1. HTML Introduction

1.1. What is HTML5?

HTML5 is the latest standard for HTML. The previous version of HTML, HTML 4.01, came in 1999, and the internet has changed significantly since then.

HTML5 was designed to replace both HTML 4, XHTML, and the HTML DOM Level 2.

It was specially designed to deliver rich content without the need for additional plugins. The current version delivers everything from animation to graphics, music to movies, and can also be used to build complicated web applications.

HTML5 is also cross-platform. It is designed to work whether you are using a PC, or a Tablet, a Smartphone, or a Smart TV.

1.2. How Did HTML5 Get Started?

HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

WHATWG was working with web forms and applications, and W3C was working with XHTML 2.0. In 2006, they decided to cooperate and create a new version of HTML.

Some rules for HTML5 were established:

- New features should be based on HTML, CSS, DOM, and JavaScript
- The need for external plugins (like Flash) should be reduced
- Error handling should be easier than in previous versions
- Scripting has to be replaced by more markup
- HTML5 should be device-independent
- The development process should be visible to the public

1.3. The HTML5 <!DOCTYPE>

In HTML5 there is only one DOCTYPE declaration, and it is very simple:

```
<!DOCTYPE html>
```

1.4. A Minimum HTML5 Document

Below is a simple HTML5 document, with the minimum of required tags:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>

<body>
Content of the document......
</body>

</html>
```

1.5. HTML5 - New Features

Some of the most interesting new features in HTML5 are:

- The <canvas> element for 2D drawing
- The <video> and <audio> elements for media playback
- Support for local storage
- New content-specific elements, like <article>, <footer>, <header>, <nav>, <section>
- New form controls, like calendar, date, time, email, url, search

1.6. Browser Support for HTML5

All major browsers (Chrome, Firefox, Internet Explorer, Safari, Opera) support the new HTML5 elements and APIs, and continue to add new HTML5 features to their latest versions.

The HTML 5 working group includes AOL, Apple, Google, IBM, Microsoft, Mozilla, Nokia, Opera, and hundreds of other vendors.

2. HTML5 New Elements

2.1. New Elements in HTML5

The internet, and the use of the internet, has changed a lot since 1999, when HTML 4.01 became a standard.

Today, several elements in HTML 4.01 are obsolete, never used, or not used the way they were intended. All those elements are removed or re-written in HTML5.

To better handle today's internet needs, HTML5 has also included new elements for drawing graphics, displaying media content, for better page structure and better form handling, and several new APIs, such as drag and drop, get the geographical position of a user, store local data, and more.

Below is a list of the new HTML elements, introduced by HTML5, and a description of what they are used for.

2.2. The New <canvas> Element

Note: The links in the tables below point to our HTML5 Reference. However, you will learn more about these new elements in this tutorial.

Tag	Description
<canvas>	Defines graphic drawing using JavaScript

2.3. New Media Elements

Tag	Description
<audio>	Defines sound or music content
<embed>	Defines containers for external applications (like plug-ins)
<source>	Defines sources for <video> and <audio>
<track>	Defines tracks for <video> and <audio>
<video>	Defines video or movie content

2.4. New Form Elements

Tag	Description
<datalist>	Defines pre-defined options for input controls
<keygen>	Defines a key-pair generator field (for forms)
<output>	Defines the result of a calculation

2.5. New Semantic/Structural Elements

HTML5 offers new elements for better structure:

Tag	Description
<article>	Defines an article in the document
<aside>	Defines content aside from the page content
<bdi>	Defines a part of text that might be formatted in a different direction from other text outside it
<details>	Defines additional details that the user can view or hide
<dialog>	Defines a dialog box or window
<figcaption>	Defines a caption for a <figure> element
<figure>	Defines self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for the document or a section
<header>	Defines a header for the document or a section
<main>	Defines the main content of a document
<mark>	Defines marked or highlighted text
<menuitem>	Defines a command/menu item that the user can invoke from a popup menu
<meter>	Defines a scalar measurement within a known range (a gauge)
<nav>	Defines navigation links in the document
<progress>	Defines the progress of a task
<rp>	Defines what to show in browsers that do not support ruby annotations
<rt>	Defines an explanation/pronunciation of characters (for East Asian typography)
<ruby>	Defines a ruby annotation (for East Asian typography)
<section>	Defines a section in the document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time
<wbr>	Defines a possible line-break

2.6. Removed Elements

The following HTML 4.01 elements has been removed from HTML5:

- <acronym>
- <applet>
- <basefont>
- <big>
-
- <dir>
- <font>
- <frame>
- <frameset>
- <noframes>
- <strike>
- <tt>

3. HTML5 Semantic Elements

Semantic = Meaning.

Semantic elements = Elements with meaning.

3.1. What are Semantic Elements?

A semantic element clearly describes its meaning to both the browser and the developer.

Examples of **non-semantic** elements: and - Tells nothing about its content.

Examples of **semantic** elements: <form>, <table>, and <img> - Clearly defines its content.

3.2. Browser Support

Internet Explorer 9+, Firefox, Chrome, Safari and Opera supports the semantic elements described in this chapter.

Note: Internet Explorer 8 and earlier does not support these elements. However, there is a solution. Look at the end of this chapter.

3.3. New Semantic Elements in HTML5

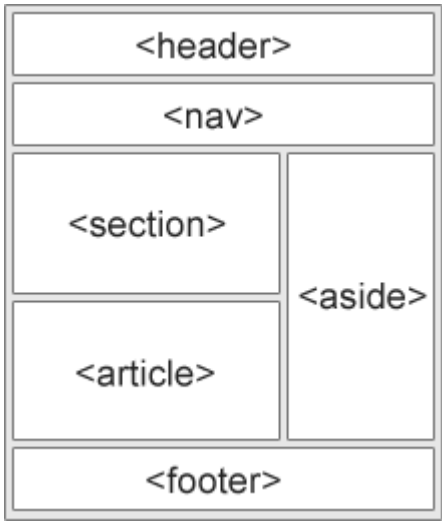
Many of existing web sites today contains HTML code like this:

```
<div id="nav">, <div class="header">, or <div id="footer">
```

, to indicate navigation links, header, and footer.

HTML5 offers new semantic elements to clearly define different parts of a web page:

- <header>
- <nav>
- <section>
- <article>
- <aside>
- <figure>
- <figcaption>
- <footer>
- <details>
- <summary>
- <mark>
- <time>



3.4. HTML5 <section> Element

The <section> element defines a section in a document.

According to W3C's HTML5 documentation: "A section is a thematic grouping of content, typically with a heading."

Example

```
<section>
  <h1>WWF</h1>
  <p>The World Wide Fund for Nature (WWF) is....</p>
</section>
```

3.5. HTML5 <article> Element

The <article> element specifies independent, self-contained content.

An article should make sense on its own and it should be possible to distribute it independently from the rest of the web site.

Examples of where an <article> element can be used:

- Forum post
- Blog post
- News story
- Comment

Example

```
<article>
  <h1>Internet Explorer 9</h1>
  <p>Windows Internet Explorer 9 (abbreviated as IE9)
was released to
  the  public on March 14, 2011 at 21:00 PDT.....</p>
</article>
```

3.6. HTML5 <nav> Element

The <nav> element defines a set of navigation links.

The <nav> element is intended for large blocks of navigation links. However, not all links in a document should be inside a <nav> element!

Example

```
<nav>
<a href="/html/">HTML</a> |
<a href="/css/">CSS</a> |
<a href="/js/">JavaScript</a> |
<a href="/jquery/">jQuery</a>
</nav>
```

3.7. HTML5 <aside> Element

The <aside> element defines some content aside from the content it is placed in (like a sidebar).

The aside content should be related to the surrounding content.

Example

```
<p>My family and I visited The Epcot center this
summer.</p>

<aside>
  <h4>Epcot Center</h4>
  <p>The Epcot Center is a theme park in Disney
World, Florida.</p>
</aside>
```

3.8. HTML5 <header> Element

The <header> element specifies a header for a document or section.

The <header> element should be used as a container for introductory content.

You can have several <header> elements in one document.

The following example defines a header for an article:

Example

```
<article>
  <header>
    <h1>Internet Explorer 9</h1>
    <p><time pubdate
datetime="2011-03-15"></time></p>
  </header>
  <p>Windows Internet Explorer 9 (abbreviated as IE9)
```

```
      was released to
        the  public on March 14, 2011 at 21:00 PDT.....</p>
      </article>
```

3.9. HTML5 <footer> Element

The <footer> element specifies a footer for a document or section.

A <footer> element should contain information about its containing element.

A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

You can have several <footer> elements in one document.

Example

```
      <footer>
        <p>Posted by: Hege Refsnes</p>
        <p><time pubdate datetime="2012-03-01"></time></p>
      </footer>
```

3.10. HTML5 <figure> and <figcaption> Elements

The <figure> tag specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.

While the content of the <figure> element is related to the main flow, its position is independent of the main flow, and if removed it should not affect the flow of the document.

The <figcaption> tag defines a caption for a <figure> element.

The <figcaption> element can be placed as the first or last child of the <figure> element.

Example

```
      <figure>
        <img src="img_pulpit.jpg" alt="The Pulpit Rock"
      width="304" height="228">
        <figcaption>Fig1. - The Pulpit Pock,
      Norway.</figcaption>
      </figure>
```

3.11. Can We Start Using These Semantic Elements?

The elements explained above are all block elements (except <figcaption>).

To get these elements to work properly in older browsers, set the display property to block in your style sheet (this causes older browsers to render these elements correctly):

```
header, section, footer, aside, nav, main, article, figure
{
display: block;
}
```

3.12. Problem With Internet Explorer 8 And Earlier

IE8 and earlier does not know how to render CSS on elements that it doesn't recognize. You cannot style new HTML5 elements like <header>, <section>, <footer>, <aside>, <nav>, <article>, <figure>.

Thankfully, Sjoerd Visscher has discovered a JavaScript workaround called **HTML5 Shiv**; to enable styling of HTML5 elements in versions of Internet Explorer prior to version 9.

You can download and read more about the HTML5 Shiv at: http://code.google.com/p/html5shiv/

To enable the HTML5 Shiv (after downloading), insert the following code into the <head> element:

```
<!--[if lt IE 9]>
<script src="html5shiv.js"></script>
<![endif]-->
```

The code above is a comment that only versions earlier than IE9 reads. It must be placed in the <head> element because Internet Explorer needs to know about the elements before it renders them.

3.13. Semantic Elements in HTML5

Below is an alphabetical list of the new semantic elements in HTML5. The links goes to our complete HTML5 Reference.

Tag	Description
<article>	Defines an article
<aside>	Defines content aside from the page content
<details>	Defines additional details that the user can view or hide
<figcaption>	Defines a caption for a <figure> element
<figure>	Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for a document or section
<header>	Specifies a header for a document or section
<main>	Specifies the main content of a document

<mark>	Defines marked/highlighted text
<nav>	Defines navigation links
<section>	Defines a section in a document
<summary>	Defines a visible heading for a <details> element
<time>	Defines a date/time

4. HTML5 Input Types

4.1. HTML5 New Input Types

HTML5 has several new input types for forms. These new features allow better input control and validation.

This chapter covers the new input types:

- color
- date
- datetime
- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

Not all browsers support all the new input types. However, you can already start using them; If they are not supported, they will behave as regular text fields.

4.2. Input Type: color

The color type is used for input fields that should contain a color.

Example

Select a color from a color picker:

```
Select your favorite color:
<input type="color" name="favcolor">
```

4.3. Input Type: date

The date type allows the user to select a date.

Example

Define a date control:

```
Birthday:
<input type="date" name="bday">
```

4.4. Input Type: datetime

The datetime type allows the user to select a date and time (with time zone).

Example

Define a date and time control (with time zone):

```
Birthday (date and time):
<input type="datetime" name="bdaytime">
```

4.5. Input Type: datetime-local

The datetime-local type allows the user to select a date and time (no time zone).

Example

Define a date and time control (no time zone):

```
Birthday (date and time):
<input type="datetime-local" name="bdaytime">
```

4.6. Input Type: email

The email type is used for input fields that should contain an e-mail address.

Example

Define a field for an e-mail address (will be automatically validated when submitted):

```
E-mail:
<input type="email" name="email">
```

Tip: Safari on iPhone recognizes the email type, and changes the on-screen keyboard to match it (adds @ and .com options).

4.7. Input Type: month

The month type allows the user to select a month and year.

Example

Define a month and year control (no time zone):

```
Birthday (month and year):
<input type="month" name="bdaymonth">
```

4.8. Input Type: number

The number type is used for input fields that should contain a numeric value.

You can also set restrictions on what numbers are accepted:

Example

Define a numeric field (with restrictions):

```
Quantity (between 1 and 5):
<input type="number" name="quantity" min="1" max="5">
```

Use the following attributes to specify restrictions:

- max - specifies the maximum value allowed
- min - specifies the minimum value allowed
- step - specifies the legal number intervals
- value - Specifies the default value

4.9. Input Type: range

The range type is used for input fields that should contain a value from a range of numbers.

You can also set restrictions on what numbers are accepted.

Example

Define a control for entering a number whose exact value is not important (like a slider control):

```
<input type="range" name="points" min="1" max="10">
```

Use the following attributes to specify restrictions:

- max - specifies the maximum value allowed
- min - specifies the minimum value allowed
- step - specifies the legal number intervals
- value - Specifies the default value

4.10. Input Type: search

The search type is used for search fields (a search field behaves like a regular text field).

Example

Define a search field (like a site search, or Google search):

```
Search Google:
<input type="search" name="googlesearch">
```

4.11. Input Type: tel

The tel type is used for input fields that should contain a telephone number.

Example

Define a field for entering a telephone number:

```
Telephone:
<input type="tel" name="usrtel">
```

4.12. Input Type: time

The time type allows the user to select a time.

Example

Define a control for entering a time (no time zone):

```
Select a time:
<input type="time" name="usr_time">
```

4.13. Input Type: url

The url type is used for input fields that should contain a URL address.

The value of the url field is automatically validated when the form is submitted.

Example

Define a field for entering a URL:

```
Add your homepage:
<input type="url" name="homepage">
```

Tip: Safari on iPhone recognizes the url input type, and changes the on-screen keyboard to match it (adds .com option).

4.14. Input Type: week

The week type allows the user to select a week and year.

Example

Define a week and year control (no time zone):

```
Select a week:
<input type="week" name="week_year">
```

4.15. HTML5 <input> Tag

Tag	Description
<input>	Defines an input control

5. HTML5 Form Elements

5.1. HTML5 New Form Elements

HTML5 has the following new form elements:

- <datalist>
- <keygen>
- <output>

Not all browsers support all the new form elements. However, you can already start using them; If they are not supported, they will behave as regular text fields.

5.2. HTML5 <datalist> Element

The <datalist> element specifies a list of pre-defined options for an <input> element.

The <datalist> element is used to provide an "autocomplete" feature on <input> elements. Users will see a drop-down list of pre-defined options as they input data.

Use the <input> element's list attribute to bind it together with a <datalist> element.

Example

An <input> element with pre-defined values in a <datalist>:

```
<input list="browsers">

<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

5.3. HTML5 <keygen> Element

The purpose of the <keygen> element is to provide a secure way to authenticate users.

The <keygen> tag specifies a key-pair generator field in a form.

When the form is submitted, two keys are generated, one private and one public.

The private key is stored locally, and the public key is sent to the server. The public key could be used to generate a client certificate to authenticate the user in the future.

Example

A form with a keygen field:

```
<form action="demo_keygen.asp" method="get">
Username: <input type="text" name="usr_name">
Encryption: <keygen name="security">
<input type="submit">
</form>
```

5.4. HTML5 <output> Element

The <output> element represents the result of a calculation (like one performed by a script).

Example

Perform a calculation and show the result in an <output> element:

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
<input type="range" id="a" value="50">100 +
<input type="number" id="b" value="50">=
<output name="x" for="a b"></output>
</form>
```

5.5. HTML5 New Form Elements

Tag	Description
<datalist>	Specifies a list of pre-defined options for an <input> element
<keygen>	Specifies a key-pair generator field in a form
<output>	Represents the result of a calculation

6. HTML5 Form Attributes

6.1. HTML5 New Form Attributes

HTML5 has several new attributes for <form> and <input>.

New attributes for <form>:

- autocomplete
- novalidate

New attributes for <input>:

- autocomplete
- autofocus
- form
- formaction
- formenctype
- formmethod
- formnovalidate
- formtarget
- height and width
- list
- min and max
- multiple
- pattern (regexp)
- placeholder
- required
- step

6.2. <form> / <input> autocomplete Attribute

The autocomplete attribute specifies whether a form or input field should have autocomplete on or off.

When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

Tip: It is possible to have autocomplete "on" for the form, and "off" for specific input fields, or vice versa.

Note: The autocomplete attribute works with <form> and the following <input> types: text, search, url, tel, email, password, datepickers, range, and color.

Example

An HTML form with autocomplete on (and off for one input field):

```
<form action="demo_form.asp" autocomplete="on">
  First name:<input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  E-mail: <input type="email" name="email" autocomplete="off"><br>
  <input type="submit">
</form>
```

Tip: In some browsers you may need to activate the autocomplete function for this to work.

6.3. <form> novalidate Attribute

The novalidate attribute is a boolean attribute.

When present, it specifies that the form-data (input) should not be validated when submitted.

Example

Indicates that the form is not to be validated on submit:

```
<form action="demo_form.asp" novalidate>
  E-mail: <input type="email" name="user_email">
  <input type="submit">
</form>
```

6.4. <input> autofocus Attribute

The autofocus attribute is a boolean attribute.

When present, it specifies that an <input> element should automatically get focus when the page loads.

Example

Let the "First name" input field automatically get focus when the page loads:

```
First name:<input type="text" name="fname" autofocus>
```

6.5. <input> form Attribute

The form attribute specifies one or more forms an <input> element belongs to.

Tip: To refer to more than one form, use a space-separated list of form ids.

Example

An input field located outside the HTML form (but still a part of the form):

```
<form action="demo_form.asp" id="form1">
  First name: <input type="text" name="fname"><br>
  <input type="submit" value="Submit">
</form>

Last name: <input type="text" name="lname" form="form1">
```

6.6. <input> formaction Attribute

The formaction attribute specifies the URL of a file that will process the input control when the form is submitted.

The formaction attribute overrides the action attribute of the <form> element.

Note: The formaction attribute is used with type="submit" and type="image".

Example

An HTML form with two submit buttons, with different actions:

```
<form action="demo_form.asp">
  First name: <input type="text" name="fname"><br>
```

```
    Last name: <input type="text" name="lname"><br>
    <input type="submit" value="Submit"><br>
    <input type="submit" formaction="demo_admin.asp"
    value="Submit as admin">
  </form>
```

6.7. <input> formenctype Attribute

The formenctype attribute specifies how the form-data should be encoded when submitting it to the server (only for forms with method="post")

The formenctype attribute overrides the enctype attribute of the <form> element.

Note: The formenctype attribute is used with type="submit" and type="image".

Example

Send form-data that is default encoded (the first submit button), and encoded as "multipart/form-data" (the second submit button):

```
  <form action="demo_post_enctype.asp" method="post">
    First name: <input type="text" name="fname"><br>
    <input type="submit" value="Submit">
    <input type="submit" formenctype="multipart/form-data"
    value="Submit as Multipart/form-data">
  </form>
```

6.8. <input> formmethod Attribute

The formmethod attribute defines the HTTP method for sending form-data to the action URL.

The formmethod attribute overrides the method attribute of the <form> element.

Note: The formmethod attribute can be used with type="submit" and type="image".

Example

The second submit button overrides the HTTP method of the form:

```
  <form action="demo_form.asp" method="get">
    First name: <input type="text" name="fname"><br>
    Last name: <input type="text" name="lname"><br>
    <input type="submit" value="Submit">
    <input type="submit" formmethod="post" formaction="demo_post.asp"
    value="Submit using POST">
  </form>
```

6.9. <input> formnovalidate Attribute

The novalidate attribute is a boolean attribute.

When present, it specifies that the <input> element should not be validated when submitted.

The formnovalidate attribute overrides the novalidate attribute of the <form> element.

Note: The formnovalidate attribute can be used with type="submit".

Example

A form with two submit buttons (with and without validation):

```
<form action="demo_form.asp">
  E-mail: <input type="email" name="userid"><br>
  <input type="submit" value="Submit"><br>
  <input type="submit" formnovalidate value="Submit without
validation">
</form>
```

6.10. <input> formtarget Attribute

The formtarget attribute specifies a name or a keyword that indicates where to display the response that is received after submitting the form.

The formtarget attribute overrides the target attribute of the <form> element.

Note: The formtarget attribute can be used with type="submit" and type="image".

Example

A form with two submit buttons, with different target windows:

```
<form action="demo_form.asp">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit as normal">
  <input type="submit" formtarget="_blank"
  value="Submit to a new window">
</form>
```

6.11. <input> height and width Attributes

The height and width attributes specify the height and width of an <input> element.

Note: The height and width attributes are only used with <input type="image">.

Tip: Always specify both the height and width attributes for images. If height and width are set, the space required for the image is reserved when the page is loaded. However, without these attributes, the browser does not know the size of the image, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the images load).

Example

Define an image as the submit button, with height and width attributes:

```
<input type="image" src="submit.gif" alt="Submit" width="48" height="48">
```

6.12. <input> list Attribute

The list attribute refers to a <datalist> element that contains pre-defined options for an <input> element.

Example

An <input> element with pre-defined values in a <datalist>:

```
<input list="browsers">

<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

6.13. <input> min and max Attributes

The min and max attributes specify the minimum and maximum value for an <input> element.

Note: The min and max attributes works with the following input types: number, range, date, datetime, datetime-local, month, time and week.

Example

```
<input> elements with min and max values:
Enter a date before 1980-01-01:
<input type="date" name="bday" max="1979-12-31">

Enter a date after 2000-01-01:
<input type="date" name="bday" min="2000-01-02">

Quantity (between 1 and 5):
<input type="number" name="quantity" min="1" max="5">
```

6.14. <input> multiple Attribute

The multiple attribute is a boolean attribute.

When present, it specifies that the user is allowed to enter more than one value in the <input> element.

Note: The multiple attribute works with the following input types: email, and file.

Example

A file upload field that accepts multiple values:

```
Select images: <input type="file" name="img" multiple>
```

6.15. <input> pattern Attribute

The pattern attribute specifies a regular expression that the <input> element's value is checked against.

Note: The pattern attribute works with the following input types: text, search, url, tel, email, and password.

Tip: Use the global title attribute to describe the pattern to help the user.

Tip: Learn more about regular expressions in our JavaScript tutorial.

Example

An input field that can contain only three letters (no numbers or special characters):

```
Country code: <input type="text" name="country_code"
pattern="[A-Za-z]{3}" title="Three letter country code">
```

6.16. <input> placeholder Attribute

The placeholder attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format).

The short hint is displayed in the input field before the user enters a value.

Note: The placeholder attribute works with the following input types: text, search, url, tel, email, and password.

Example

An input field with a placeholder text:

```
<input type="text" name="fname" placeholder="First name">
```

6.17. <input> required Attribute

The required attribute is a boolean attribute.

When present, it specifies that an input field must be filled out before submitting the form.

Note: The required attribute works with the following input types: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

Example

A required input field:

```
Username: <input type="text" name="usrname" required>
```

6.18. <input> step Attribute

The step attribute specifies the legal number intervals for an <input> element.

Example: if step="3", legal numbers could be -3, 0, 3, 6, etc.

Tip: The step attribute can be used together with the max and min attributes to create a range of legal values.

Note: The step attribute works with the following input types: number, range, date, datetime, datetime-local, month, time and week.

Example

An input field with a specified legal number intervals:

```
<input type="number" name="points" step="3">
```

6.19. HTML5 <input> Tag

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control

7. HTML5 Canvas

The <canvas> element is used to draw graphics, on the fly, on a web page.

The example at the left shows a red rectangle, a gradient rectangle, a multicolor rectangle, and some multicolor text that is drawn onto the canvas.

7.1. What is Canvas?

The HTML5 <canvas> element is used to draw graphics, on the fly, via scripting (usually JavaScript).

The <canvas> element is only a container for graphics. You must use a script to actually draw the graphics.

Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

7.2. Browser Support

Internet Explorer 9+, Firefox, Opera, Chrome, and Safari support the <canvas> element.

Note: Internet Explorer 8 and earlier versions, do not support the <canvas> element.

7.3. Create a Canvas

A canvas is a rectangular area on an HTML page, and it is specified with the <canvas> element.

Note: By default, the <canvas> element has no border and no content.

The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Note: Always specify an id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas.

Tip: You can have multiple <canvas> elements on one HTML page.

To add a border, use the style attribute:

Example

```
<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #000000;">
</canvas>
```

7.4. Draw Onto The Canvas With JavaScript

All drawing on the canvas must be done inside a JavaScript:

Example

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
</script>
```

Example explained:

First, find the <canvas> element:

```
var c = document.getElementById("myCanvas");
```

Then, call its getContext() method (you must pass the string "2d" to the getContext() method):

```
var ctx = c.getContext("2d");
```

The getContext("2d") object is a built-in HTML5 object, with many properties and methods for drawing paths, boxes, circles, text, images, and more.

The next two lines draw a red rectangle:

```
ctx.fillStyle = "#FF0000";
ctx.fillRect(0,0,150,75);
```

The fillStyle property can be a CSS color, a gradient, or a pattern. The default fillStyle is #000000 (black).

The fillRect(*x,y,width,height*) method draws a rectangle filled with the current fill style.

7.5. Canvas Coordinates

The canvas is a two-dimensional grid.

The upper-left corner of the canvas has coordinate (0,0)

So, the fillRect() method above had the parameters (0,0,150,75).

This means: Start at the upper-left corner (0,0) and draw a 150x75 pixels rectangle.

7.6. Canvas - Paths

To draw straight lines on a canvas, we will use the following two methods:

- moveTo(*x,y*) defines the starting point of the line

- lineTo(*x,y*) defines the ending point of the line

To actually draw the line, we must use one of the "ink" methods, like stroke().

Example

Define a starting point in position (0,0), and an ending point in position (200,100). Then use the stroke() method to actually draw the line:

JavaScript:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
```

To draw a circle on a canvas, we will use the following method:

- arc(x,y,r,start,stop)

To actually draw the circle, we must use one of the "ink" methods, like stroke() or fill().

Example

Create a circle with the arc() method:

JavaScript:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(95,50,40,0,2*Math.PI);
ctx.stroke();
```

7.7. Canvas - Text

To draw text on a canvas, the most important property and methods are:

- font - defines the font properties for text
- fillText(*text,x,y*) - Draws "filled" text on the canvas
- strokeText(*text,x,y*) - Draws text on the canvas (no fill)

Using fillText():

Example

Write a 30px high filled text on the canvas, using the font "Arial":

JavaScript:

```
var c = document.getElementById("myCanvas");
```

```
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.fillText("Hello World",10,50);
```

Using strokeText():

Example

Write a 30px high text (no fill) on the canvas, using the font "Arial":

JavaScript:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "30px Arial";
ctx.strokeText("Hello World",10,50);
```

7.8. Canvas - Gradients

Gradients can be used to fill rectangles, circles, lines, text, etc. Shapes on the canvas are not limited to solid colors.

There are two different types of gradients:

- createLinearGradient(*x,y,x1,y1*) - Creates a linear gradient
- createRadialGradient(*x,y,r,x1,y1,r1*) - Creates a radial/circular gradient

Once we have a gradient object, we must add two or more color stops.

The addColorStop() method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

To use the gradient, set the fillStyle or strokeStyle property to the gradient, and then draw the shape, like a rectangle, text, or a line.

Using createLinearGradient():

Example

Create a linear gradient. Fill rectangle with the gradient:

JavaScript:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd = ctx.createLinearGradient(0,0,200,0);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
```

```
ctx.fillRect(10,10,150,80);
```

Using createRadialGradient():

Example

Create a radial/circular gradient. Fill rectangle with the gradient:

JavaScript:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");

// Create gradient
var grd =
ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle = grd;
ctx.fillRect(10,10,150,80);
```

7.9. Canvas - Images

To draw an image on a canvas, we will use the following method:

- drawImage(*image,x,y*)

Example

Draw the image onto the canvas:

JavaScript:

```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
var img = document.getElementById("scream");
ctx.drawImage(img,10,10);
```

7.10. HTML Canvas Reference

For a complete reference of all the properties and methods that can be used with the Canvas object (with try-it examples on every property and method), go to our Canvas Reference.

Tag	Description
<canvas>	Used to draw graphics, on the fly, via scripting (usually JavaScript)

8. HTML5 Inline SVG

HTML5 has support for inline SVG.

SVG Sorry, your browser does not support inline SVG.

8.1. What is SVG?

- SVG stands for Scalable Vector Graphics
- SVG is used to define vector-based graphics for the Web
- SVG defines the graphics in XML format
- SVG graphics do NOT lose any quality if they are zoomed or resized
- Every element and every attribute in SVG files can be animated
- SVG is a W3C recommendation

8.2. SVG Advantages

Advantages of using SVG over other image formats (like JPEG and GIF) are:

- SVG images can be created and edited with any text editor
- SVG images can be searched, indexed, scripted, and compressed
- SVG images are scalable
- SVG images can be printed with high quality at any resolution
- SVG images are zoomable (and the image can be zoomed without degradation)

8.3. Browser Support

Internet Explorer 9+, Firefox, Opera, Chrome, and Safari support inline SVG.

8.4. Embed SVG Directly Into HTML Pages

In HTML5, you can embed SVG elements directly into your HTML page:

Example

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

<svg width="300" height="200">
  <polygon points="100,10 40,180 190,60 10,60 160,180"

style="fill:lime;stroke:purple;stroke-width:5;fill-rule:evenodd;" />
</svg>
```

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

8.5. Differences Between SVG and Canvas

SVG is a language for describing 2D graphics in XML.

Canvas draws 2D graphics, on the fly (with a JavaScript).

SVG is XML based, which means that every element is available within the SVG DOM. You can attach JavaScript event handlers for an element.

In SVG, each drawn shape is remembered as an object. If attributes of an SVG object are changed, the browser can automatically re-render the shape.

Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser. If its position should be changed, the entire scene needs to be redrawn, including any objects that might have been covered by the graphic.

8.6. Comparison of Canvas and SVG

The table below shows some important differences between Canvas and SVG:

Canvas	SVG
• Resolution dependent • No support for event handlers • Poor text rendering capabilities • You can save the resulting image as .png or .jpg • Well suited for graphic-intensive games	• Resolution independent • Support for event handlers • Best suited for applications with large rendering areas (Google Maps) • Slow rendering if complex (anything that uses the DOM a lot will be slow) • Not suited for game applications

9. HTML5 Video

9.1. Video on the Web

Many modern websites show videos. HTML5 provides a standard for showing them. Before HTML5, there was no standard for showing videos/movies on web pages. Before HTML5, videos could only be played with a plug-in (like flash). However, different browsers supported different plug-ins.

HTML5 defines a new element which specifies a standard way to embed a video or movie on a web page: the <video> element.

9.2. Browser Support

Internet Explorer 9+, Firefox, Opera, Chrome, and Safari support the <video> element.

Note: Internet Explorer 8 and earlier versions, do not support the <video> element.

9.3. HTML5 Video - How It Works

To show a video in HTML5, this is all you need:

Example

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

The control attribute adds video controls, like play, pause, and volume.

It is also a good idea to always include width and height attributes. If height and width are set, the space required for the video is reserved when the page is loaded. However, without these attributes, the browser does not know the size of the video, and cannot reserve the appropriate space to it. The effect will be that the page layout will change during loading (while the video loads).

You should also insert text content between the <video> and </video> tags for browsers that do not support the <video> element. The <video> element allows multiple <source> elements. <source> elements can link to different video files. The browser will use the first recognized format.

9.4. Video Formats and Browser Support

Currently, there are 3 supported video formats for the <video> element: MP4, WebM, and Ogg:

- **MP4** = MPEG 4 files with H264 video codec and AAC audio codec.
- **WebM** = WebM files with VP8 video codec and Vorbis audio codec.

- **Ogg** = Ogg files with Theora video codec and Vorbis audio codec.

Browser	MP4	WebM	Ogg
Internet Explorer	YES	NO	NO
Chrome	YES	YES	YES
Firefox	NO **Update:** Firefox 21 running on Windows 7, Windows 8, Windows Vista, and Android now supports MP4	YES	YES
Safari	YES	NO	NO
Opera	NO	YES	YES

9.5. MIME Types for Video Formats

Format	MIME-type
MP4	video/mp4
WebM	video/webm
Ogg	video/ogg

9.6. HTML5 <video> - DOM Methods and Properties

HTML5 has DOM methods, properties, and events for the <video> and <audio> elements.

These methods, properties, and events allow you to manipulate <video> and <audio> elements using JavaScript.

There are methods for playing, pausing, and loading, for example and there are properties (like duration and volume). There are also DOM events that can notify you when the <video> element begins to play, is paused, is ended, etc.

For a full reference go to our HTML5 Audio/Video DOM Reference.

9.7. HTML5 Video Tags

Tag	Description
<video>	Defines a video or movie
<source>	Defines multiple media resources for media elements, such as <video> and <audio>
<track>	Defines text tracks in media players

10. HTML5 Audio

10.1. Audio on the Web

HTML5 provides a standard for playing audio files. Before HTML5, there was no standard for playing audio files on a web page. Before HTML5, audio files had to be played with a plug-in (like flash). However, different browsers supported different plug-ins.

HTML5 defines a new element which specifies a standard way to embed an audio file on a web page: the <audio> element.

10.2. Browser Support

Internet Explorer 9+, Firefox, Opera, Chrome, and Safari support the <audio> element.

Note: Internet Explorer 8 and earlier versions, do not support the <audio> element.

10.3. HTML5 Audio - How It Works

To play an audio file in HTML5, this is all you need:

Example

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>
```

The control attribute adds audio controls, like play, pause, and volume.

You should also insert text content between the <audio> and </audio> tags for browsers that do not support the <audio> element.

The <audio> element allows multiple <source> elements. <source> elements can link to different audio files. The browser will use the first recognized format.

10.4.Audio Formats and Browser Support

Currently, there are 3 supported file formats for the <audio> element: MP3, Wav, and Ogg:

Browser	MP3	Wav	Ogg
Internet Explorer	YES	NO	NO

Chrome	YES	YES	YES
Firefox	NO **Update:** Firefox 21 running on Windows 7, Windows 8, Windows Vista, and Android now supports MP3	YES	YES
Safari	YES	YES	NO
Opera	NO	YES	YES

10.5. MIME Types for Audio Formats

Format	MIME-type
MP3	audio/mpeg
Ogg	audio/ogg
Wav	audio/wav

10.6. HTML5 Audio Tags

Tag	Description
<audio>	Defines sound content
<source>	Defines multiple media resources for media elements, such as <video> and <audio>

11. HTML5 Geolocation

11.1. Locate the User's Position

HTML5 Geolocation is used to locate a user's position . The HTML5 Geolocation API is used to get the geographical position of a user. Since this can compromise user privacy, the position is not available unless the user approves it.

11.2. Browser Support

Internet Explorer 9+, Firefox, Chrome, Safari and Opera support Geolocation.

Note: Geolocation is much more accurate for devices with GPS, like iPhone.

11.3. HTML5 - Using Geolocation

Use the getCurrentPosition() method to get the user's position.

The example below is a simple Geolocation example returning the latitude and longitude of the user's position:

Example

```
<script>
var x = document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.getCurrentPosition(showPosition);
    }
  else{x.innerHTML = "Geolocation is not supported by this
browser.";}
  }
function showPosition(position)
  {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

Example explained:

- Check if Geolocation is supported
- If supported, run the getCurrentPosition() method. If not, display a message to the user
- If the getCurrentPosition() method is successful, it returns a coordinates object to the function

specified in the parameter (showPosition)
- The showPosition() function gets the displays the Latitude and Longitude

The example above is a very basic Geolocation script, with no error handling.

11.4. Handling Errors and Rejections

The second parameter of the getCurrentPosition() method is used to handle errors. It specifies a function to run if it fails to get the user's location:

Example

```
function showError(error)
  {
  switch(error.code)
    {
    case error.PERMISSION_DENIED:
      x.innerHTML = "User denied the request for Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML = "Location information is unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML = "The request to get user location timed out."
      break;
    case error.UNKNOWN_ERROR:
      x.innerHTML = "An unknown error occurred."
      break;
    }
  }
```

Error Codes:

- Permission denied - The user did not allow Geolocation
- Position unavailable - It is not possible to get the current location
- Timeout - The operation timed out

11.5. Displaying the Result in a Map

To display the result in a map, you need access to a map service that can use latitude and longitude, like Google Maps:

Example

```
function showPosition(position)
{
var latlon = position.coords.latitude + "," +
position.coords.longitude;

var img_url = "http://maps.googleapis.com/maps/api/staticmap?
```

```
      center="
      +latlon+"&zoom=14&size=400x300&sensor=false";

      document.getElementById("mapholder").innerHTML = "<img
      src='"+img_url+"'>";
      }
```

11.6. Location-specific Information

This page demonstrated how to show a user's position on a map. However, Geolocation is also very useful for location-specific information.

Examples:

- Up-to-date local information
- Showing Points-of-interest near the user
- Turn-by-turn navigation (GPS)

11.7. The getCurrentPosition() Method - Return Data

The getCurrentPosition() method returns an object if it is successful. The latitude, longitude and accuracy properties are always returned. The other properties below are returned if available.

Property	Description
coords.latitude	The latitude as a decimal number
coords.longitude	The longitude as a decimal number
coords.accuracy	The accuracy of position
coords.altitude	The altitude in meters above the mean sea level
coords.altitudeAccuracy	The altitude accuracy of position
coords.heading	The heading as degrees clockwise from North
coords.speed	The speed in meters per second
timestamp	The date/time of the response

11.8. Geolocation object - Other interesting Methods

- watchPosition() - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
- clearWatch() - Stops the watchPosition() method.

The example below shows the watchPosition() method. You need an accurate GPS device to test this (like iPhone):

Example

```
<script>
var x = document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.watchPosition(showPosition);
    }
  else{x.innerHTML = "Geolocation is not supported by this
browser.";}
  }
function showPosition(position)
  {
  x.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

12. HTML5 Drag and Drop

12.1. Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location. In HTML5, drag and drop is part of the standard, and any element can be draggable.

12.2. Browser Support

Internet Explorer 9+, Firefox, Opera, Chrome, and Safari support drag and drop.

Note: Drag and drop does not work in Safari 5.1.2.

12.3. HTML5 Drag and Drop Example

Example

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev){
ev.preventDefault();
}

function drag(ev){
ev.dataTransfer.setData("Text",ev.target.id);
}

function drop(ev){
ev.preventDefault();
var data = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69">

</body>
</html>
```

It might seem complicated, but lets go through all the different parts of a drag and drop event.

12.4. Make an Element Draggable

First of all: To make an element draggable, set the draggable attribute to true:

```
<img draggable="true">
```

12.5. What to Drag - ondragstart and setData()

Then, specify what should happen when the element is dragged.

In the example above, the ondragstart attribute calls a function, drag(event), that specifies what data to be dragged.

The dataTransfer.setData() method sets the data type and the value of the dragged data:

```
function drag(ev) {
  ev.dataTransfer.setData("Text",ev.target.id);
}
```

In this case, the data type is "Text" and the value is the id of the draggable element ("drag1").

12.6. Where to Drop - ondragover

The ondragover event specifies where the dragged data can be dropped.

By default, data/elements cannot be dropped in other elements. To allow a drop, we must prevent the default handling of the element.

This is done by calling the event.preventDefault() method for the ondragover event:

```
event.preventDefault()
```

12.7. Do the Drop - ondrop

When the dragged data is dropped, a drop event occurs.

In the example above, the ondrop attribute calls a function, drop(event):

```
function drop(ev)
{
ev.preventDefault();
var data = ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data));
}
```

Code explained:

- Call preventDefault() to prevent the browser default handling of the data (default is open as link

on drop)

- Get the dragged data with the dataTransfer.getData("Text") method. This method will return any data that was set to the same type in the setData() method
- The dragged data is the id of the dragged element ("drag1")
- Append the dragged element into the drop element

13. HTML Web Storage

13.1. What is HTML5 Web Storage?

With HTML5, web pages can store data locally within the user's browser. HTML5 web storage, better than cookies. Earlier, this was done with cookies. However, Web Storage is more secure and faster. The data is not included with every server request, but used ONLY when asked for. It is also possible to store large amounts of data, without affecting the website's performance.

The data is stored in name/value pairs, and a web page can only access data stored by itself.

Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.

13.2. Browser Support

Web storage is supported in Internet Explorer 8+, Firefox, Opera, Chrome, and Safari.

Note: Internet Explorer 7 and earlier versions, do not support Web Storage.

13.3. HTML5 Web Storage Objects

HTML5 Web Storage provides two new objects for storing data on the client:

- ***window.localStorage*** - stores data with no expiration date
- ***code.sessionStorage*** - stores data for one session (data is lost when the tab is closed)

Before using web storage, check browser support for localStorage and sessionStorage:

```
if(typeof(Storage)!=="undefined")
  {
  // Code for localStorage/sessionStorage.
  }
else
  {
  // Sorry! No Web Storage support..
  }
```

13.4. The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML=localStorage.getItem("lastname");
```

Example explained:

- Create a localStorage name/value pair with name="lastname" and value="Smith"
- Retrieve the value of "lastname" and insert it into the element with id="result"

The example above could also be written like this:

```
// Store
localStorage.lastname = "Smith";
// Retrieve
document.getElementById("result").innerHTML=localStorage.lastname;
The syntax for removing the "lastname" localStorage item is as follows:
localStorage.removeItem("lastname");
```

Note: Name/value pairs are always stored as strings. Remember to convert them to another format when needed!

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:

Example

```
if (localStorage.clickcount)
  {
  localStorage.clickcount = Number(localStorage.clickcount) + 1;
  }
else
  {
  localStorage.clickcount = 1;
  }
document.getElementById("result").innerHTML="You have clicked the button " +
localStorage.clickcount + " time(s).";
```

13.5. The sessionStorage Object

The sessionStorage object is equal to the localStorage object, **except** that it stores the data for only one session. The data is deleted when the user closes the browser window.

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
if (sessionStorage.clickcount)
  {
  sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
  }
else
  {
```

```
        sessionStorage.clickcount = 1;
        }
    document.getElementById("result").innerHTML="You have clicked the button " +
    sessionStorage.clickcount + " time(s) in this session.";
```

14. HTML5 Application Cache

With HTML5 it is easy to make an offline version of a web application, by creating a cache manifest file.

14.1. What is Application Cache?

HTML5 introduces application cache, which means that a web application is cached, and accessible without an internet connection.

Application cache gives an application three advantages:

1. Offline browsing - users can use the application when they're offline
2. Speed - cached resources load faster
3. Reduced server load - the browser will only download updated/changed resources from the server

14.2. Browser Support

Internet Explorer 10, Firefox, Chrome, Safari and Opera support Application cache.

14.3. HTML5 Cache Manifest Example

The example below shows an HTML document with a cache manifest (for offline browsing):

Example

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">

<body>
The content of the document......
</body>

</html>
```

14.4. Cache Manifest Basics

To enable application cache, include the manifest attribute in the document's <html> tag:

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
...
</html>
```

Every page with the manifest attribute specified will be cached when the user visits it. If the manifest attribute is not specified, the page will not be cached (unless the page is specified directly in the manifest file).

The recommended file extension for manifest files is: ".appcache"

A manifest file needs to be served with the **correct MIME-type**, which is "text/cache-manifest". Must be configured on the web server.

14.5. The Manifest File

The manifest file is a simple text file, which tells the browser what to cache (and what to never cache).

The manifest file has three sections:

- **CACHE MANIFEST** - Files listed under this header will be cached after they are downloaded for the first time
- **NETWORK** - Files listed under this header require a connection to the server, and will never be cached
- **FALLBACK** - Files listed under this header specifies fallback pages if a page is inaccessible

CACHE MANIFEST

The first line, CACHE MANIFEST, is required:

```
CACHE MANIFEST
/theme.css
/logo.gif
/main.js
```

The manifest file above lists three resources: a CSS file, a GIF image, and a JavaScript file. When the manifest file is loaded, the browser will download the three files from the root directory of the web site. Then, whenever the user is not connected to the internet, the resources will still be available.

NETWORK

The NETWORK section below specifies that the file "login.asp" should never be cached, and will not be available offline:

```
NETWORK:
login.asp
```

An asterisk can be used to indicate that all other resources/files require an internet connection:

```
NETWORK:
*
```

FALLBACK

The FALLBACK section below specifies that "offline.html" will be served in place of all files in the /html/ catalog, in case an internet connection cannot be established:

```
FALLBACK:
/html/ /offline.html
```

Note: The first URI is the resource, the second is the fallback.

14.6. Updating the Cache

Once an application is cached, it remains cached until one of the following happens:

- The user clears the browser's cache
- The manifest file is modified (see tip below)
- The application cache is programmatically updated

14.7. Example - Complete Cache Manifest File

```
CACHE MANIFEST
# 2012-02-21 v1.0.0
/theme.css
/logo.gif
/main.js

NETWORK:
login.asp

FALLBACK:
/html/ /offline.html
```

Tip: Lines starting with a "#" are comment lines, but can also serve another purpose. An application's cache is only updated when its manifest file changes. If you edit an image or change a JavaScript function, those changes will not be re-cached. Updating the date and version in a comment line is one way to make the browser re-cache your files.

14.8. Notes on Application Cache

Be careful with what you cache.

Once a file is cached, the browser will continue to show the cached version, even if you change the file on the server. To ensure the browser updates the cache, you need to change the manifest file.

Note: Browsers may have different size limits for cached data (some browsers have a 5MB limit per site).

15. HTML5 Web Workers

15.1. What is a Web Worker?

A web worker is a JavaScript running in the background, without affecting the performance of the page. When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

15.2. Browser Support

Internet Explorer 10, Firefox, Chrome, Safari and Opera support Web workers.

15.3. Check Web Worker Support

Before creating a web worker, check whether the user's browser supports it:

```
if(typeof(Worker) !== "undefined")
  {
  // Yes! Web worker support!
  // Some code.....
  }
else
  {
  // Sorry! No Web Worker support..
  }
```

15.4. Create a Web Worker File

Now, let's create our web worker in an external JavaScript.

Here, we create a script that counts. The script is stored in the "demo_workers.js" file:

```
var i = 0;

function timedCount()
{
i = i + 1;
postMessage(i);
setTimeout("timedCount()",500);
}

timedCount();
```

The important part of the code above is the **postMessage()** method - which is used to post a message back to the HTML page.

Note: Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.

15.5. Create a Web Worker Object

Now that we have the web worker file, we need to call it from an HTML page.

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if(typeof(w) == "undefined")
  {
  w = new Worker("demo_workers.js");
  }
```

Then we can send and receive messages from the web worker.

Add an "onmessage" event listener to the web worker.

```
w.onmessage = function(event){
  document.getElementById("result").innerHTML = event.data;
  };
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.

15.6. Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the terminate() method:

```
w.terminate();
```

15.7. Full Web Worker Example Code

We have already seen the Worker code in the .js file. Below is the code for the HTML page:

Example

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

<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
```

```
<button onclick="stopWorker()">Stop Worker</button>
<br><br>

<script>
var w;

function startWorker()
{
if(typeof(Worker) !== "undefined")
{
  if(typeof(w) == "undefined")
    {
    w = new Worker("demo_workers.js");
    }
  w.onmessage = function (event){
    document.getElementById("result").innerHTML = event.data;
  };
}
else
{
document.getElementById("result").innerHTML="Sorry, your browser
does not support Web Workers...";
}
}

function stopWorker()
{
w.terminate();
}
</script>

</body>
</html>
```

15.8. Web Workers and the DOM

Since web workers are in external files, they do not have access to the following JavaScript objects:

- The window object
- The document object
- The parent object

16. HTML5 Server-Sent Events

HTML5 Server-Sent Events allow a web page to get updates from a server.

16.1. Server-Sent Events - One Way Messaging

A server-sent event is when a web page automatically gets updates from a server. This was also possible before, but the web page would have to ask if any updates were available. With server-sent events, the updates come automatically.

Examples: Facebook/Twitter updates, stock price updates, news feeds, sport results, etc.

16.2. Browser Support

Server-Sent Events are supported in all major browsers, except Internet Explorer.

16.3. Receive Server-Sent Event Notifications

The EventSource object is used to receive server-sent event notifications:

Example

```
var source = new EventSource("demo_sse.php");
source.onmessage = function(event) {
  document.getElementById("result").innerHTML + =event.data + "<br>";
 };
```

Example explained:

- Create a new EventSource object, and specify the URL of the page sending the updates (in this example "demo_sse.php")
- Each time an update is received, the onmessage event occurs
- When an onmessage event occurs, put the received data into the element with id="result"

16.4. Check Server-Sent Events Support

In the example above there were some extra lines of code to check browser support for server-sent events:

```
if(typeof(EventSource) !== "undefined")
  {
  // Yes! Server-sent events support!
  // Some code.....
  }
else
```

```
        {
        // Sorry! No server-sent events support..
        }
```

16.5. Server-Side Code Example

For the example above to work, you need a server capable of sending data updates (like PHP or ASP).

The server-side event stream syntax is simple. Set the "Content-Type" header to "text/event-stream". Now you can start sending event streams.

Code in PHP (demo_sse.php):

```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

$time = date('r');
echo "data: The server time is: {$time}\n\n";
flush();
?>
```

Code in ASP (VB) (demo_sse.asp):

```
<%
Response.ContentType = "text/event-stream"
Response.Expires = -1
Response.Write("data: " & now())
Response.Flush()
%>
```

Code explained:

- Set the "Content-Type" header to "text/event-stream"
- Specify that the page should not cache
- Output the data to send (**Always** start with "data: ")
- Flush the output data back to the web page

16.6. The EventSource Object

In the examples above we used the onmessage event to get messages. But other events are also available:

Events	Description
onopen	When a connection to the server is opened
onmessage	When a message is received
onerror	When an error occurs