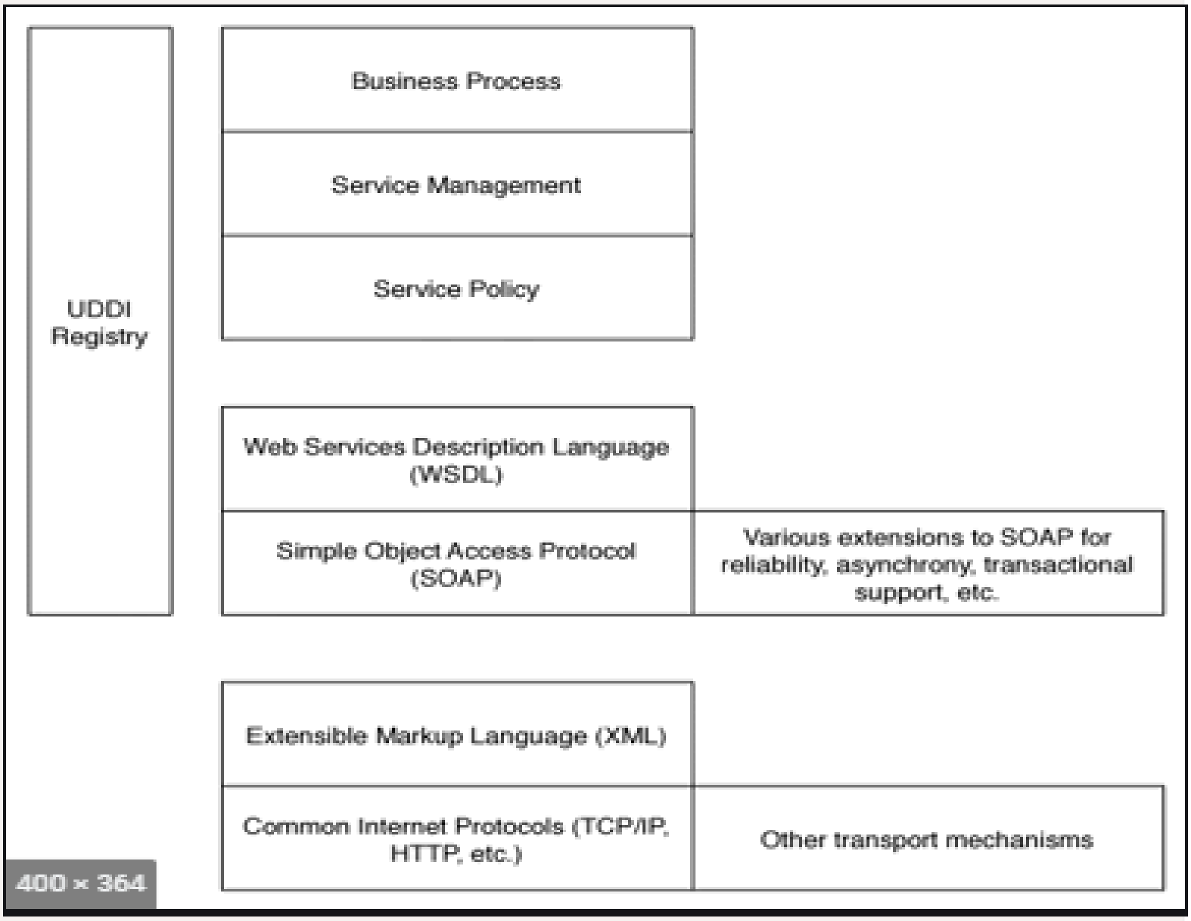
```
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
```

```
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
  </service>
</definitions>
```

UNIT-V

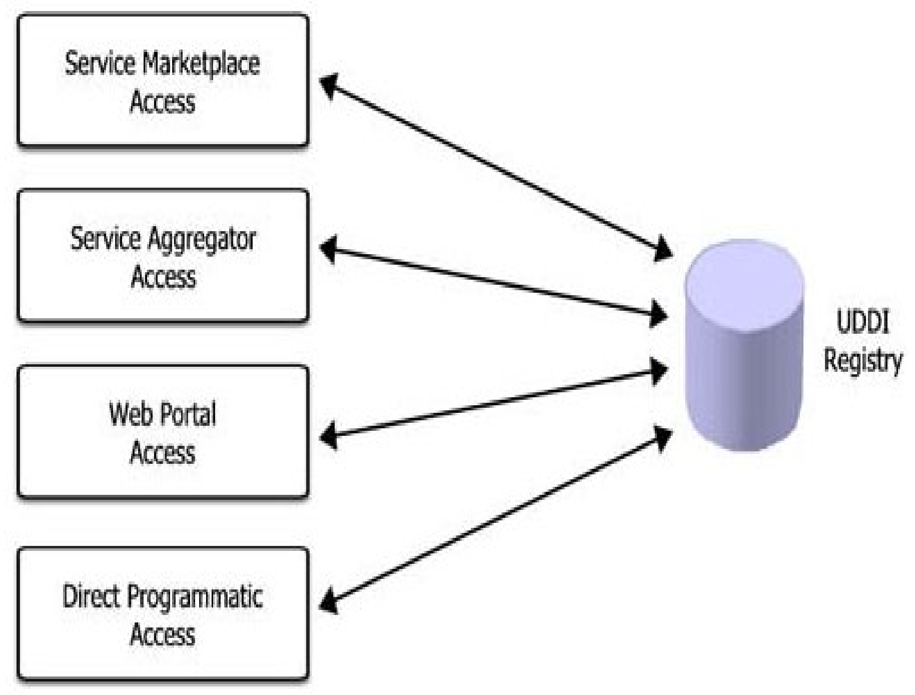
UDDI:

UDDI stands for Universal Description, Discovery, and Integration. Is an XML-based standard for describing, publishing, and finding **web** services. **UDDI** is an open industry initiative, enabling businesses to discover each other and define how they interact over the **Internet**.


UDDI Registry
Business Process
Service Management
Service Policy
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)
Various extensions to SOAP for reliability, asynchrony, transactional support, etc.
Extensible Markup Language (XML)
Common Internet Protocols (TCP/IP, HTTP, etc.)
Other transport mechanisms
400 × 364

The UDDI Business Registry:

The **UDDI Business Registry** (UBR) is a global implementation of the **UDDI** specification. The UBR is a single **registry** for Web services. A group of companies operate and host UBR nodes, each of which is an identical copy of all other nodes.

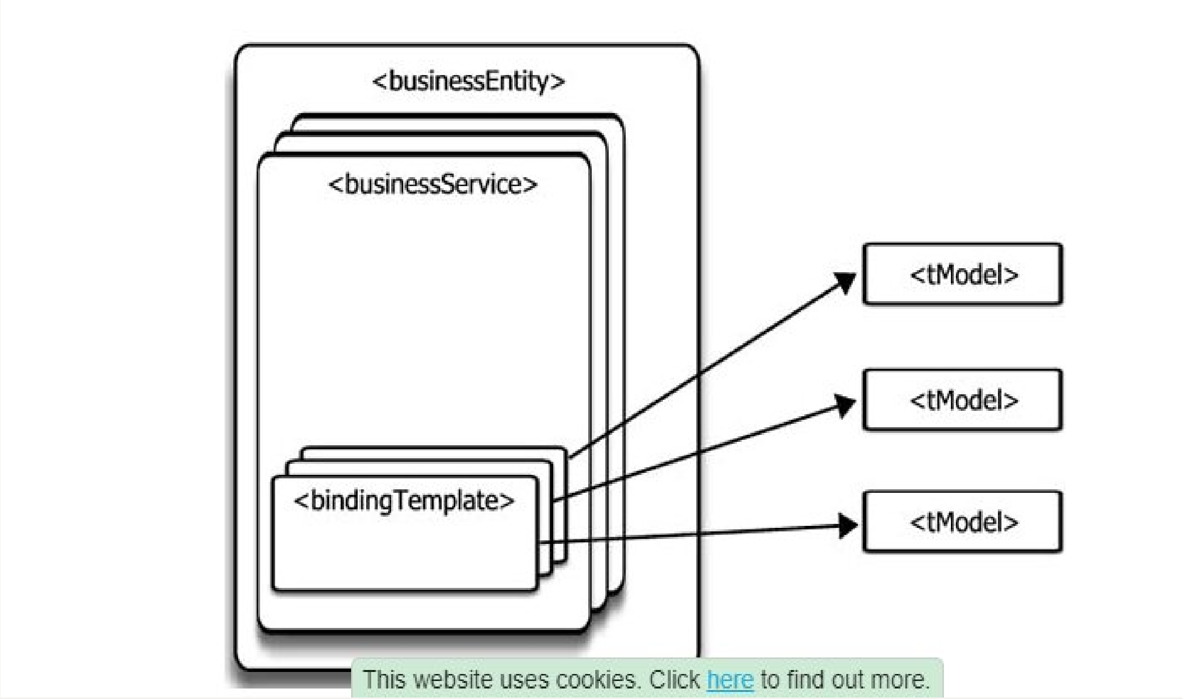

Service Marketplace Access
Service Aggregator Access
Web Portal Access
Direct Programmatic Access
UDDI Registry

The UDDI under covers:

As such the data structures used by UDDI provide not only technical interface information about a service itself, but also information necessary to classify, manage, and locate services. depicts the interrelationships between the core UDDI data structures.

- **Business Identification** - Multiple names and descriptions of the business, comprehensive contact information, and standard business identifiers such as a tax identifier.
- **Categories** - Standard categorization information (for example a D-U-N-S business category number).
- **Service Description** - Multiple names and descriptions of a service. As a container for service information, companies can advertise numerous services, while clearly displaying the ownership of services. The **`bindingTemplate`** information describes how to access the service.
- **Standards Compliance** - In some cases it is important to specify compliance with standards. These standards might display detailed technical requirements on how to use the service.

- **Custom Categories -** It is possible to publish proprietary specifications (tModels) that identify or categorize businesses or services.



This website uses cookies. Click here to find out more.

Example:

```
<bindingTemplateserviceKey="uuid:D6F1B765-BDB3-4837-828D-
8284301E5A2A"
bindingKey="uuid:C0E6D5A8-C446-4f01-99DA-70E212685A40">
<description>Hello World SOAP Binding</description>
<accessPointURLType="http">http://localhost:8080</accessPoint>

<tModelInstanceDetails>
<tModelInstanceInfotModelKey="uuid:EB1B645F-CF2F-491f-811A-
4868705F5904">
<instanceDetails>
<overviewDoc>
<description>
                references the description of the WSDL service definition
</description>
```

```
<overviewURL>
                http://localhost/helloworld.wsdl
</overviewURL>
</overviewDoc>
</instanceDetails>
</tModelInstanceInfo>
</tModelInstanceDetails>
</bindingTemplate>
```

Accessing UDDI:

UDDI is itself a Web service and as such, applications can communicate with an UDDI registry by sending and receiving XML messages. This makes the access both language and platform independent.

Although it's possible, it is unlikely that programmers will deal with the low-level details of sending and receiving XML messages.

UDDI Overview:

UDDI is an XML-based standard for describing, publishing, and finding **web** services. **UDDI** stands for Universal **Description**, Discovery, and Integration. **UDDI** is a specification for a distributed registry of **web** services. ... **UDDI** uses **Web** Service Definition Language(**WSDL**) to describe interfaces to **web** services.

- UDDI white pages: basic information such as a company name, address, and phone numbers, as well as other standard business identifiers like Dun & Bradstreet and tax numbers.
- UDDI yellow pages: detailed business data, organized by relevant business classifications. The UDDI version of the yellow pages classifies businesses according to the newer NAICS (North American Industry Classification System) codes, as opposed to the SIC (Standard Industrial Classification) codes.

- UDDI green pages: information about a company's key business processes, such as operating platform, supported programs, purchasing methods, shipping and billing requirements, and other higher-level business protocols.

  Example:

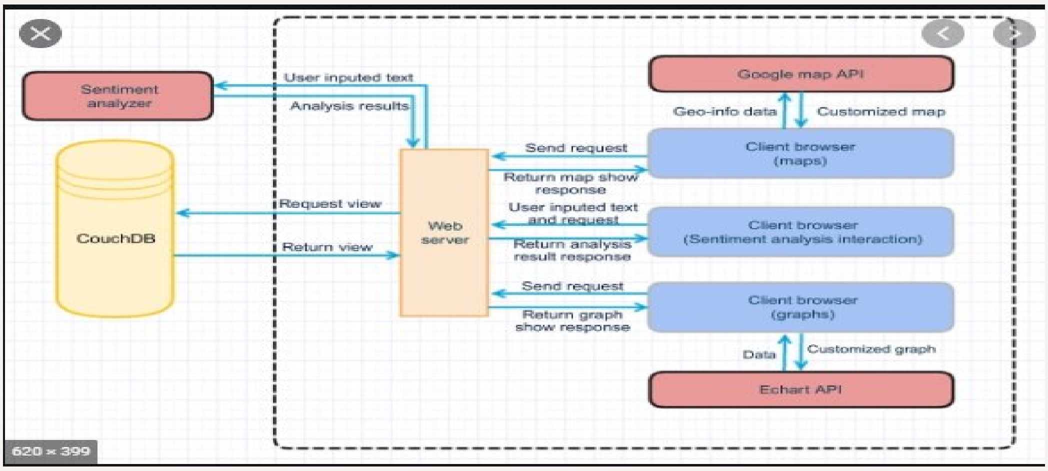
```
POST /save_business HTTP/1.1
Host: www.XYZ.com
Content-Type: text/xml; charset ="utf-8"
Content-Length: nnnn
SOAPAction:"save_business"

<?xml version ="1.0" encoding ="UTF-8"?>
<Envelope xmlns ="http://schemas/xmlsoap.org/soap/envelope/">
<Body>
<save_business generic="2.0" xmlns ="urn:uddi-org:api_v2">
<businessKey ="">
</businessKey>

<name>
        XYZ,PvtLtd.
</name>

<description>
Companyis involved in giving Stat-of-the-art....
</description>

<identifierBag>...</identifierBag>
...
</save_business>
</Body>
</Envelope>
```

Web services:

A **Web service** is a software **service** used to communicate between two devices on a network. More specifically, a **Web service** is a software application with a standardized way of providing interoperability between disparate applications. It does so over HTTP using **technologies** such as XML, SOAP, WSDL, and UDDI.

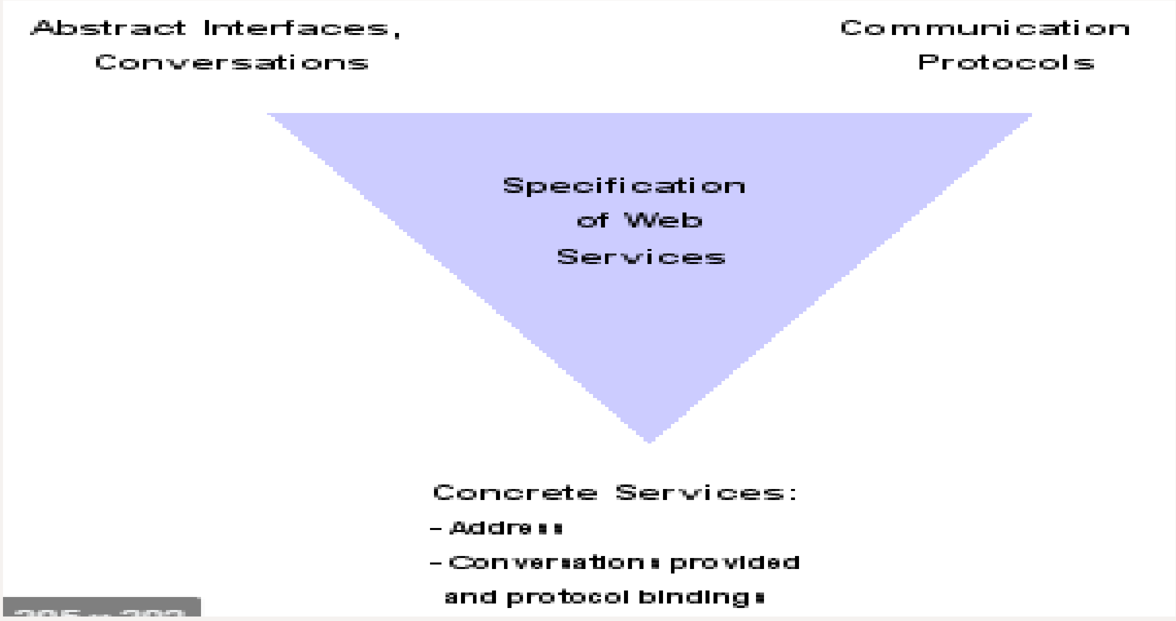
Components of Web Services:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)



<u>Web services conversation language:</u>

Web Services Conversation Language (WSCL) **Web Services Conversation Language** (WSCL) allows the business level **conversations** or public processes supported by a **Web service** to be defined. WSCL specifies the XML documents being exchanged, and the allowed sequencing of these document exchanges



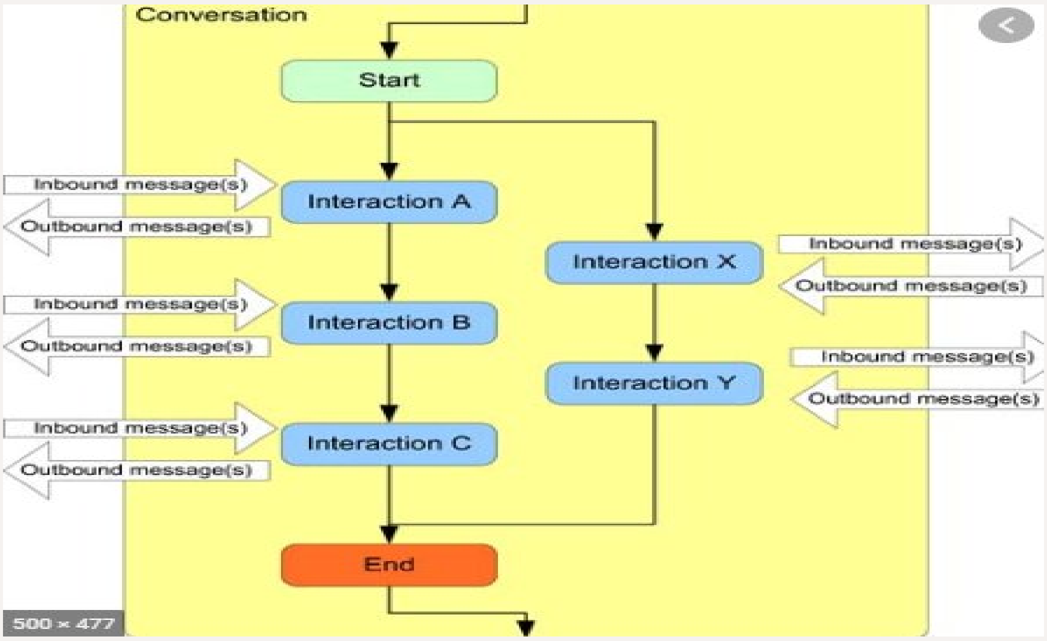
.

WSCL Interface components:

Web Services Conversation Language (WSCL) allows the business level **conversations** or public processes supported by a **Web service** to be defined. WSCL specifies the XML documents being exchanged, and the allowed sequencing of these document exchanges.

Example:

```
<?xml version="1.0" encoding="UTF-8"?><xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" xmlns:m="http://money.example.org"
targetNamespace="http://money.example.org"><xs:complexType
name="money"><xs:sequence><xs:element name="currency"
type="xs:string"/><xs:element name="value"
type="xs:decimal"/></xs:sequence></xs:complexType><xs:element
name="money" type="m:money"/></xs:schema
```



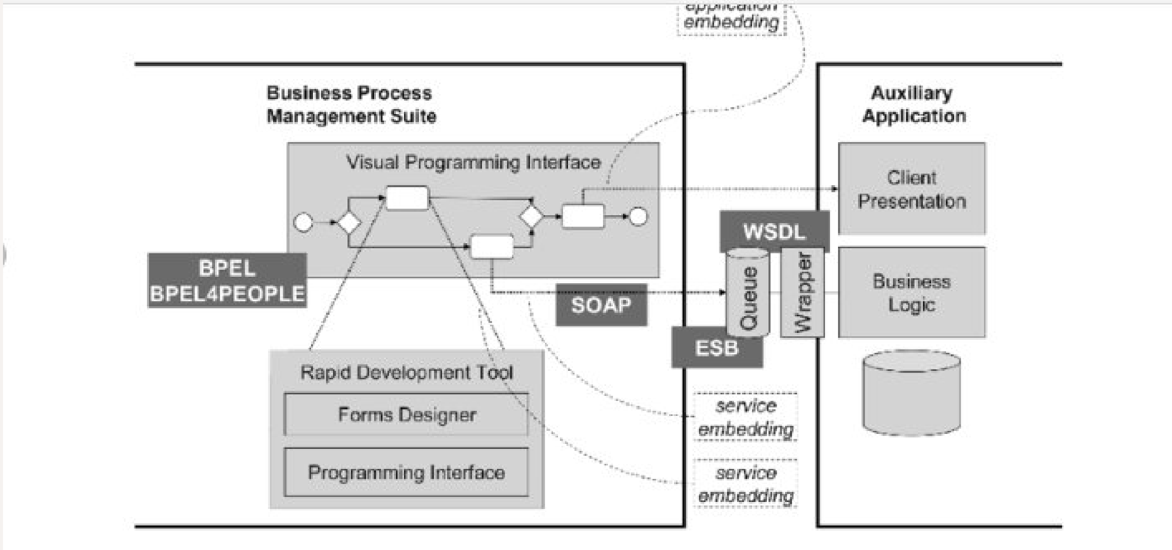
Relationship between WSDL and WSCL:

WSCL is a capable, if minimal interface description language. It is based on the notion of being able to represent a conversation as a state machine, which is then itself represented in an XML form.

The protocols used to enable the sending and receipt of messages.Services: The concrete service implementation that provides a network accessible address for a particular protocol at which the given message set is understood—i.e., the location of an instance of the Web service.

Business process management:

Business Process Management (BPM) is a new IT framework for visualizing, improving, and executing new or existing **processes**. Fujitsu has introduced **Web**-service integration **technology** for this new framework. This paper introduces **Web**-service integration **technology**.



Typical business process management systems based on web-service technology.

Workflow:

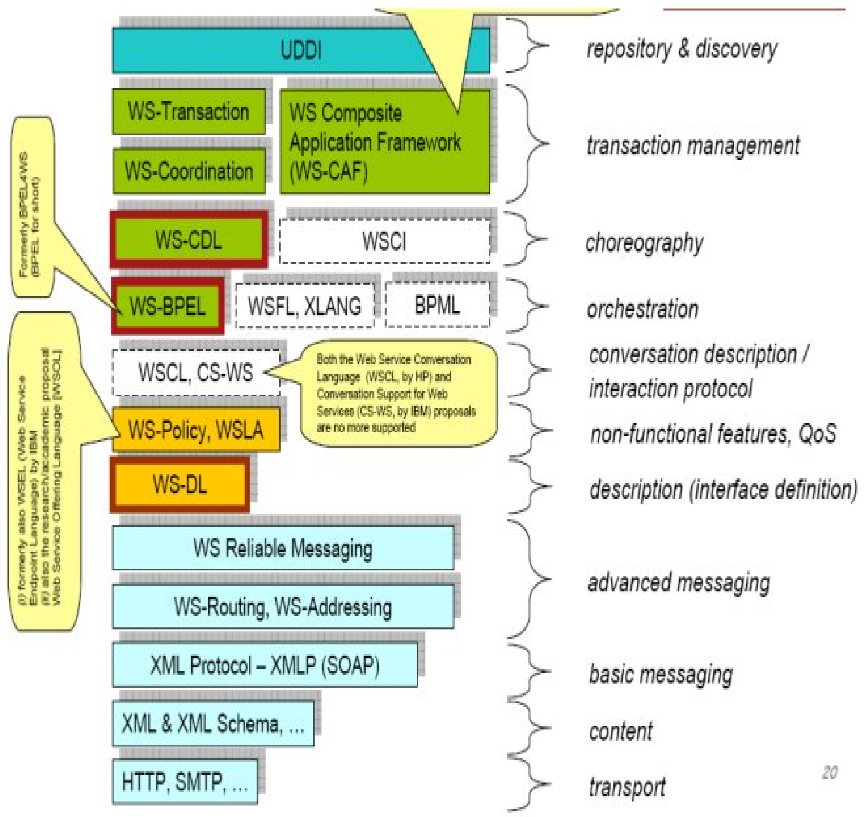
Workflows can also have more complex dependencies; for example if a document is to be translated into several languages, a translation manager could select the languages and each selection would then be activated as a work order form for a different translator. Only when all the translators have completed their respective tasks would the next task in the process be activated. It is process management from top level to lower level.

Workflow management system:

Workflow management systems provide antomated framework for managing intra- and inter- enterprise business processes,There are two types of tasks in the system: transactional and non-transactional.

Business process execution language for web services:

BPELWS (**Business Process Execution** Language for **Web Services**) is a **process** modeling language. It supersedes XLANG (Microsoft) and WSFL(IBM). It is built on top of WSDL. Today BPEL represents the widely accepted standard in **web service** composition.



UDDI

WS-Transaction

WS-Coordination

WS Composite Application Framework (WS-CAF)

WS-CDL

WSCI

WS-BPEL

WSFL, XLANG

BPML

WSCL, CS-WS

WS-Policy, WSLA

WS-DL

WS Reliable Messaging

WS-Routing, WS-Addressing

XML Protocol – XMLP (SOAP)

XML & XML Schema, …

HTTP, SMTP, …

repository & discovery

transaction management

choreography

orchestration

conversation description / interaction protocol

non-functional features, QoS

description (interface definition)

advanced messaging

basic messaging

content

transport

Formerly BPEL4WS (BPEL for short)

(i) formerly also WSEL (Web Service Endpoint Language) by IBM (ii) also the research/academic proposal Web Service Offering Language [WSOL]

Both the Web Service Conversation Language (WSCL, by HP) and Conversation Support for Web Services (CS-WS, by IBM) proposals are no more supported