CSWD

Introduction to XML
Extensible Markup Language

CSWD

What is XML

- XML stands for eXtensible Markup Language.
- A markup language is used to provide information about a document.
- Tags are added to the document to provide the extra information.
- HTML tags tell a browser how to display the document.
- XML tags give a reader some idea what some of the data means.

CSWD

What is XML Used For?

- XML documents are used to transfer data from one place to another often over the Internet.
- XML subsets are designed for particular applications.
- One is RSS (Rich Site Summary or Really Simple Syndication). It is used to send breaking news bulletins from one web site to another.
- A number of fields have their own subsets. These include chemistry, mathematics, and books publishing.
- Most of these subsets are registered with the W3Consortium and are available for anyone's use.

CSWD

Advantages of XML

- XML is text (Unicode) based.
  - Takes up less space.
  - Can be transmitted efficiently.
- One XML document can be displayed differently in different media.
  - Html, video, CD, DVD,
  - You only have to change the XML document in order to change all the rest.
- XML documents can be modularized. Parts can be reused.

CSWD

Example of an HTML Document

```
<!Doctype html>
<html>
   <head><title>Example</title></head.
<body>
   <h1>This is a large header<h1>
   <h2>Smaller header is here</h2>
</body>
</html>
```

CSWD

Example of an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
   <head><title>Example</title></head>
<body>
   <h1>This is a large header</h1>
   <h2>Smaller header is here</h2>
</body>
</html>
```

CSWD

Typical example of an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<address>
    <name>Alice Person</name>
    <email>aPerson@aol.com</email>
    <phone>212-346-1234</phone>
    <birthday>1985-03-22</birthday>
</address>
```

CSWD

Difference Between HTML and XML

- HTML tags have a fixed meaning and browsers know what it is.
- XML tags are different for different applications, and users know what they mean.
- HTML tags are used for display.
- XML tags are used to describe documents and data
  - NB They can be styled using XSL (XML Style Language) and transformed using XSLT (XSL Transformations).

CSWD

XML Rules

- Tags are enclosed in angle brackets.
- Tags come in pairs with start-tags and end-tags.
- Tags must be properly nested.
  - **<name><email>…</name></email> is not allowed.**
  - **<name><email>…</email><name> is.**
- Tags that do not have end-tags must be terminated by a ‘/’.
  -
 is an html example.

CSWD

More XML Rules

- Tags are case sensitive.
  - **<address> is not the same as <Address>**
- XML in any combination of cases is not allowed as part of a tag.
- Tags may not contain ‘<‘ or ‘&’.
- Tags follow Java naming conventions, except that a single colon and other characters are allowed. They must begin with a letter and may not contain white space.
- Documents must have a single *root* tag that begins the document.

Encoding

- XML (like Java) uses Unicode to encode characters.
- Unicode comes in many flavors. The most common one used in the West is UTF-8.
- UTF-8 is a variable length code. Characters are encoded in 1 byte, 2 bytes, or 4 bytes.
- The first 128 characters in Unicode are ASCII.
- In UTF-8, the numbers between 128 and 255 code for some of the more common characters used in western Europe, such as ã, á, å, or ç.
- Two byte codes are used for some characters not listed in the first 256 and some Asian ideographs.
- Four byte codes can handle any ideographs that are left.
- Those using non-western languages should investigate other versions of Unicode.

CSWD

Well-Formed Documents

- An XML document is said to be well-formed if it follows all the rules.
- An XML parser is used to check that all the rules have been obeyed.
- Recent browsers such as Internet Explorer 5 and Netscape 7 come with XML parsers.
- Parsers are also available for free download over the Internet. One is Xerces, from the Apache open-source project.
- Java 1.4 also supports an open-source parser.

XML Example Revisited

```
<?xml version=“1.0” encoding="UTF-8"?>
<address>
    <name>Alice Person</name>
    <email>aPerson@aol.com</email>
    <phone>212-346-1234</phone>
     <birthday>1985-03-22</birthday>
 </address>
```

- Markup for the data aids understanding of its purpose.
- A flat text file is not nearly so clear.

Alice Person
aPerson@aol.com
212-346-1234
1985-03-22

- The last line looks like a date, but what is it for?

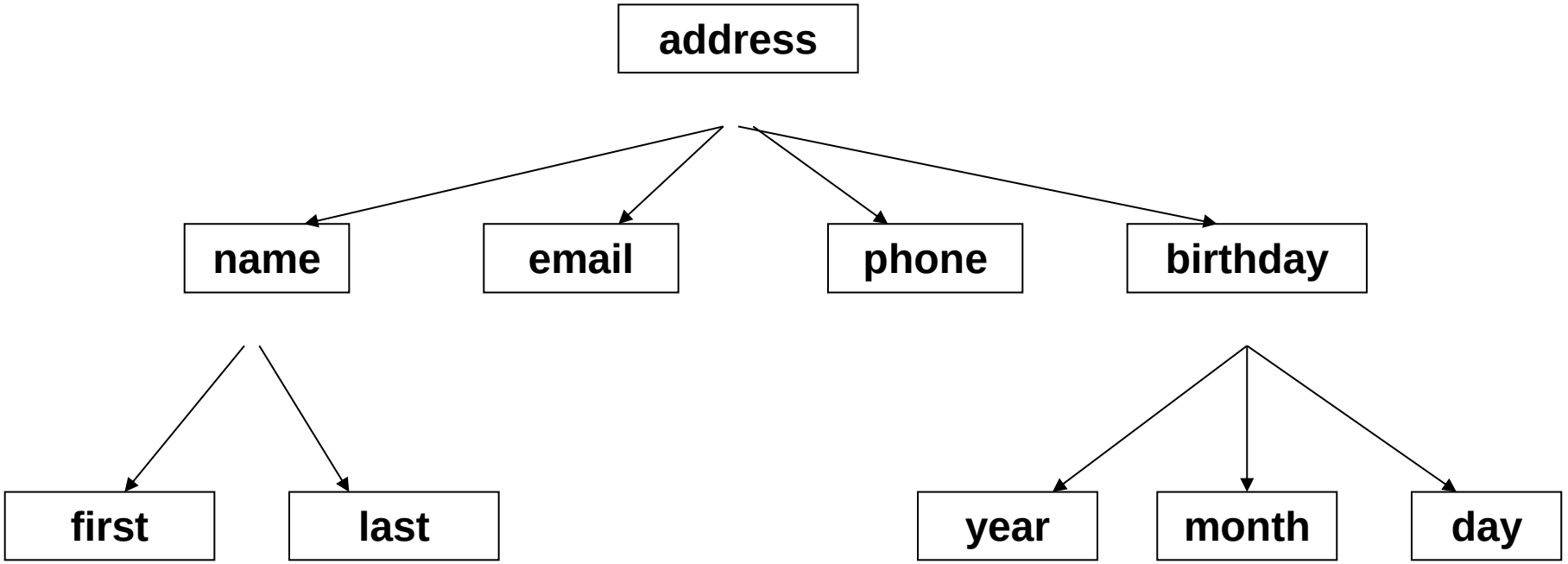
CSWD

Expanded Example

```
<?xml version = “1.0” encoding="UTF-8"?>
<address>
    <name>
        <first>Alice</first>
        <last>Person</last>
    </name>
    <email>aPerson@aol.com</email>
    <phone>123-45-6789</phone>
    <birthday>
        <year>1983</year>
        <month>07</month>
        <day>15</day>
    </birthday>
</address>
```

CSWD

XML Files are Trees



CSWD

XML Trees

- An XML document has a single root node.
- The tree is a general ordered tree.
  - A parent node may have any number of children.
  - Child nodes are ordered, and may have siblings.
- Preorder traversals are usually used for getting information out of the tree.

CSWD

Validity

- A well-formed document has a tree structure and obeys all the XML rules.
- A particular application may add more rules in either a DTD (document type definition) or in a schema.
- Many specialized DTDs and schemas have been created to describe particular areas.
- These range from disseminating news bulletins (RSS) to chemical formulas.
- DTDs were developed first, so they are not as comprehensive as schema.

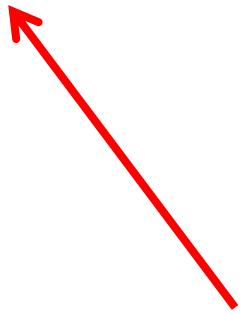
CSWD

rss – xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<rss>
   <channel>
      <item>
            <title>NYT Health</title>
            <link>http://www.nytimes.com/services/xml/rss/nyt/Research.xml</link>
      </item>
      <item>
            <title>CNN Entertainment</title>
            <link>http://rss.cnn.com/rss/cnn_showbiz.rss</link>
      </item>
      <item>
            <title>Flickr current images</title>
            <link>http://api.flickr.com/services/feeds/photos_public.gne?format=rss2</link>
      </item>
   </channel>
</rss>
```

usernames.xml example with ***styling***

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="usernames.xsl"?>
  <employees>
     <employee employee_id="ggl101">
       <FirstName>Jenny</FirstName>
       <LastName>Elisibirt</LastName>
       <gender>Female</gender>
     </employee>
     <employee employee_id="ggl121">
       <FirstName>Alfa</FirstName>
       <LastName>Romeo</LastName>
       <gender>Male</gender>
     </employee>
  </employees>
```



Name of the style file

usernames.xsl

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

<xsl:template match="globalGuideLine">
  <xsl:apply-templates>
    <xsl:sort select="FirstName"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="employee">
  Employee ID:  <xsl:apply-templates select="@employee_id"/>
  First Name:   <xsl:apply-templates select="FirstName"/>
  Last Name:    <xsl:apply-templates select="LastName"/>
  Gender:       <xsl:apply-templates select="gender"/>
  <br/>
  <xsl:text>
  </xsl:text>
</xsl:template>

</xsl:stylesheet>
```

CSWD

Introduction to JSON Data

(sometimes referred to as the x in AJAX – javascript joke!)

CSWD

Data Interchange

- The key idea in Ajax.
- An alternative to page replacement.
- Applications delivered as pages.
- How should the data be delivered?
  - Variety of data formats around
  - Already seen the ***AJAX message pattern*** at play in the lab with text, partial html, XML and **JSON**

CSWD

JSON

- JavaScript Object Notation
- Minimal
- Textual
- Subset of JavaScript
- A Subset of ECMA-262 Third Edition.
- Language Independent,
  - available in many natural languages and programming languages
- Light-weight
- Easy to parse
  - **JSON is not a markup language.**

CSWD

JavaScript Object Notation (JSON)

- Standard for “serializing” data objects, usually in files
- Human-readable, useful for data interchange
- Also useful for representing & storing semistructured data

```
[ {
   "title": "My First entry",
   "slug": "first-entry",
   "description": "Lorem ipsum dolor sit amet.",
   "text": "Lorem ipsum dolor sit amet...",
   "timeCreated": "20/08/1989 12:45",
   "author": "John Doe"
  },
  {
   "title": "My Second entry",
   "slug": "second title entry",
…
```

JavaScript Object Notation (JSON)

- No longer tied to JavaScript
- Parsers for many languages

```
[ {
   "title": "My First entry",
   "slug": "first-entry",
   "description": "Lorem ipsum dolor sit amet.",
   "text": "Lorem ipsum dolor sit amet...",
   "timeCreated": "20/08/1989 12:45",
   "author": "John Doe"
  },
  {
   "title": "My Second entry",
   "slug": "second title entry",
…
```

CSWD

Object

- Objects are unordered containers of key/value pairs
- Objects are wrapped in `{ }`
- `,` separates key/value pairs
- `:` separates keys and values
- Keys are strings
- Values are JSON values
  - struct, record, hashtable, object

Object

```
{ "name":"CSWD",
"exam in module": true,
"grade":"honours",
"level":4,
"format":{
      "type":"elective",
      "lectures":true,
      "tutorials":false }
}
```

```
{
 "Books": [
  {
     "ISBN": "ISBN-0-59-651774-2",
     "Price": 20,
     "Edition": 1,
     "Title": "JavaScript: The Good Parts",
     "Authors": [
        {
           "First_Name": "Douglas",
           "Last_Name": "Crockford"
        }
     ]
  }
 ],
 "Magazines": [
  {
     "Title": "National Geographic",
     "Month": "January",
     "Year": 2008
  },
  {
     "Title": "Newsweek",
     "Month": "February",
     "Year": 2009
  }
 ]
}
```

Basic constructs (recursive)

- Base values

  number, string, boolean, …

- Objects { }

  sets of label-value pairs

- Arrays [ ]

  lists of values

CSWD

Array

- Arrays are ordered sequences of values
- Arrays are wrapped in `[]`
- `,` separates values
- JSON does not talk about indexing.
  - An implementation can start array indexing at 0 or 1.

CSWD

Array

```
["Sunday", "Monday", "Tuesday", "Wednesday",
   "Thursday", "Friday", "Saturday"]


[
    [0, -1, 0],
    [1, 0, 0],
    [0, 0, 1]
]
```

CSWD

Arrays vs Objects

- Use objects when the key names are arbitrary strings.
- Use arrays when the key names are sequential integers.

MIME Media Type

`application/json`

CSWD

Relational Model versus JSON

	Relational	JSON
Structure	Tables	Nested Sets Arrays
Schema	Fixed in advance	"self-describing" Flexible
Queries	Simple expressive languages	∅ widely used
Ordering	None.	Arrays.
Implementation	Native systems.	Coupled with PLs. NoSQL Systems.

JSON - XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<Books>
        <ISBN>ISBN-0-59-651774-2</ISBN>
        <Price>20</Price>
        <Edition>1</Edition>
        <Title>JavaScript: The Good Parts</Title>
        <Authors>
                <First_Name>Douglas</First_Name>
                <Last_Name>Crockford</Last_Name>
        </Authors>
</Books>
<Magazines>
        <Title>National Geographic</Title>
        <Month>January</Month>
        <Year>2008</Year>
</Magazines>
<Magazines>
        <Title>Newsweek</Title>
        <Month>February</Month>
        <Year>2009</Year>
</Magazines>
```

Q[2]. Why is this XML file not **Well-Formed**?

CSWD

XML versus JSON

	XML	JSON
Verbosity	More	Less
Complexity	More	Less
Validity	DTDs widely used XSDs	JSON Schema not widely used
Prog. Interface	Clunky "Impedence mismatch"	More direct
Querying	XPath - XQuery XSLT -	JSON Path JAQL JSON Query

Syntactically valid JSON

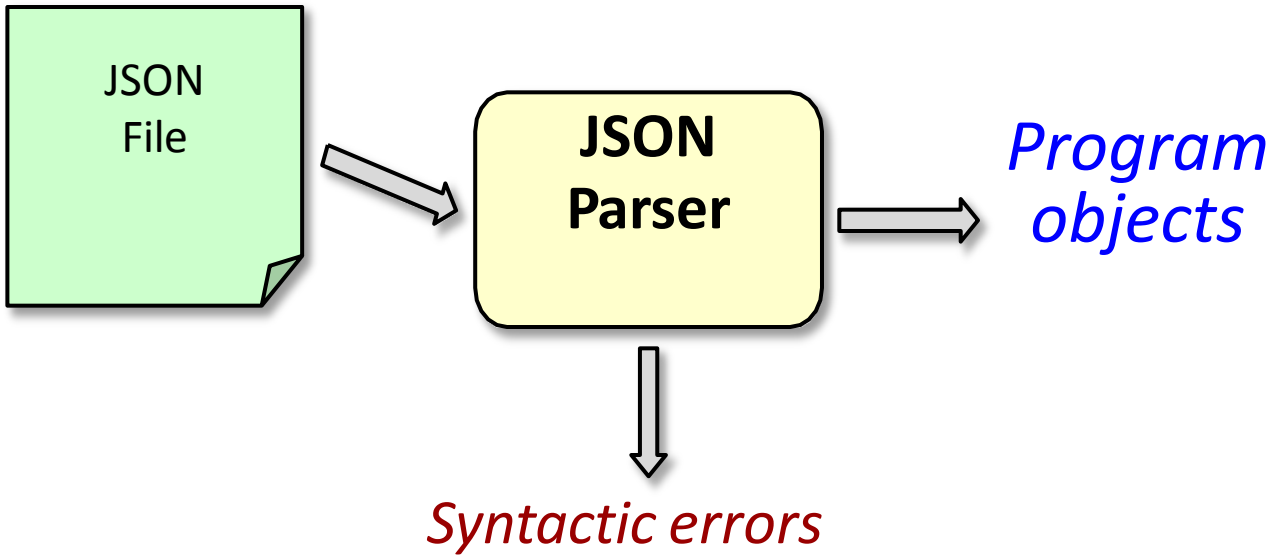
```
{
 "Books": [
  {
     "ISBN": "ISBN-0-59-651774-2",
     "Price": 20,
     "Edition": 1,
     "Title": "JavaScript: The Good Parts",
     "Authors": [
        {
           "First_Name": "Douglas",
           "Last_Name": "Crockford"
        }
     ]
  }
 ],
 "Magazines": [
  {
     "Title": "National Geographic",
     "Month": "January",
     "Year": 2008
  },
  {
     "Title": "Newsweek",
     "Month": "February",
     "Year": 2009
  }
 ]
}
```

Adheres to basic structural requirements

- Sets of label-value pairs
- Arrays of values
- Base values from predefined types

Adheres to basic structural requirements

- Sets of label-value pairs
- Arrays of values
- Base values from predefined types



supplant

```
var template = '<table border="{border}">' +
    '<tr><th>Last</th><td>{last}</td></tr>' +
    '<tr><th>First</th><td>{first}</td></tr>' +
    '</table>';

var data = {
    "first"  : "Jane",
    "last"   : "Doe",
    "border" : "2"
};

mydiv.innerHTML = template.supplant(data);
```

supplant

```
String.prototype.supplant = function (o) {
    return this.replace(/{([^{}]*)}/g,
        function (a, b) {
            var r = o[b];
            return typeof r === 'string' ?
                r : a;
        }
    );
};
```

Data Data Everywhere but not a drop to use!

- Where/How do we store this data?

CSWD

Why Local Storage?

- Data accessed over the internet can never be as fast as accessing data locally
- Data accessed over internet not secure
- HTML5 storage is **on client**

Persistent Local Storage

- Native client applications use operating system to store data such as preferences or runtime state
- Stored in registry, INI files, XML or other places using key/value pairs
- Web applications can’t do this

CSWD

Cookies

- Invented early in Web's history as a way to store persistent data ("magic cookies")
- Small pieces of information about a user stored by Web server as text files on user's computer
- Can be temporary or persistent

Cookies

- Included with every HTTP request – slows down application by transmitting same information repeatedly
- Sends unencrypted data over internet with every HTTP request
- Limited to 4KB data
- Example: filling out a text form field

Cookies not enough

- More storage space
- On the client
- Beyond page refresh
- Not transmitted to server

CSWD

HTML5 Storage

- Provides standardized API
- Implemented natively
- Consistent across browsers
- HTML5 storage is a specification named “Web Storage”
  - Previously part of HTML5 specifications
  - Split into its own specification
  - Different browsers may call it “Local Storage” or “DOM Storage”

Web Application Support

- Supported by latest version of all browsers!

IE 8+	Firefox 3.5+	Safari 4.0+	Chrome 4.0+	Opera 10.5+	IPhone 2.0+	Android 2.0+

- window object
- Before using, detect whether browser supports it

Check for HTML5 Storage

```
function supports_html5_storage() {
        try {
          return 'localStorage' in window
          && window['localStorage'] !== null;
        } catch (e) {
           return false;
        }
      }
```

Or use Modernizr

```
if (Modernizr.localstorage) {
   // window.localStorage is available!
} else {
   // no native support for HTML5 storage :(
   // maybe try dojox.storage or a third-party solution
}
```

Using HTML5 Storage example

localstorage object

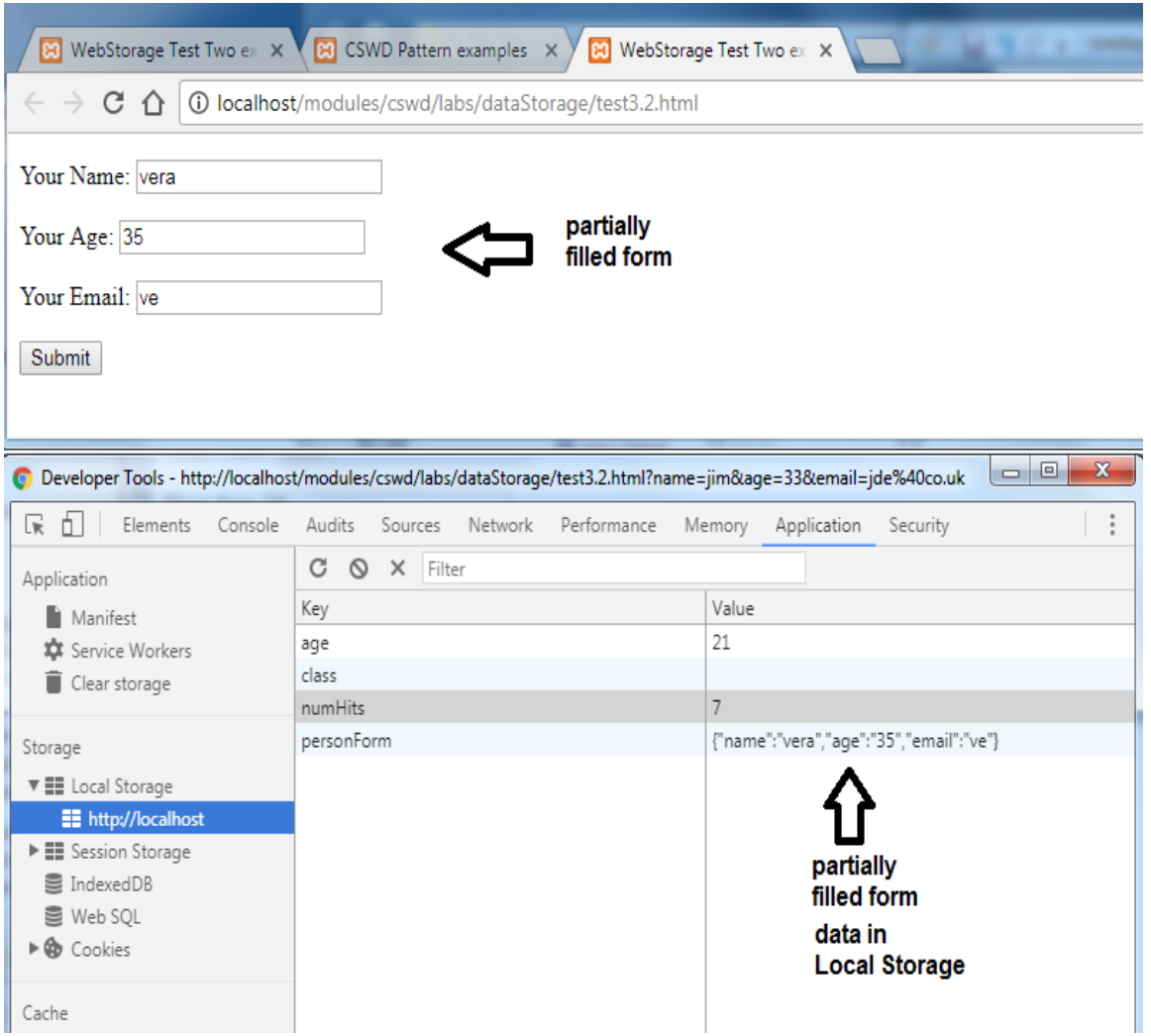
setItem( )

getItem( )

removeItem( )

clear( )

Image opposite shows partial completed form!



Using HTML5 Storage

Tracking changes to the HTML5 storage area

```
if (window.addEventListener) {
  window.addEventListener("storage", handle_storage, false);
} else {
  window.attachEvent("onstorage", handle_storage);
};
```

Using HTML5 Storage

Tracking changes to the HTML5 storage area

The handle_storage callback function will be called with a StorageEvent object, except in Internet Explorer where the event object is stored in window.event.

```
function handle_storage(e) {
  if (!e) { e = window.event; }
}
```

Using HTML5 Storage

StorageEvent Object

PROPERTY	TYPE	DESCRIPTION
`key`	string	the named key that was added, removed, or modified
`oldValue`	any	the previous value (now overwritten), or `null` if a new item was added
`newValue`	any	the new value, or `null` if an item was removed
`url`*	string	the page which called a method that triggered this change

CSWD

Using HTML5 Storage

- Limitations in current browsers:
- 5 MB of storage from each origin.
- Can not ask user for more storage (except for Opera, sort of)

Beyond Key/Value Pairs: Competing Visions

Indexed Database API

Formerly known as WebSimpleDB

Now colloquially referred to as “indexedDB”

Next Week