- JavaScript Form Validation
- JavaScript Cookies
- JavaScript Timing Events

```
}
```

JavaScript Form Validation

1. Required Fields
2. E-mail Validation

JavaScript can be used to validate data in HTML forms before sending off the content to a server.
Form data that typically are checked by a JavaScript could be:

1. Has the user left required fields empty?
2. Has the user entered a valid e-mail address?
3. Has the user entered a valid date?
4. Has the user entered text in a numeric field?

1. Required Fields

The following function checks whether a required field has been left empty. If the required field is blank, an alert is displayed, and the function returns false. If a value is entered, the function returns true (means that data is OK):

```
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value==””)
{
alert(alerttxt);return false;
}
else
{
return true;
}
}
}
```

The entire script with the HTML form could look something like this:

```
<html>
<head>
<script type=”text/javascript”>
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value==””)
{
alert(alerttxt);return false;
}
else
{
return true;
}
}
}
function validate_form(thisform)
{
with (thisform)
{
if (validate_required(email,”Email must be filled out!”)==false)
{email.focus();return false;}
}
}
</script>
</head>
<body>

<form action=”submit.htm” onsubmit=”return validate_form(this)” method=”post”>
Email: <input type=”text” name=”email” size=”30”>
<input type=”submit” value=”Submit”>
</form>
</body>
</html>
```

2. E-mail Validation

The following function checks whether the content follows the general syntax of an e-mail address.

This means that the input data must contain at least an @ sign and a dot (.). Also, the @ must not be the first character of the e-mail address, and the last dot must at least be one character after the @ sign:

```
function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf(“@”);
dotpos=value.lastIndexOf(“.”);
if (apos<1||dotpos-apos<2)
{alert(alerttxt);return false;}
else {return true;}
}
}
```

The entire script with the HTML form could look something like this:

```
<html>
<head>
<script type=”text/javascript”>
```

```
function validate_email(field,alerttxt)
{

with (field)
{
apos=value.indexOf(“@”);
dotpos=value.lastIndexOf(“.”);
if (apos<1||dotpos-apos<2)
{alert(alerttxt);return false;}
else {return true;}
}
}
function validate_form(thisform)
{
with (thisform)
{
if (validate_email(email,”Not a valid e-mail address!”)==false)
{email.focus();return false;}
}
}
</script>
</head>
<body>
<form action=”submit.htm” onsubmit=”return validate_form(this);” method=get”>
Email: <input type=”text” name=”email” size=”30”>
<input type=”submit” value=”Submit”>
</form>
</body>
</html>
```

Javascript Cookies

1. What Is a Cookie?
2. Create and Store a Cookie.

A cookie is often used to identify a user.

1. What Is a Cookie?
A cookie is a variable that is stored on the visitor’s computer. Each time the same computer requests a page with a browser, it sends the cookie, too. With JavaScript, you can both create and retrieve cookie values.

Examples of cookies:

Name cookie. The first time a visitor arrives on your Web page, she must fill in her name. The name then is stored in a cookie. Next time the visitor arrives at your page, she could get a welcome message like “Welcome Jane Doe!” The name is retrieved from the stored cookie.

Password cookie. The first time a visitor arrives on your Web page, she must fill in a password. The password then is stored in a cookie. Next time the visitor arrives at your page, the password is retrieved from the cookie.

Date cookie. The first time a visitor arrives to your Web page, the current date is stored in a cookie. Next time the visitor arrives at your page, she could get a message like “Your last visit was on Tuesday, August 11, 2005!” The date is retrieved from the stored cookie.

2. Create and Store a Cookie
In this example we create a cookie that stores the name of a visitor. The first time a visitor arrives at the Web page, she is asked to fill in her name. The name then is stored in a cookie. The next time the visitor arrives at the same page, she sees a welcome message.

First, we create a function that stores the name of the visitor in a cookie variable:

```
function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
}
```

The parameters of the preceding function hold the name of the cookie, the value of the cookie, and the number of days until the cookie expires.

In the preceding function, we first convert the number of days to a valid date and then we add the number of days until the cookie should expire. After that, we store the cookie name, cookie value, and the expiration date in the document.cookie object.

Then we create another function that checks whether the cookie has been set:

```
function getCookie(c_name)
{
if (document.cookie.length>0)
{
c_start=document.cookie.indexOf(c_name + "=");
if (c_start!=-1)
{
c_start=c_start + c_name.length+1;
c_end=document.cookie.indexOf(";",c_start);
if (c_end==-1) c_end=document.cookie.length;
return unescape(document.cookie.substring(c_start,c_
end));
}
}
return "";
}
```

The preceding function first checks whether a cookie is stored at all in the document.cookie object. If the document.cookie object holds some cookies, then check to see whether our specific cookie is stored. If our cookie is found, then return the value; if not, return an empty string.

Last, we create the function that displays a welcome message if the cookie is set, and if the cookie is not set, it displays a prompt box asking for the name of the user:

```
function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
{
alert('Welcome again '+username+'!');
}
else
{
username=prompt('Please enter your name:',"");
if (username!=null && username!="")
{
setCookie('username',username,365);
}
}
}
```

The following example runs the checkCookie() function when the page loads.

```
<html>
<head>
<script type="text/javascript">
function getCookie(c_name)
{
if (document.cookie.length>0)
{
c_start=document.cookie.indexOf(c_name + "=");
if (c_start!=-1)
{
c_start=c_start + c_name.length+1 ;
c_end=document.cookie.indexOf(";",c_start);
if (c_end==-1) c_end=document.cookie.length
return unescape(document.cookie.substring(c_start,c_
end));
}
}
return ""
}
function setCookie(c_name,value,expiredays)
{
var exdate=new Date();
exdate.setDate(exdate.getDate()+expiredays);
document.cookie=c_name+ "=" +escape(value)+((expiredays==null) ? "" : ";
expires="+exdate.toGMTString());
}
function checkCookie()
{
username=getCookie('username');
if (username!=null && username!="")
{
alert('Welcome again '+username+'!');
}
else
{
username=prompt('Please enter your name:',"");
if (username!=null && username!="")
{
setCookie('username',username,365);
}
}
}
</script>
</head>
<body onLoad="checkCookie()">
</body>
</html>
```

JavaScript Timing Events

1. The setTimeout() Method
2. The clearTimeout() Method

With JavaScript, it is possible to execute some code after a specified time interval.

This is called timing events.
It’s very easy to time events in JavaScript. The two key methods that are used are as follows:

1. setTimeout()—Executes a code some time in the future
2. clearTimeout()—Cancels the setTimeout()

1. The setTimeout() Method
The syntax is as follows:

```
var t=setTimeout("javascript statement",milliseconds);
```

The setTimeout() method returns a value. In the preceding statement, the value is stored in a variable called t. If you want to cancel this setTimeout(), you can refer to it using the variable name.
The first parameter of setTimeout() is a string that contains a JavaScript statement.

This statement could be a statement like "alert('5 seconds!')" or a call to a function, like "alertMsg()".
The second parameter indicates how many milliseconds from now you want to execute the first parameter.

When the button is clicked in the following example, an alert box is displayed after 3 seconds.

```
<html>
<head>
<script type="text/javascript">
function timedMsg()
{
var t=setTimeout("alert('I am displayed after 3 seconds!')",3000);
}
</script>
</head>
<body>
<form>
<input type="button" value="Display alert box!" onClick="timedMsg()" />
</form>
</body>
</html>
```

To get a timer to work in an infinite loop, you must write a function that calls itself.
In the following example, when a button is clicked, the input field starts to count (forever) starting at 0.

Notice that you also have a function that checks whether the timer is already running, to avoid creating additional timers if the button is clicked more than once.

```
<html>
<head>
<script type="text/javascript">
var c=0;
var t;
var timer_is_on=0;
function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}
function doTimer()
{
if (!timer_is_on)
{
timer_is_on=1;
timedCount();
}
}
</script>
</head>
<body>
<form>
<input type="button" value="Start count!" onClick="doTimer()">
<input type="text" id="txt">
</form>
<p>Click on the button above. The input field will count forever, starting at 0.</p>
</body>
</html>
```

The following example is another simple timing using the setTimeout() method.

```
<html>
<head>
<script type="text/javascript">
function timedText()
{
var t1=setTimeout("document.getElementById('txt').value='2
seconds!'",2000);
var t2=setTimeout("document.getElementById('txt').value='4
seconds!'",4000);
var t3=setTimeout("document.getElementById('txt').value='6
seconds!'",6000);
}
</script>
</head>
<body>
```

```
<form>
<input type="button" value="Display timed text!"
onclick="timedText()" />
<input type="text" id="txt" />
</form>
<p>Click on the button above. The input field will tell you
when two, four, and six seconds have passed.</p>
</body>
</html>
```

The following example shows a clock created with a timing event.

```
<html>
<head>
<script type="text/javascript">
function startTime()
{
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
// add a zero in front of numbers<10
m=checkTime(m);
s=checkTime(s);
document.getElementById('txt').innerHTML=h+":"+m+":"+s;
t=setTimeout('startTime()',500);
}
function checkTime(i)
{
if (i<10)
{
i="0" + i;
}
return i;
}
</script>
</head>

<body onload="startTime()">
<div id="txt"></div>
</body>
</html>
```

2. The clearTimeout() Method

The syntax is as follows:
clearTimeout(setTimeout_variable)

The following example is the same as the previous infinite loop example. The only difference is that we have now added a "Stop Count!" button that stops the timer.

```
<html>
<head>
<script type="text/javascript">
var c=0;
var t;
var timer_is_on=0;
function timedCount()
{
document.getElementById('txt').value=c;
c=c+1;
t=setTimeout("timedCount()",1000);
}
function doTimer()
{
if (!timer_is_on)
{
timer_is_on=1;
timedCount();
}
}
function stopCount()
{
clearTimeout(t);
timer_is_on=0;
}
</script>
</head>
<body>
<form>
<input type="button" value="Start count!" onclick="doTimer()"
/>
<input type="text" id="txt" />
<input type="button" value="Stop count!" onclick="stopCount()" />
</form>
<p>
Click on the "Start count!" button above to start the timer.
The input field will count forever, starting at 0. Click on
the "Stop count!" button to stop the counting. Click on
the "Start count!" button to start the timer again.
</p>
</body>
</html>
```