



Mark-up languages



Overview

- ▶ XML
- ▶ XML Schema
- ▶ XHTML / HTML
- ▶ HTML Forms



Mark-up

Introduction to Mark-up

Marking up text means adding annotations around pieces of it to explicitly indicate properties of those pieces. The mark-up was originally to aid people, mark-up men, to know how to present the text on a page. In HTML, it is still used for this purpose. XML generalises from this, so that mark-up is specific for the purposes of each application. Notice that, as long as mark-up does not overlap, it describes a hierarchy of annotations.



Mark-up

CHAPTER

Introduction to Mark-up

TITLE

Marking up text means adding annotations around pieces of it to explicitly indicate properties of those pieces. The mark-up was originally to aid people, mark-up men, to know how to present the text on a page. In HTML, it is still used for this purpose. XML generalises from this, so that mark-up is specific for the purposes of each application. Notice that, as long as mark-up does not overlap, it describes a hierarchy of annotations.

Mark-up

CHAPTER

Introduction to Mark-up

TITLE

NEW TERM

Marking up text means adding annotations around pieces of it to explicitly indicate properties of those pieces. The mark-up was originally to aid people, mark-up men, to know how to present the text on a page. In HTML, it is still used for this purpose. XML generalises from this, so that mark-up is specific for the purposes of each application. Notice that, as long as mark-up does not overlap, it describes a hierarchy of annotations.

Mark-up

CHAPTER

Introduction to Mark-up TITLE

NEW TERM

Marking up text means adding annotations around pieces of it to explicitly indicate properties of those pieces. The mark-up was originally to aid people, mark-up men, to know how to present the text on a page. In HTML, it is still used for this purpose. XML generalises from this, so that mark-up is specific for the purposes of each application. Notice that, as long as mark-up does not overlap, it describes a hierarchy of annotations.

TECHNICAL
TERM

TECHNICAL
TERM

Mark-up in XML

<chapter><title>Introduction to Mark-up**</title>**

Marking up text means adding annotations around pieces of it to explicitly indicate properties of those pieces. The mark-up was originally to aid people, **<newterm>**mark-up**</newterm>**, to know how to present the text on a page. In **<technical>**HTML**</technical>**, it is still used for this purpose. **<technical>**XML**</technical>** generalises from this, so that mark-up is specific for the purposes of each application. Notice that, as long as mark-up does not overlap, it describes a hierarchy of annotations.**</chapter>**



Mark-up languages

- ▶ A mark-up language is a format for a document, where text is annotated (marked-up) with computer-parseable information on:
 - ▶ how the text should be interpreted,
 - ▶ how it should be presented,
 - ▶ how one piece of text relates to another
 - ▶ etc.



HTML as mark-up

- ▶ One particular mark-up language is Hypertext Markup Language (HTML), which is the language of web pages
- ▶ In HTML, links to other pages and the presentational structure are both encoded in mark-up annotations to the text of the webpage





XML



XML

- ▶ eXtensible Markup Language (XML) is a general-purpose mark-up language
- ▶ XML is used to mark-up arbitrary data, not just readable text documents
- ▶ An HTML document can be seen as a specific kind of XML document (though this is not exactly the case, as we will see later)



Tags

- ▶ To mark up data in XML you precede it with an **opening tag** and follow it with a **closing tag**
- ▶ An opening tag has the form `<NAME>`, where NAME is the name of the tag
- ▶ A closing tag has the form `</NAME>`
- ▶ So, if the tag “title” is used to mark a piece of text as the title of the document it has the form:

`<title>The title of my document</title>`



Elements

- ▶ A part of a document surrounded in opening/closing tags is called an **element**
- ▶ Elements can be nested in a hierarchy, where each tag is understood partly by the other tags which surround it

<element1>

 <element2>

 <element3>

 </element3>

 </element2>

</element1>



Element hierarchy example

- ▶ In the following example, the first <title> is understood as the title of the first chapter and the second title for the second chapter, each enclosed chapter by <chapter> tags

<chapter>

<title>Introduction</title>

This is what I'm going to say...

</chapter>

<chapter>

<title>Conclusions</title>

This is what I've said

</chapter>



Empty elements

- ▶ An element with no content can be abbreviated to a single tag of the form `<NAME/>`
- ▶ Empty elements can be used where the mark-up has meaning regardless of any text
- ▶ For example, in HTML, the `<hr/>` element denotes a horizontal rule across the page

This text is above the horizontal rule

`<hr/>`

This text is below the horizontal rule



Attributes

- ▶ Elements can be parameterised with annotations: **attributes**
- ▶ An attribute consists of a name and a value (in 'quotes'/'“quotes”')
- ▶ Specifies info about the element it annotates
- ▶ Put inside element's opening tag after name

```
<chapter      columns = “two-column”  
              author = “Samhar Mahmoud”>
```

...

```
</chapter>
```

- ▶ There are a few special attributes
 - ▶ **xmlns** Namespace declaration
 - ▶ **xml:lang** The language in which the text is written



Documents

- ▶ An **XML document** contains a single element, the **root** of the hierarchy, enclosing all the other elements
- ▶ Preceding the root element should be a prolog consisting of two special tags:
 - ▶ XML Declaration
 - ▶ Document Type Declaration (optional)
- ▶ The Document Type Declaration was used to specify the **schema** followed by the XML document



XML declaration

- ▶ A special tag `<?xml..?>`, the **XML declaration**, is put before the root to specify the form of XML used in a set of attributes
 - ▶ **Version**: The version of XML used
 - ▶ **Encoding**: The character set used
 - ▶ **Standalone**: Can the document be parsed on its own, or needs other documents to be parsed first?

```
<?xml      version = "1.0"  
          encoding = "UTF-16"  
          standalone = "yes"?>
```



XML document example

```
<?xml version = "1.0"?>
```

```
<book>
```

```
  <chapter>
```

```
    <title>Introduction</title>
```

```
    This is what I'm going to say...
```

```
  </chapter>
```

```
  <chapter>
```

```
    <title>Conclusions</title>
```

```
    This is what I've said
```

```
  </chapter>
```

```
</book>
```



Entities

- ▶ As XML uses particular characters to delimit the elements and attributes, they cannot be used inside the marked-up text itself
- ▶ Instead, **entities**, special strings, are used in their place
- ▶ When the XML is parsed, the entities are replaced by the character they represent

"	“	<	<
&	&	>	>
'	‘		



XML entities example

Transmitted XML:

```
<book>
```

```
  <title>War &amp; Peace</title>
```

```
  Peace &gt; War
```

```
</book>
```

Parsed text:

War & Peace

Peace > War



Comments

- ▶ Ignored in parsing
- ▶ Used to help those looking at the XML directly (as with source code comments)
- ▶ Written in the form:

`<!-- Comment -->`
- ▶ May appear anywhere in document after the XML declaration



XML and applications

- ▶ Each XML-based application understands particular tags
 - ▶ A web browser understands tags describing the content of a web page
 - ▶ An employee database understands tags relating to employee details
- ▶ With XML being exchanged between hosts and between applications, it is important to distinguish tags with different meanings or uses in different contexts



Namespaces

- ▶ A single tag name may be used for different purposes and have different meanings
 - ▶ <title> may mean a chapter title in an XML book
 - ▶ <title> may mean Mr/Mrs/Miss/Dr in an employee database
- ▶ **Namespaces** are used to allow software to know how to interpret tags
- ▶ A namespace is identified by a URI, and every element and attribute name has a namespace
 - ▶ <http://www.w3.org/TR/REC-html40>
 - ▶ <http://docbook.org/ns/docbook>



Namespace declarations

- ▶ There are two ways to give the namespace of an element or attribute:
 - ▶ **Default namespaces** give a namespace to every element in a hierarchy except where overridden
 - ▶ **Namespace prefixes** are identifiers added to an element/attribute name to give its namespace



Default namespaces

- ▶ **xmlns = “NS-URI”** attribute in an element specifies that the element and all elements within it will by default use namespace NS-URI

```
<chapter xmlns = “http://book.namespace”>  
  <section>  
    <picture xmlns = “http://pictures”/>  
  </section>  
</chapter>
```

- ▶ chapter & section namespace: “http://book.namespace”
- ▶ picture namespace: “http://pictures”



Namespace prefixes

- ▶ A prefix can be defined for a namespace using the attribute **xmlns:PREFIX = “NS-URI”**
- ▶ PREFIX is a code preceding each name that has namespace NS-URI

```
<book:chapter  
    xmlns:book = “http://book.namespace”  
    xmlns:person = “http://per.namespace”  
    person:author = “Samhar Mahmoud”>  
    <book:section>...</book:section>  
</book:chapter>
```



Mixed default and prefixes

```
<outer xmlns = "http://one">  
  <middle xmlns:t = "http://two">  
    <t:inner> some text </t:inner>  
  </middle>  
</outer>
```

► Namespaces

- outer: http://one
- middle: http://one
- inner: http://two





XML Schema



XML Schema

- ▶ XML schema documents are, themselves, written in XML
- ▶ An XML Schema document is often referred to as an XSD (XML Schema Definition) and given the file extension .xsd
- ▶ XML Schema 1.0 documents use the namespace:

<http://www.w3.org/2001/XMLSchema>



XML Schema documents

- ▶ The root element is `<schema>`
- ▶ The **target namespace** is the namespace of the elements/attributes defined in the schema (`http://www.example.org` in the example below)
- ▶ It is declared as an attribute of the `<schema>` element:

```
<xs:schema targetNamespace = "http://www.example.org"  
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

...

```
</xs:schema>
```

- ▶ We will use **xs:** as prefix for XML Schema namespace



Simple types

- ▶ A simple type is the type for the text inside elements, and for attribute values
- ▶ XML Schema has pre-defined simple types such as:
 - ▶ string
 - ▶ integer
 - ▶ decimal
 - ▶ boolean
 - ▶ time
 - ▶ date
 - ▶ dateTime



Simple type elements

- ▶ To assert, in XML Schema, that an element has a given type, the following structure is used:

Schema format

```
<element name="ELEM-NAME" type="TYPE"/>
```

Schema example

```
<xs:element      name = "numberOfPages"  
                  type = "xs:integer"/>
```

XML conforming to schema

```
<numberOfPages>25</numberOfPages>
```



Simple type attributes

- ▶ An attribute always has a simple type and is defined in a similar fashion

Schema format

```
<attribute name="ATTR-NAME" type="TYPE"/>
```

Schema example

```
<xs:attribute      name = "numberOfPages"  
                  type = "xs:integer"/>
```

XML conforming to schema

```
<book numberOfPages = "56"/>
```



Default values

- ▶ We can give a default value for the content of an element/attribute or fix it to one value:

```
<element    name = "ELEM-NAME"  
           type = "TYPE"  
           default = "DEFAULT-VALUE"/>
```

```
<element    name = "ELEM-NAME"  
           type = "TYPE"  
           fixed = "FIXED-VALUE"/>
```



Enumerator simple types

- ▶ Custom simple types can be defined
- ▶ Simple types can be created as enumerator, which limits the content of an element to a set of acceptable values.

```
<xs:simpleType name = "clothSizeType">  
  <xs:restriction base = "xs:string">  
    <xs:enumeration value="S" />  
    <xs:enumeration value="M" />  
    <xs:enumeration value="L" />  
    <xs:enumeration value="XL" />  
  </xs:restriction>  
</xs:simpleType>
```

Pattern simple types

- ▶ Simple types can be created as **patterns** (regular expressions) which values must match

```
<xs:simpleType name = "postCode">  
  <xs:restriction base = "xs:string">  
    <xs:pattern  
      value = "[A-Z]{2}[0-9]{2}[A-Z]{2}"/>  
  </xs:restriction>  
</xs:simpleType>
```

Pattern examples

- ▶ One of x, y or z characters:

```
<xs:pattern value="[xyz]"/>
```

- ▶ Zero or more lower case letters:

```
<xs:pattern value="([a-z])*"/>
```

- ▶ One or more letters:

```
<xs:pattern value="([a-zA-Z])+"/>
```

- ▶ Choice of strings:

```
<xs:pattern value="male|female"/>
```



Embedded complex types

- ▶ To specify the hierarchical structure of an element, a complex type is used
- ▶ A complex element type definition has the form:

```
<xs:element name = "ELEM-NAME">  
  <xs:complexType>  
    ...  
  </xs:complexType>  
</xs:element>
```



Referenced complex types

- ▶ Alternatively, complex types can be defined and named independently
- ▶ Then referenced by one or more elements

```
<xs:complexType name = "TYPE-NAME">  
    ...  
</xs:complexType>
```

```
<xs:element name="NAME" type="TYPE-NAME"/>
```



References and namespaces

- ▶ Remember: elements and types defined in the schema have the target namespace
- ▶ Therefore, references to types/elements must use this namespaces

```
<xs:schema      targetNamespace = "http://red"
                xmlns:tns = "http://red"
...
    <xs:element  name = "book"
                type = "tns:bookType"/>
```



Sequence complex types

- ▶ The **sequence** type requires sub-elements to be present in an order (below, sub-elements are referenced by name)

```
<xs:element name = "book">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "tns:title"/>
      <xs:element ref = "tns:section"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



Sequence XML example

- ▶ The schema on the previous slide allows the following XML

```
<book>  
  <title>Introduction</title>  
  <section>Some text...</section>  
</book>
```

- ▶ The sequence must not be out of order
- ▶ All sub-elements are present



All complex types

- ▶ An **all** type requires all sub-elements to appear, but in any order

```
<xs:element name = "book">
  <xs:complexType>
    <xs:all>
      <xs:element ref = "tns:title"/>
      <xs:element ref = "tns:section"/>
      <xs:element ref = "tns:numberOfPages"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```



Choice complex types

- ▶ A **choice** type requires one of a choice of sub-elements to appear

```
<xs:element name = "book">  
  <xs:complexType>  
    <xs:choice>  
      <xs:element ref = "tns:content"/>  
      <xs:element ref = "tns:synopsis"/>  
    </xs:choice>  
  </xs:complexType>  
</xs:element>
```



Min and max occurrences

- ▶ We can restrict the number of times that a sub-element occurs using **minOccurs** and **maxOccurs** attributes (if not specified, then by default both = “1”)
- ▶ Zero or one occurrence:

```
<xs:element      ref = “tns:content”  
                minOccurs = “0”  
                maxOccurs = “1”/>
```

- ▶ Two or more occurrences:

```
<xs:element      ref = “tns:content”  
                minOccurs = “2”  
                maxOccurs = “unbounded”/>
```



Combining structures

- ▶ We can use one structure within another

```
<xs:element name = "contact">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "tns:name"/>
      <xs:choice>
        <xs:element ref = "tns:email"/>
        <xs:element ref = "tns:postAddress"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



Any type

- ▶ The **any** XSD element can be used where you want to allow the XML to be extended with arbitrary content (elements, attributes)
- ▶ Used where you use element or attribute
- ▶ **anyElement** only allows arbitrary elements
- ▶ **anyAttribute** only allows arbitrary attributes

```
<xs:any/>
```

```
<xs:anyElement/>
```

```
<xs:anyAttribute/>
```



Any example

```
<xs:element name = "book">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "tns:title"/>
      <xs:any      minOccurs = "0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<book>
  <title>Introduction</title>
  <dissertation>My text...</dissertation>
</book>
```

Attributes of elements

- ▶ Attributes are defined after sub-elements in a complex type

```
<xs:element name = "book">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref = "tns:title"/>
      <xs:element ref = "tns:section"/>
    </xs:sequence>
    <xs:attribute name = "author" type = "xs:string"/>
  </xs:complexType>
</xs:element>
```



Interleaved text

- ▶ To interleave text and mark-up in an element (as in an HTML body), we need to declare its type mixed

```
<xs:element name = "book">  
  <xs:complexType mixed = "true">  
    <xs:sequence>  
      <xs:element ref = "tns:title"/>  
      <xs:element ref = "tns:section"/>  
      ...  
      
    
</xs:element>
```

```
<book><title>...</title>The following sections will...  
<section>...</section></book>
```





Hypertext Mark-up Language (HTML)



Hypertext Markup Language

- ▶ Hypertext Mark-up Language (HTML) is the language that web pages are written in (also other simple formatted documents, e.g. emails)
- ▶ It is an XML-like language for encoding hypertext documents
- ▶ It has pre-defined elements and attributes used for describing the structure of a web page and how it should be presented in a browser
- ▶ When you visit a page in a web browser, HTTP (over TCP/IP) is used to download the HTML document from the server



Hypertext

- ▶ **Hypertext** is a term coined by Ted Nelson in the 1960s
- ▶ Hypertext is non-linear
 - ▶ Books, newspaper articles etc. are written to be read from top to bottom, left to right (or right to left)
 - ▶ Hypertext is text which includes links to other text, so that the text branches and can be read via multiple paths
- ▶ In a web page, when you click on a link you issue a request to download the linked page
- ▶ Later, we see how links are encoded in HTML



Text documents

Chapter 1

In this chapter, we find out how messages are *routed* between hosts on the internet.

In Chapter 2, we will look at how communication on the internet works by using several *layers* of protocols.

Chapter 2

As we saw in Chapter 1, hosts are referred to by IP addresses. In this chapter, we will see how IP fits into a series of layers of protocols that used on the internet.



HyperText documents

Chapter 1

In this chapter, we find out how messages are *routed* between hosts on the internet.

In **Chapter 2** we will look at how communication on the internet works by using several *layers* of protocols.

LINK

LINK Chapter 2

As we saw in **Chapter 1**, hosts are referred to by IP addresses. In this chapter, we will see how IP fits into a series of layers of protocols that used on the internet.



HTML and XHTML

- ▶ HTML has been through several versions, the current standard being HTML 5.
- ▶ HTML 4.01 was almost, but not quite, XML, e.g.
 - ▶ Not every element needs a closing tag in HTML
 - ▶ Namespaces mean nothing in HTML 4.01
- ▶ XHTML 1.0 is a fully XML version of HTML
- ▶ The differences between the same webpage written in HTML 4.01 and XHTML 1.0 are small
- ▶ HTML 5 can be written in the form of HTML 4.01 or XHTML 1.0, and provides additional elements and attributes to allow easier video embedding, typesetting document sections, new form input elements, and many other changes



Structure of an HTML document

- ▶ The root element of every HTML document is
`<html>`
- ▶ The document is split into two elements
- ▶ The head, `<head>`, contains information about the document, including its name (`<title>`)
- ▶ The body, `<body>`, contains the content of the webpage/document



HTML document outline

```
<html>
```

```
  <head>
```

```
    <title>My First Webpage</title>
```

```
  </head>
```

```
  <body>
```

```
    . . .
```

```
  </body>
```

```
</html>
```



HTML document body

- ▶ Text in the body appears in a browser viewing the document
- ▶ The browser decides how to present it, and will ignore whitespace, so the HTML:

```
<body>           Hello.  
                  My name is           Samhar.  
</body>
```

- ▶ will appear in the browser as:
Hello. My name is Samhar.



Structural mark-up

- ▶ The text can be marked-up with elements stating structural detail, e.g.
- ▶ **<H1>text</H1>**: The text inside the element is a heading and so will appear big and bold on the page (H2, H3... for smaller headings)
- ▶ **<P>text</P>**: The text will appear as a paragraph distinct from the rest of the text
- ▶ HTML 5 adds more such elements, including **<SECTION>** to distinguish a section of a document



Presentation mark-up

- ▶ Other mark-up is presentational:
 - ▶ **<CENTER>text</CENTER>**: The text will be centred on the page
 - ▶ **
**: A line break will appear between the text before and after the element
 - ▶ **text** or **text**: The text is shown in bold
 - ▶ **<I>text</I>** or **<EMPHASIZE>text</EMPHASIZE>**: The text is shown in italics
- ▶ Also presentational attributes applied to multiple elements, e.g.:
align = “left” **align = “center”** **align = “right”**



Lists

- ▶ The hierarchical XML element structure allows for structured presentational mark-up
- ▶ The following appears as a bullet list:

``

` A `

` B `

``

- A
- B

- ▶ The following appears as a numbered list:

``

` A `

` B `

``

1. A
2. B



Tables

- ▶ More element nesting allows for complex structures
- ▶ Tables are split into rows, `<tr>`, then cells, `<td>`

```
<table>
```

```
  <tr>
```

```
    <td> A </td>
```

```
    <td> B </td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td> C </td>
```

```
    <td> D </td>
```

```
  </tr>
```

```
</table>
```

A	B
C	D



Images

- ▶ An image can be included in a document with an **** element
- ▶ SRC attribute gives the URL where to load the image from (often a relative URL as the image will be on the same server as the page)

``

- ▶ The URL is used by the browser to download the image using an HTTP GET request



Cascading Style Sheets

- ▶ It is encouraged practice that websites use a cascading style sheet (CSS) to define presentation rather than presentational HTML elements
- ▶ CSS is a versatile way of defining how a web page should look
- ▶ A style sheet is a separate document (or can be included in HTML document head) specifying how elements of an HTML page appear
- ▶ One style sheet may be used for a many pages to give a uniform look to a whole website
- ▶ CSS allows separation of content and appearance



Links

- ▶ A hypertext document contains links to others
- ▶ A piece of text in the document is marked up to be a link with the **<a>** element
- ▶ The linked URL is given in the **HREF** attribute
- ▶ In the following the word “here” becomes a link to the Department webpage

You can find the Department homepage

```
<a href = “http://www.inf.kcl.ac.uk/”>here</a>
```

You can find the Department homepage **here**



Fragment anchors

- ▶ Fragment anchors are often useful to link to a position in a large document, so the user goes to the right text
- ▶ An anchor names a position in a document
- ▶ Uses `<a>` with the NAME attribute

``This section of the page describes cats...

- ▶ To link to an anchor of a page at a given URL, add `#anchor` to the end of the URL

`cats`



HTML forms

- ▶ Pages can have forms to be filled in, submitted
- ▶ On submission (clicking the submit button), the form data is sent to a HTTP server
- ▶ A form is defined in a `<form>` element, with attributes:
 - ▶ **method**: HTTP method used to send the form data
 - ▶ **action**: The URL of the server to process the data

```
<form      method = "POST"  
          action = "http://ex.com:8080/process">
```



HTML form elements

- ▶ A form contains:
 - ▶ Normal HTML
 - ▶ **<input>** elements specifying boxes to be filled in
 - ▶ A specific **<input>** element adding a submit button to submit the form
- ▶ Every input has a name attribute and a type attribute
- ▶ Different types of input have different attributes and content
- ▶ Types include one-line text fields, larger text areas, drop down selection lists, buttons etc.



Example form inputs

- ▶ Input text box with space for 32 characters:
`<input type="text" name="name" size="32"/>`
- ▶ Input box with hidden entry for passwords:
`<input type="password" name="pass"/>`
- ▶ Drop-down selection list with 3 options:
`<select name = "eyecolour" size = "1">`
 `<option>Blue</option>`
 `<option>Green</option>`
 `<option>Brown</option>`
`</select>`



HTML form submission

- ▶ The submit button is added as follows:

```
<input type = "submit"/>
```

- ▶ On clicking submit button, the form data is sent by HTTP to the URL given as the form's action
- ▶ The form data is sent in an encoded form, whose MIME type is

```
application/x-www-form-urlencoded
```



Encoded form data

- ▶ The form data is encoded as follows.
- ▶ For every input on the form, a string is created of the form:


InputName=InputValue

- ▶ Spaces in the input values are converted into +
- ▶ Any other characters in the input values that are not allowed in URLs are converted into their ASCII hexadecimal form, e.g. %27 for apostrophe
- ▶ The input strings are concatenated together with &



Form data example

```
<input type = "text" name = "name" size = "32"/>  
<select name = "eyecolour" size = "1">  
  <option>Blue</option>  
  <option>Brown</option>  
</select>  
<input type = "submit"/>
```

Samhar Mahmoud	
Brown	

name=Samhar+Mahmoud&eyecolour=Brown



HTTP GET form submission

- ▶ The METHOD attribute of a <form> element specifies the HTTP method for submission: GET or POST
- ▶ With GET, the form data is added to the end of the action URL with a ?

`http://example.com:8080/process?name=Samhar+Mahmoud&eyecolour=Blue`

- ▶ The web server passes the form data to the application as parameters



HTTP GET submission example

GET /process?name=Samhar+Mahmoud&eyecolour=Brown HTTP/1.1

Host: www.example.com:8080

Accept: text/xml,application/xml,application/xhtml

Accept-Language: en-gb,en-us

Accept-Charset: ISO-8859-1,utf-8

Connection: keep-alive



HTTP POST form submission

- ▶ With POST, the URL is that given in the form's ACTION attribute, with no ? extension
- ▶ The form data, encoded in the same way as for GET, becomes the HTTP message's entity body



HTTP POST submission example

POST /process HTTP/1.1

Host: www.example.com:8080

Accept: text/xml,application/xml,application/xhtml

Accept-Language: en-gb,en-us

Accept-Charset: ISO-8859-1,utf-8

Connection: keep-alive

Content-Type: application/x-www-form-urlencoded

Content-Length: 37

name=Samhar+Mahmoud&eyecolour=Brown



GET and POST compared

- ▶ GET should be used for safe and idempotent processing
- ▶ GET can be faster as less data exchanged
- ▶ POST is better for large data and security



What we've covered

- ▶ XML
- ▶ XML Schema
- ▶ XHTML / HTML
- ▶ HTML Forms

