JAVA DAVE ENVIRONMENT

JS & DOM (Document Object Model), AJAX, JSON handling & more…

Java workshop training, August, 2017.

Dr. Kishore Biswas (Forrest/柯修)

PhD. Artificial Intelligence—Natural Language Processing.

CanWay IT Training ®.

CONTENTS

1) JS & DOM (Document Object Model)
2) AJAX briefing
3) JSON handling
4) Code example & Practice
5) Revision for the test

1) JS & DOM (Document Object Model)

What is DHTML?

- ➢DHTML is the combination of several built-in browser features in fourth generation browsers that enable a web page to be more dynamic.

- ➢DHTML is NOT a scripting language (like JavaScript or VBscript), but a browser feature- or enhancement- that makes the browser dynamic

- ➢It uses a host of different technologies - JavaScript, VBScript, the Document Object Model (DOM), layers, cascading stylesheets - to create HTML that can change even after a page has been loaded into a browser

What is DHTML?

- ➢It is considered to be made up of
  - –HTML
  - –Cascading Style Sheets (CSS)
  - –Scripting language

➢All three of these components are linked via Document Object Model (DOM)

➢DOM is the interface that allows scripting languages to access the content, style, and structure of the web documents and change them dynamically

Tools of DTHML

- HTML and XML
  - Partitions and Organizes the content
- CSS
  - Defines the Presentation of the content
- Scripting - JavaScript, JScript, VBScript
  - Adds interactivity to the page
- DOM- Document Object Model
  - Defines what and how elements are exposed for script access

Components of DHTML

DHTML requires four independent components to work: HTML, Cascading Style Sheets, Scripting and the Document Object Model. The section provides a brief description of each component.

1.HTML:
HTML defines the structure of a Web page, using such basic elements as headings, forms, tables, paragraphs and links. On December 18, 1997, HTML 4.0 attained "recommended" status at the W3C. Changes and enhancements introduced in HTML 4.0 made DHTML possible.

2. Cascading Style Sheets (CSS): Similar to a template in a word-processing document, a style sheet controls the formatting of HTML elements. Like in traditional desktop publishing, one can use style sheet to specify page margins, point sizes and leading. Cascading Style Sheets is a method to determine precedence and to resolve conflicts when multiple styles are used.

3. Scripting:

Scripting provides the mechanisms to interpret user actions and produce client-side changes to a page. For example, scripts can interpret mouse actions (such as the mouse passing over a specified area of a page through the event model) and respond to the action by using a set of predefined instructions (such as highlighting the text activated by the mouse action). Although DHTML can communicate with several scripting languages, JavaScript is the de facto standard for creating cross-browser DHTML pages.

4. Document Object Model (DOM):

The DOM outlines Web page content in a way that makes it possible for HTML elements, style sheets and scripting languages to interact with each other. The W3C defines the DOM as "a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure, and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented stage."

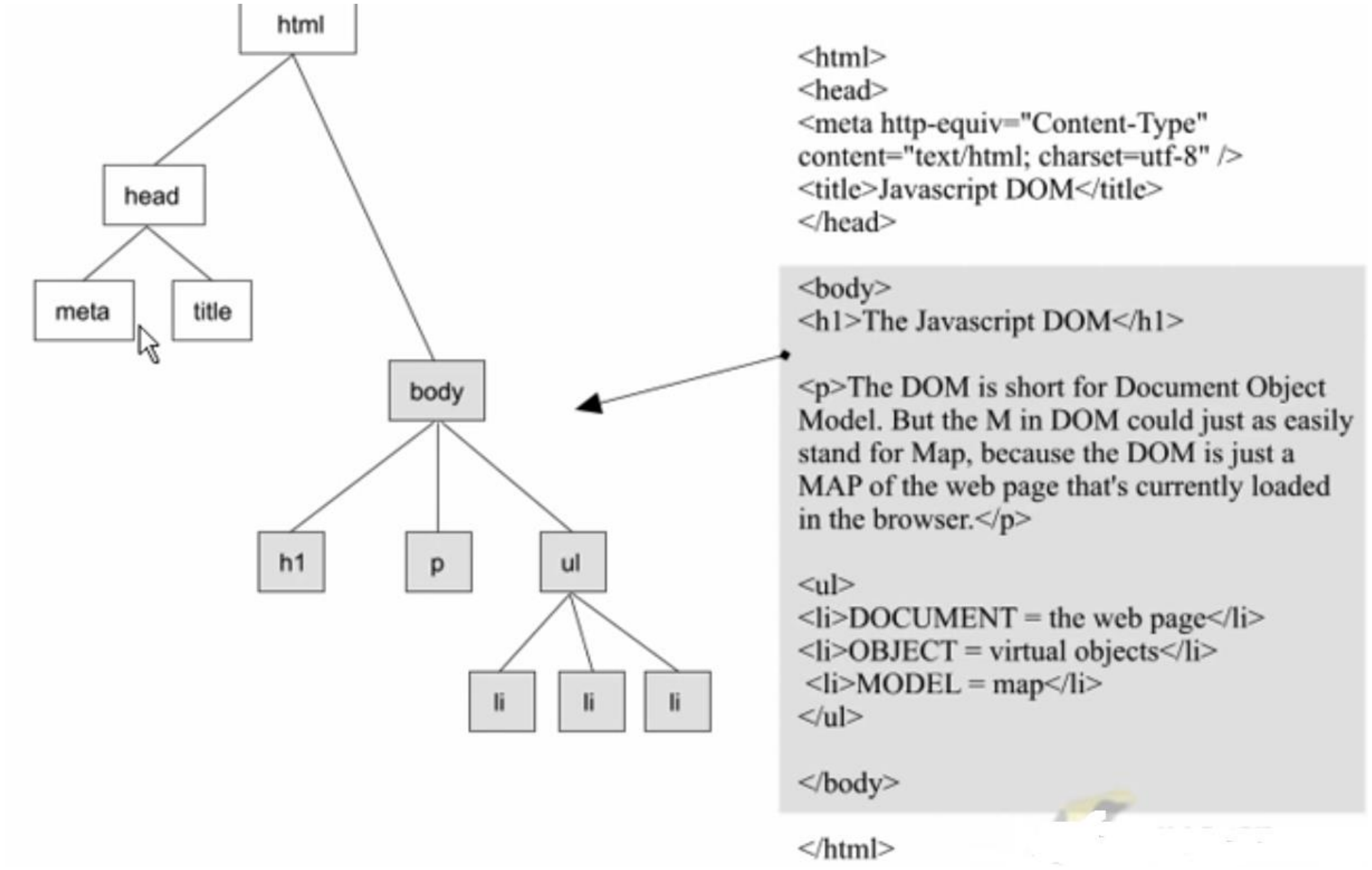
DOM INTRODUCTION

- Dynamic HTML object model
  - “The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.”
  - Great control over presentation of pages
    - Access to all elements on the page
  - Whole web page (elements, forms, frames, tables, etc.) represented in an object hierarchy
  - “Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions.”

DOM INTRODUCTION – CONT'D

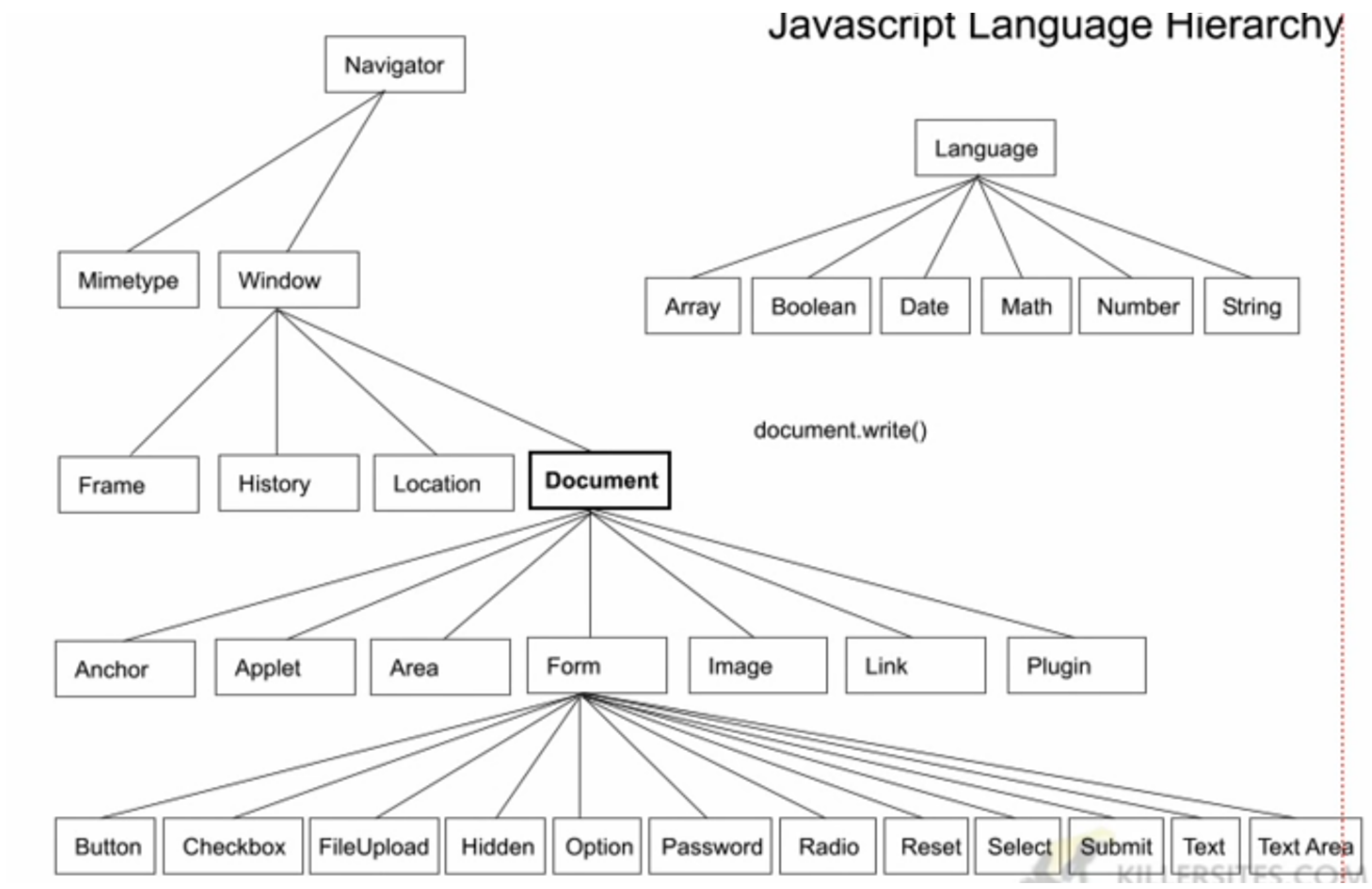
- HTML elements treated as objects
  - Attributes of these elements treated as properties of those objects
    - Objects identified with an **`ID`** attribute can be scripted with languages like JavaScript and VBScript.
    - Elements on the page can be supported by scripting that can interact with user events and change the page content dynamically.
    - The DOM dictates how the written scripting language controls the elements on the screen, such as graphics and text.

DOM TREE



```
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=utf-8" />
<title>Javascript DOM</title>
</head>

<body>
<h1>The Javascript DOM</h1>

<p>The DOM is short for Document Object
Model. But the M in DOM could just as easily
stand for Map, because the DOM is just a
MAP of the web page that's currently loaded
in the browser.</p>

<ul>
<li>DOCUMENT = the web page</li>
<li>OBJECT = virtual objects</li>
 <li>MODEL = map</li>
</ul>

</body>

</html>
```

JS HIERARCHY



DOM-Document Object Model …

- The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of the document

- The DOM details the characteristic properties of each element of a Web page, thereby detailing how we might manipulate these components and, in turn, manipulate the page

DOM-Document Object Model…

- Document Object Model is not a "part" of Scripting languages. The DOM stands alone, able to be interfaced with any programming language designed to do so

- The W3C DOM is the recommended standard to be exposed by each browser

- Microsoft Internet Explorer and Netscape do not share the same DOM.

DOM-Document Object Model…

- Both (IE and Netscape) DOMs break down Web pages into roughly four types of components
  - –Objects, Properties, Events and Methods

- **Objects :**
  - –Container which reflects a particular element of a page
  - –objects "contain" the properties and methods which apply to that element

DOM-Document Object Model…

- **Properties:**

  –Characteristics of an object

  –Example: the ‘document’ object possesses a ‘bgColor’ property which reflects the background color of the page.

  –Using a programming language (e.g. JavaScript) you may, via this property, read or modify the background color of a page

DOM-Document Object Model…

- ➢**Methods:**
  - –A method typically executes an action which acts upon the object by which it is owned

- ➢**Events:**
  - –Used to trap actions related to its owning object
  - –Typically, these actions are caused by the user

DOM

- DOM = Document Object Model

  Defines a hierarchical model of the document structure through which all document elements may be accessed

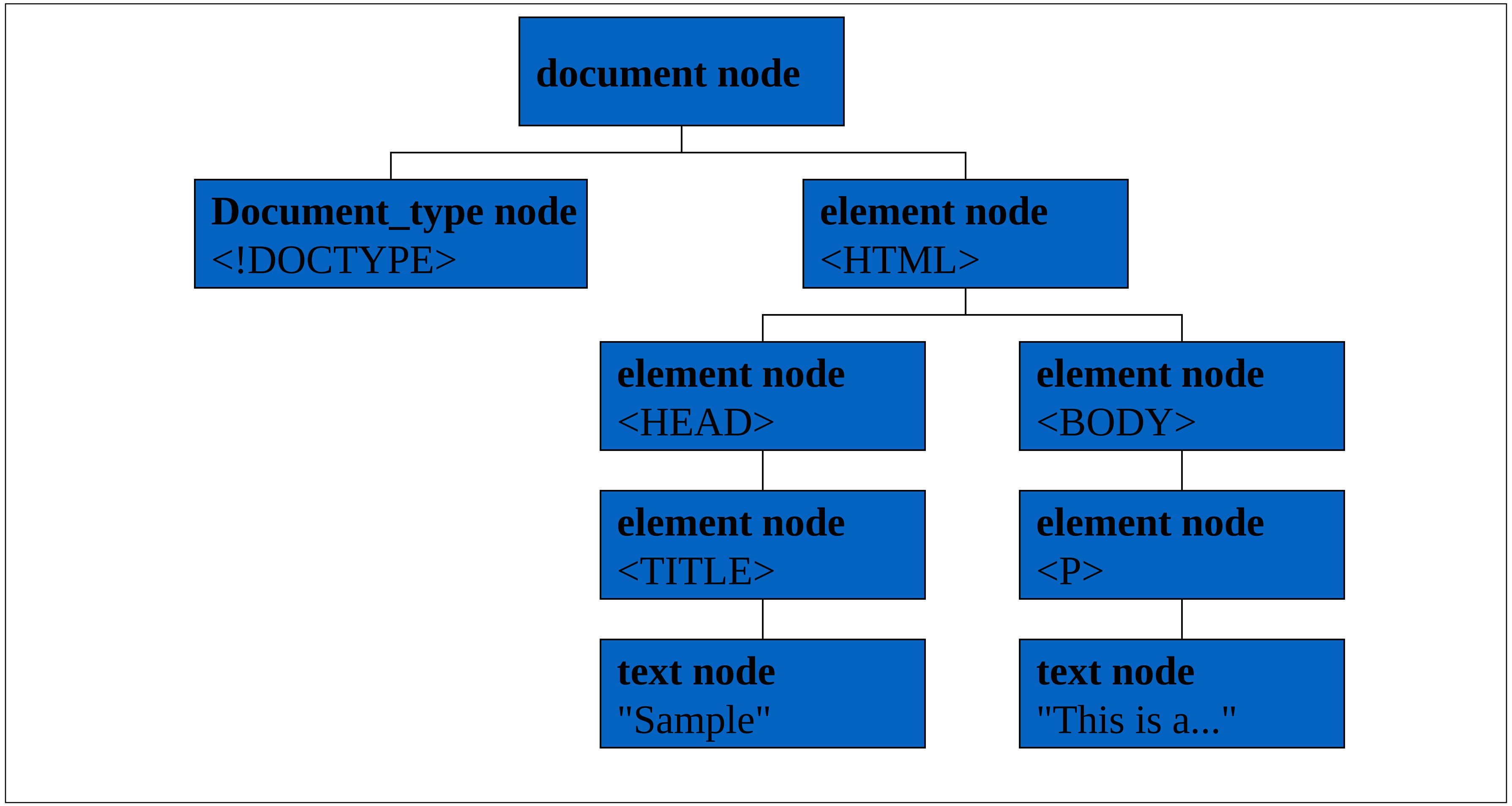
- Nodes

  The W3C DOM defines element of a document is a *node* of a particular type

- Node Types

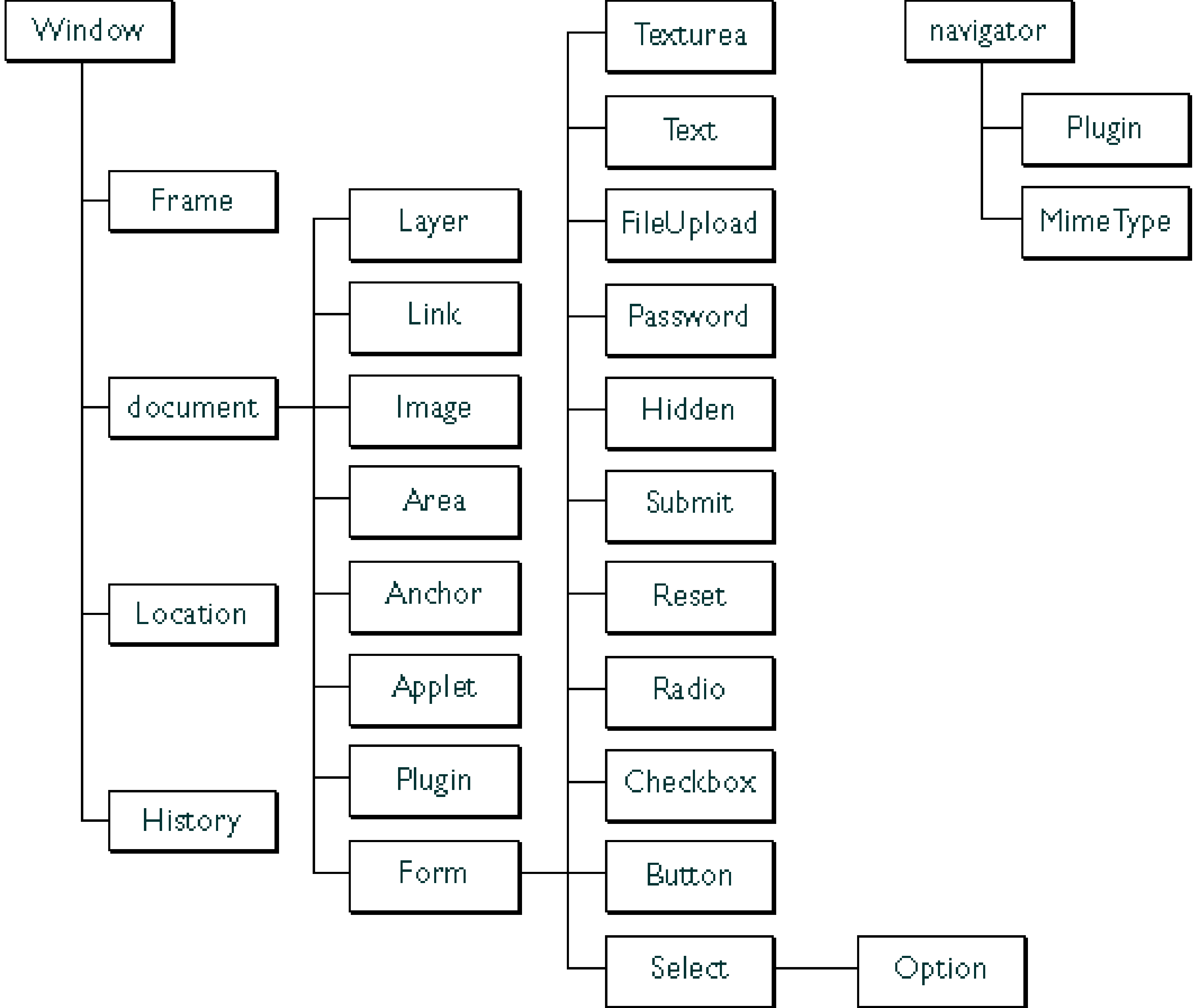
  Common types are: document node, element node, text node, attribute node, comment node, document-type node

DOM Example


document node
Document_type node <!DOCTYPE>
element node <HTML>
element node <HEAD>
element node <BODY>
element node <TITLE>
element node <P>
text node "Sample"
text node "This is a..."

Netscape DOM

- DOM "begins" at the window object;
- Other objects are below the window object in the hierarchy
- Exception is the navigator object, (whose properties provide information about the browser version,) which is a peer object of window rather than a child

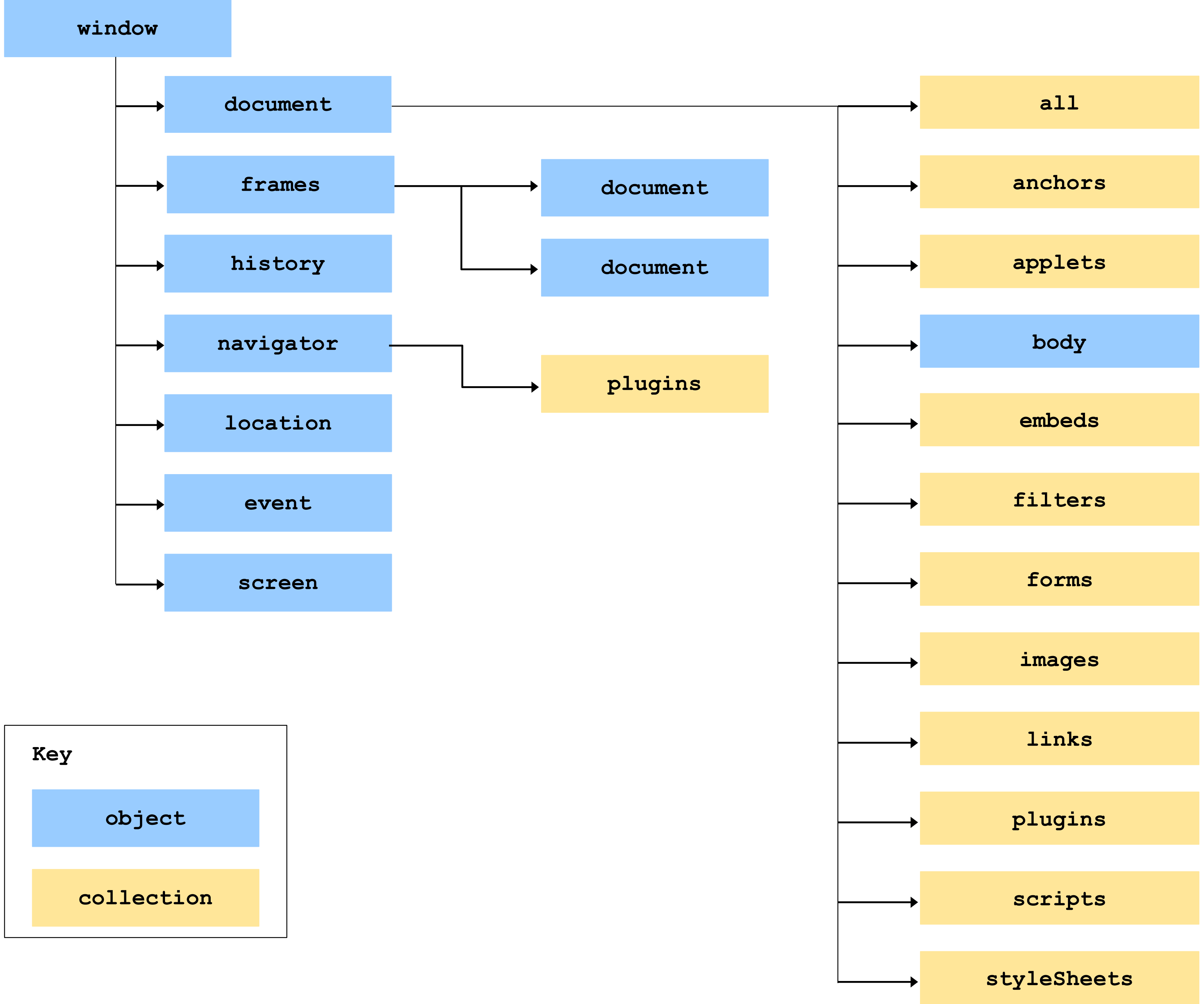

Window
Frame
document
Location
History
Layer
Link
Image
Area
Anchor
Applet
Plugin
Form
Texturea
Text
FileUpload
Password
Hidden
Submit
Reset
Radio
Checkbox
Button
Select
Option
navigator
Plugin
MimeType

Microsoft DOM

- Microsoft indexes an additional topic called as collections. A "collection," is an array-based object

- The observable difference between the two is the syntax. Netscape supports a treelike hierarchical syntax

- On the other hand, Internet Explorer exposes all HTML objects as a flat collection and lets you modify the style object


window
document
frames
history
navigator
location
event
screen
document
document
plugins
all
anchors
applets
body
embeds
filters
forms
images
links
plugins
scripts
styleSheets
Key
object
collection

Collections `all` and `children`

- *Collections* are basically arrays of related objects on a page
- **`all`**
  - Collection of all the HTML elements in a document in the order in which they appear
- **`length`** property
  - Specifies the number of elements in the collection
- **`tagName`** property of an element
  - Determines the name of the element
- Every element has its own **`all`** collection, consisting of all the elements contained within that element

ADVANTAGES & DISADVANTAGES

- Advantage
  - Robust API for DOM tree
  - Relatively simple to modify data structure and extract data
- Disadvantage
- Store entire document in memory
- As DOM was written for any language method naming not follow the standard VB programming conventions

Conclusion - Overview

- ➢DHTML – HTML, CSS, Scripting and DOM

- ➢DOM - details the characteristic properties of each element of a Web page
- ➢DHTML can make your browser dynamic and interactive

- ➢Content and design can be separated using Style sheets & uniformity of the site can be maintained using them

- ➢Validation of input’s given by the user can be done at the client side, without connection to the server

- ➢Drop down menus can be used to put a lot of information on the site

2) AJAX BRIEFING

WHAT IS AJAX (AJAJ)...

- AJAX, is a web development technique for creating interactive web applications.
- If you know JavaScript, HTML, CSS, and XML, then you need to spend just one hour to start with AJAX.

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.

What is AJAX (AJAJ)…

- Conventional web applications transmit information to and from the server using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.

WHAT IS AJAX (AJAJ)...

- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

Rich Internet Application Technology

- AJAX is the most viable Rich Internet Application (RIA) technology so far. It is getting tremendous industry momentum and several tool kit and frameworks are emerging. But at the same time, AJAX has browser incompatibility and it is supported by JavaScript, which is hard to maintain and debug.

AJAX IS BASED ON OPEN STANDARDS

AJAX is based on the following open standards:

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.

AJAX TECHNOLOGIES

AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages.

- **JavaScript**

    Loosely typed scripting language.
    JavaScript function is called when an event occurs in a page.

    Glue for the whole AJAX operation.

- **DOM**

    API for accessing and manipulating structured documents.

    Represents the structure of XML and HTML documents.

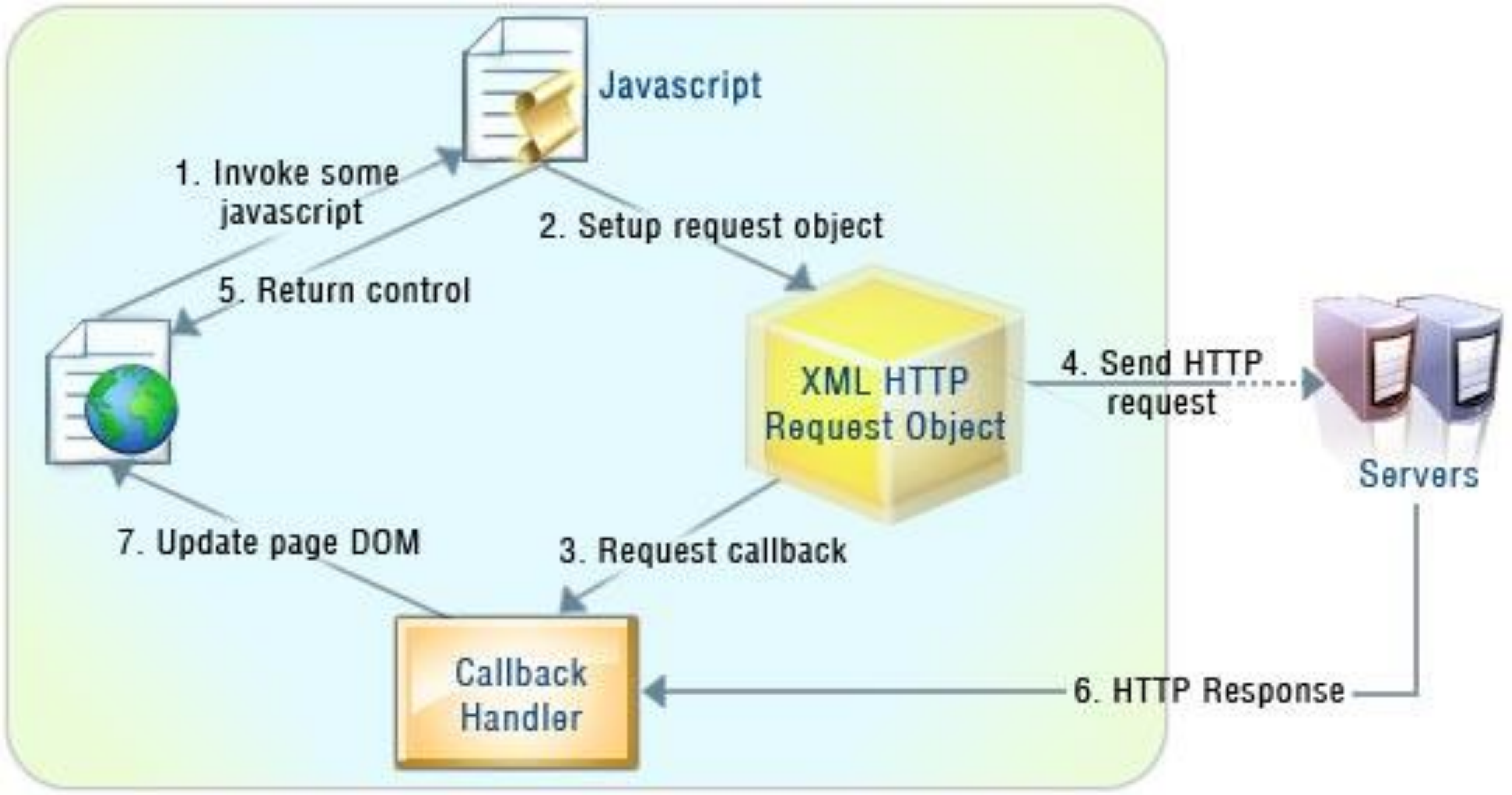
- **CSS**

    Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript.

- **XMLHttpRequest**

    JavaScript object that performs asynchronous interaction with the server.

AJAX TECH OVERVIEW



AJAX BASED TECH EXAMPLES

- **Google Maps**

  A user can drag an entire map by using the mouse, rather than clicking on a button.

  http://maps.google.com/

- **Google Suggest**

  As you type, Google will offer suggestions. Use the arrow keys to navigate the results.

  http://www.google.com/webhp?complete=1&hl=en

- **Gmail**

  Gmail is a webmail, built on the idea that email can be more intuitive, efficient and useful.

  http://gmail.com/

- **Yahoo Maps (new)**

  Now it's even easier and more fun to get where you're going!

  http://maps.yahoo.com/

STEPS OF AJAX OPERATION

- 1) A client event occurs.
- 2) An XMLHttpRequest object is created.
- 3) The XMLHttpRequest object is configured.
- 4) The XMLHttpRequest object makes an asynchronous request to the Webserver.
- 5) The Webserver returns the result containing XML document.
- 6) The XMLHttpRequest object calls the callback() function and processes the result.
- 7) The HTML DOM is updated.

A Client Event Occurs

- A JavaScript function is called as the result of an event.

- Example: validateUserId() JavaScript function is mapped as an event handler to an onkeyup event on input form field whose id is set to "userid"

- <input type="text" size="20" id="userid" name="id" onkeyup="validateUserId();">.

The XMLHttpRequest Object is Created

```
• var ajaxRequest;  // The variable that makes Ajax possible!
• function ajaxFunction(){
•   try{
•           // Opera 8.0+, Firefox, Safari
•     ajaxRequest = new XMLHttpRequest();
•   }catch (e){
•       // Internet Explorer Browsers
•     try{
•       ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
•     }catch (e) {
•           try{
•         ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
•       }catch (e){
•             // Something went wrong
•         alert("Your browser broke!");
•         return false;
•       }
•     }
•   }
• }
```

The XMLHttpRequest Object is Configured

- function validateUserId() {
-     ajaxFunction();
-
-     // Here processRequest() is the callback function.
-     ajaxRequest.onreadystatechange = processRequest;
-
-     if (!target) target = document.getElementById("userid");
-     var url = "validate?id=" + escape(target.value);
-
-     ajaxRequest.open("GET", url, true);
-     ajaxRequest.send(null);
- }

Making Asynchronous Request to the Webserver

- function validateUserId() {
-   ajaxFunction();
-
-   // Here processRequest() is the callback function.
-   ajaxRequest.onreadystatechange = processRequest;
-
-   if (!target) target = document.getElementById("userid");
-   var url = "validate?id=" + escape(target.value);
-
-   ajaxRequest.open("GET", url, true);
-   ajaxRequest.send(null);
- }

// Assume you enter Forrest in the userid box, then in the above request, the URL is set to "validate?id=Forrest".

Webserver returns the result containing XML document

```
• public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException
• {
•   String targetId = request.getParameter("id");
•
•   if ((targetId != null) && !accounts.containsKey(targetId.trim()))
•   {
•     response.setContentType("text/xml");
•     response.setHeader("Cache-Control", "no-cache");
•     response.getWriter().write("true");
•   }
•   else
•   {
•     response.setContentType("text/xml");
•     response.setHeader("Cache-Control", "no-cache");
•     response.getWriter().write("false");
•   }
• }
```

implementing server-side script in any language should be as follows:

1) Get a request from the client.
2) Parse the input from the client.
3) Do required processing.
4) Send the output to the client.

Callback function processRequest() is called

- The XMLHttpRequest object was configured to call the processRequest() function when there is a state change to the readyState of the XMLHttpRequest object. Now this function will receive the result from the server and will do the required processing. As in the following example, it sets a variable message on true or false based on the returned value from the Webserver.

```
function processRequest() {
  if (req.readyState == 4) {
    if (req.status == 200) {
      var message = ...;
...
}
```

The HTML DOM is Updated

- This is the final step and in this step, your HTML page will be updated. It happens in the following way:

- JavaScript gets a reference to any element in a page using DOM API.
- The recommended way to gain a reference to an element is to call.
- document.getElementById("userIdMessage"),
- // where "userIdMessage" is the ID attribute
- // of an element appearing in the HTML document
- JavaScript may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify the child elements.

XMLHTTPREQUEST

- The XMLHttpRequest object is the key to AJAX. It has been available ever since Internet Explorer 5.5 was released in July 2000, but was not fully discovered until AJAX and Web 2.0 in 2005 became popular.
- XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.
- The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.
- You already have seen a couple of examples on how to create an XMLHttpRequest object.

XMLHTTPREQUEST METHODS

- **abort()** : Cancels the current request.
- **getAllResponseHeaders()** : Returns the complete set of HTTP headers as a string.
- **getResponseHeader(headerName)** : Returns the value of the specified HTTP header.
- **open(method, URL)** : open(method, URL, async)
- **open(method, URL, async, userName)** :
- **open(method, URL, async, userName, password)** : Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods, such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.

3) JSON HANDLING

JSON as an XML Alternative

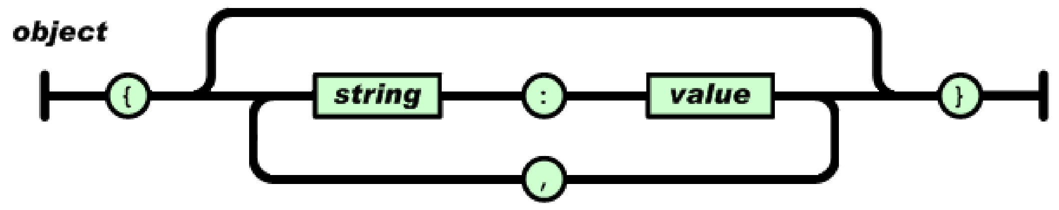
- JSON is a light-weight alternative to XML for data-interchange
- JSON = JavaScript Object Notation
  - It's really language independent
  - most programming languages can easily read it and instantiate objects or some other data structure
- Started gaining tracking ~2006 and now widely used
- http://json.org/ has more information

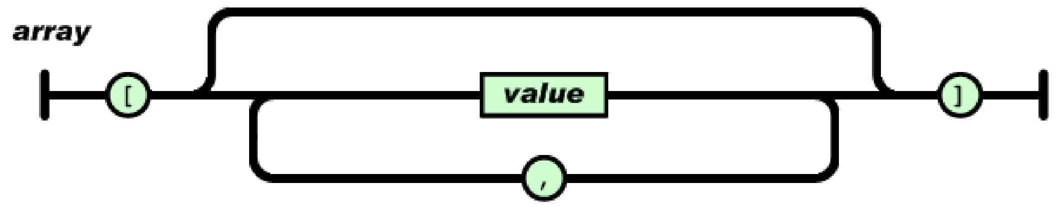
EXAMPLE

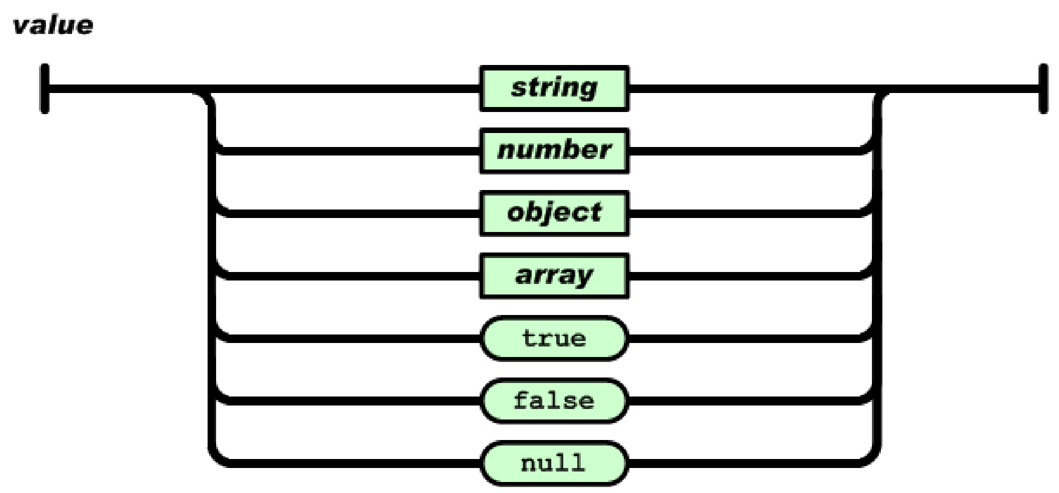
```
{"firstName": "John",
 "lastName" : "Smith",
 "age"        : 25,
 "address"   :
    {"streetAdr” : "21 2nd Street",
      "city"          : "New York",
      "state"        : "NY",
      ”zip"           : "10021"},
 "phoneNumber":
    [{"type"  : "home",
      "number": "212 555-1234"},
     {"type"  : "fax",
      "number” : "646 555-4567"}]
 }
```

- This is a JSON object with five key-value pairs
- Objects are wrapped by curly braces
- There are no object IDs
- Keys are strings
- Values are numbers, strings, objects or arrays
- Ararys are wrapped by square brackets

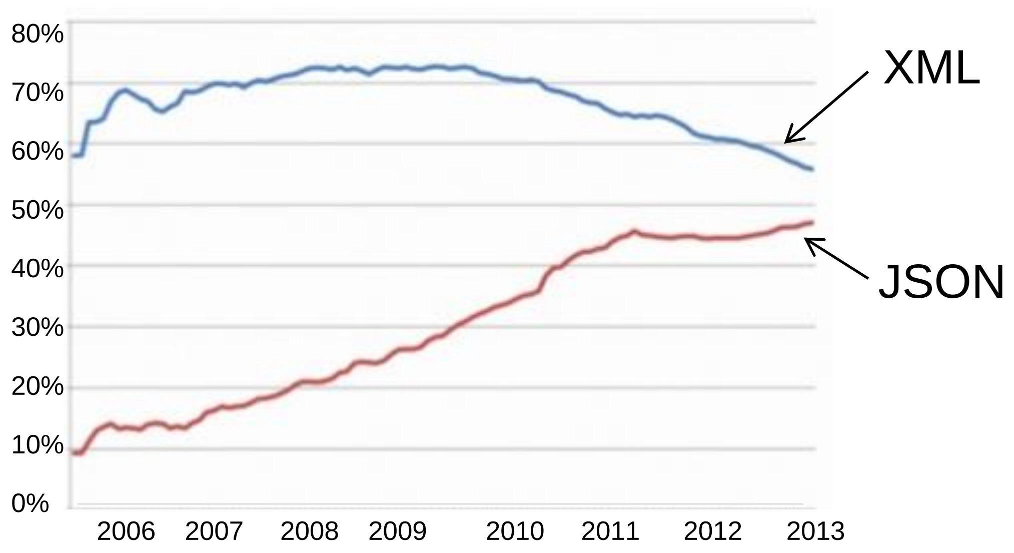
THE BNF IS SIMPLE



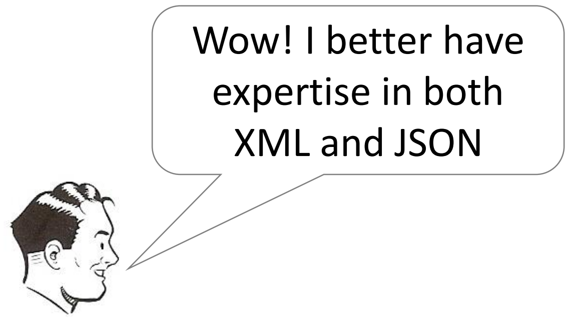




Trends in XML and JSON Usage



Based on directory of 11,000 web APIs listed at Programmable Web, December 2013



Example of XML-formatted data

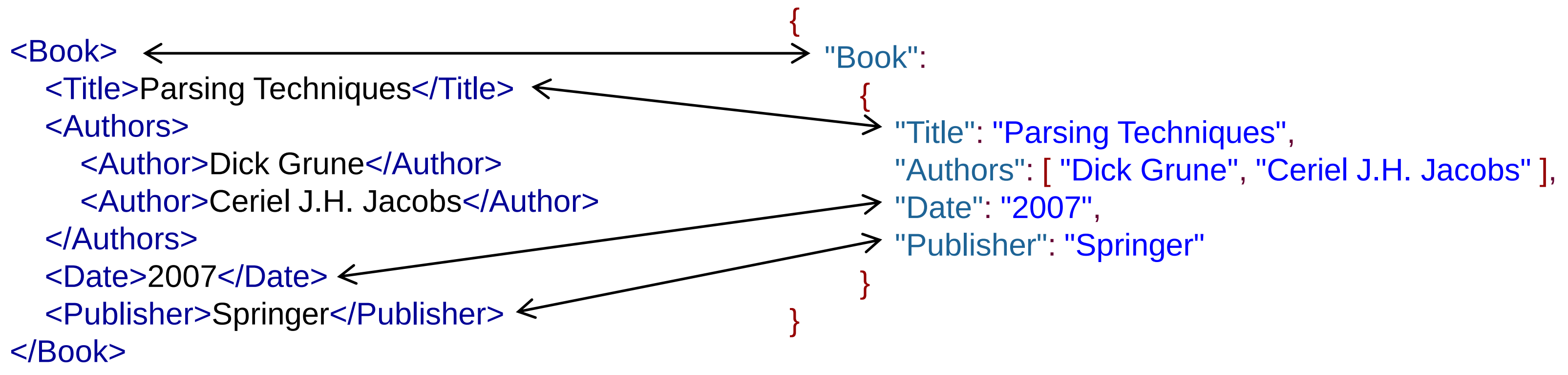
The below XML document contains data about a book: its title, authors, date of publication, and publisher.

```
<Book>
  <Title>Parsing Techniques</Title>
  <Authors>
    <Author>Dick Grune</Author>
    <Author>Ceriel J.H. Jacobs</Author>
  </Authors>
  <Date>2007</Date>
  <Publisher>Springer</Publisher>
</Book>
```

Same Data, JSON-formatted

```
{
    "Book":
        {
            "Title": "Parsing Techniques",
            "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ],
            "Date": "2007",
            "Publisher": "Springer"
        }
}
```

XML and JSON, side-by-side



```xml
<Book>
   <Title>Parsing Techniques</Title>
   <Authors>
      <Author>Dick Grune</Author>
      <Author>Ceriel J.H. Jacobs</Author>
   </Authors>
   <Date>2007</Date>
   <Publisher>Springer</Publisher>
</Book>
```

```json
{
  "Book":
    {
      "Title": "Parsing Techniques",
      "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ],
      "Date": "2007",
      "Publisher": "Springer"
    }
}
```

Creating Lists in XML and JSON

```
<Book>
   <Title>Parsing Techniques</Title>
   <Authors>
      <Author>Dick Grune</Author>
      <Author>Ceriel J.H. Jacobs</Author>
   </Authors>
   <Date>2007</Date>
   <Publisher>Springer</Publisher>
</Book>
```

$\longleftrightarrow$

```
{
   "Book":
      {
         "Title": "Parsing Techniques",
         "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ],
         "Date": "2007",
         "Publisher": "Springer"
      }
}
```

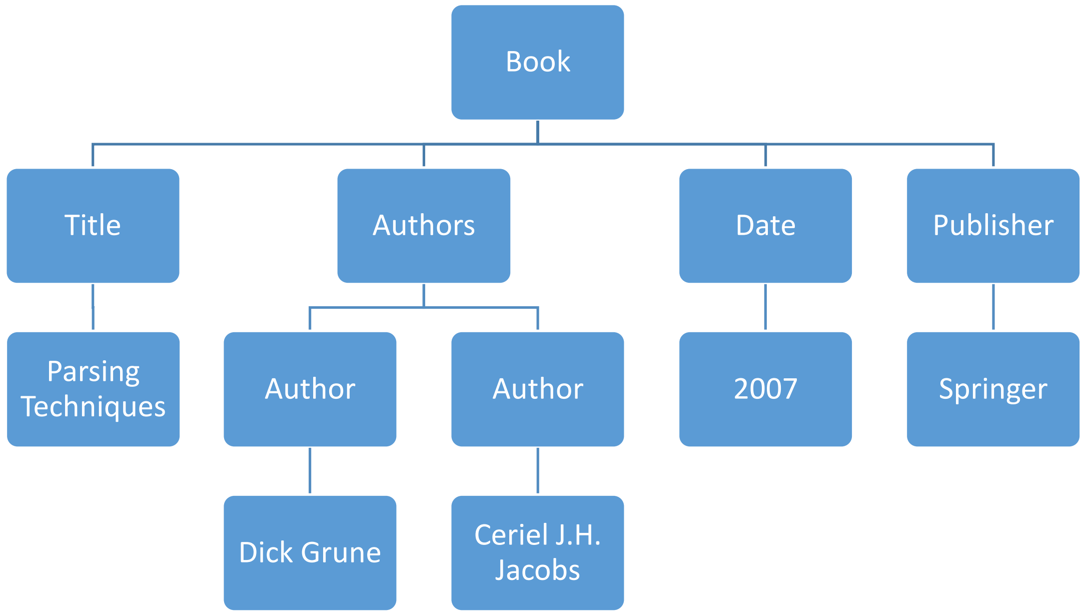
XML IS A META-LANGUAGE

- XML is a language that you use to create other languages.
- For example, on the previous slides we saw how to use XML to create a Book language, consisting of <Book>, <Title>, <Author>, and so forth.

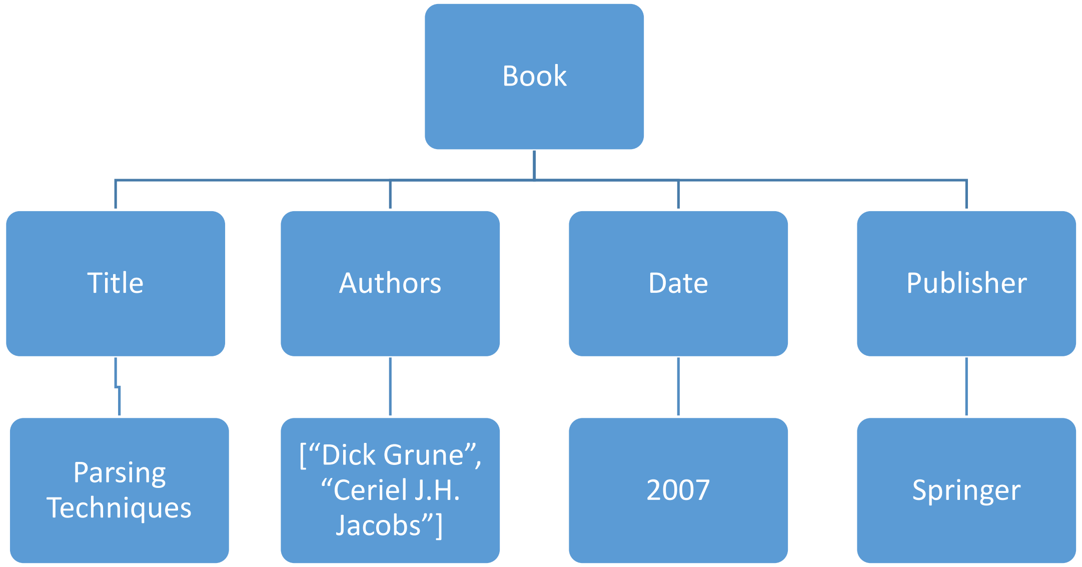
JSON is a Meta-Language

- JSON is also a language that you use to create other languages.
- For example, on the previous slides we saw how to use JSON to create a Book language, consisting of "Book", "Title", "Author", and so forth.

AN XML DOCUMENT IS A TREE



A JSON OBJECT IS A TREE



Trees are Well-Studied

- The tree data structure has been well-studied by computer scientists and mathematicians.
- There are many well-known algorithms for processing and traversing trees.
- Both XML and JSON are able to leverage this.

XML Schema for Book

```
<xs:element name="Book">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Title" type="xs:string" />
            <xs:element name="Authors">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Author" type="xs:string" maxOccurs="5"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Date" type="xs:gYear" />
            <xs:element name="Publisher" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Springer" />
                        <xs:enumeration value="MIT Press" />
                        <xs:enumeration value="Harvard Press" />
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Equivalent JSON Schema

```
{
    "$schema": http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "Book": {
            "type": "object",
            "properties": {
                "Title": {"type": "string"},
                "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
                "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
                "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
            },
            "required": ["Title", "Authors", "Date"],
            "additionalProperties": false
        }
    },
    "required": ["Book"],
    "additionalProperties": false
}
```

TITLE WITH STRING TYPE

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "Book": {
            "type": "object",
            "properties": {
                "Title": {"type": "string"},
                "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
                "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
                "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
            },
            "required": ["Title", "Authors", "Date"],
            "additionalProperties": false
        }
    },
    "required": ["Book"],
    "additionalProperties": false
}
```

```
<xs:element name="Title" type="xs:string" />
```

AUTHORS LIST

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "Book": {
            "type": "object",
            "properties": {
                "Title": {"type": "string"},
                "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
                "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
                "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
            },
            "required": ["Title", "Authors", "Date"],
            "additionalProperties": false
        }
    },
    "required": ["Book"],
    "additionalProperties": false
}
```

```
<xs:element name="Authors">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Author" type="xs:string" maxOccurs="5"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Date with Year Type

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "Book": {
            "type": "object",
            "properties": {
                "Title": {"type": "string"},
                "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
                "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
                "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
            },
            "required": ["Title", "Authors", "Date"],
            "additionalProperties": false
        }
    },
    "required": ["Book"],
    "additionalProperties": false
}
```

```
<xs:element name="Date" type="xs:gYear" />
```

Publisher with Enumeration

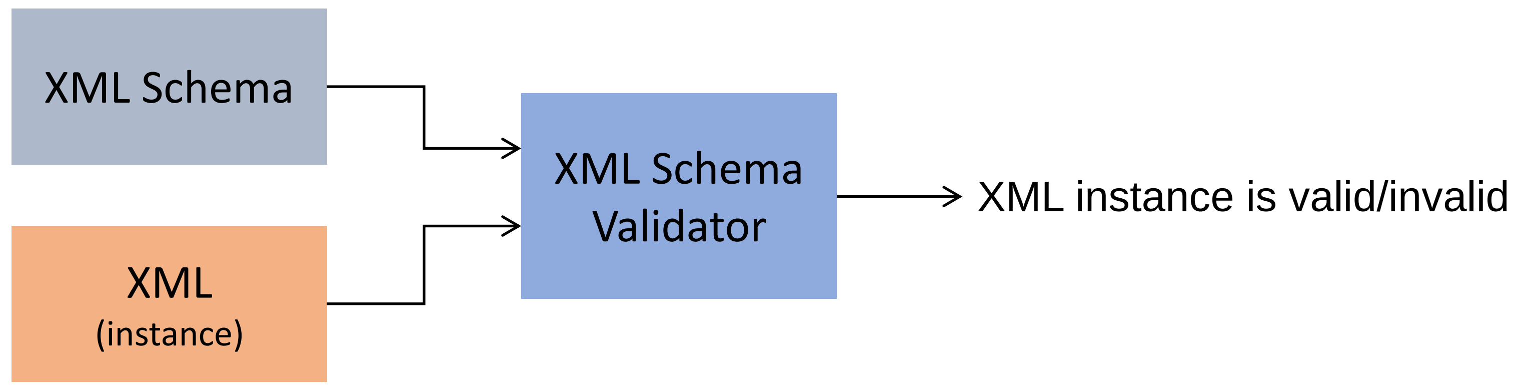
```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "Book": {
            "type": "object",
            "properties": {
                "Title": {"type": "string"},
                "Authors": {"type": "array", "minItems": 1, "maxItems": 5, "items": { "type": "string" }},
                "Date": {"type": "string", "pattern": "^[0-9]{4}$"},
                "Publisher": {"type": "string", "enum": ["Springer", "MIT Press", "Harvard Press"]}
            },
            "required": ["Title", "Authors", "Date"],
            "additionalProperties": false
        }
    },
    "required": ["Book"],
    "additionalProperties": false
}
```

```
<xs:element name="Publisher" minOccurs="0">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:enumeration value="Springer" />
         <xs:enumeration value="MIT Press" />
         <xs:enumeration value="Harvard Press" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
```

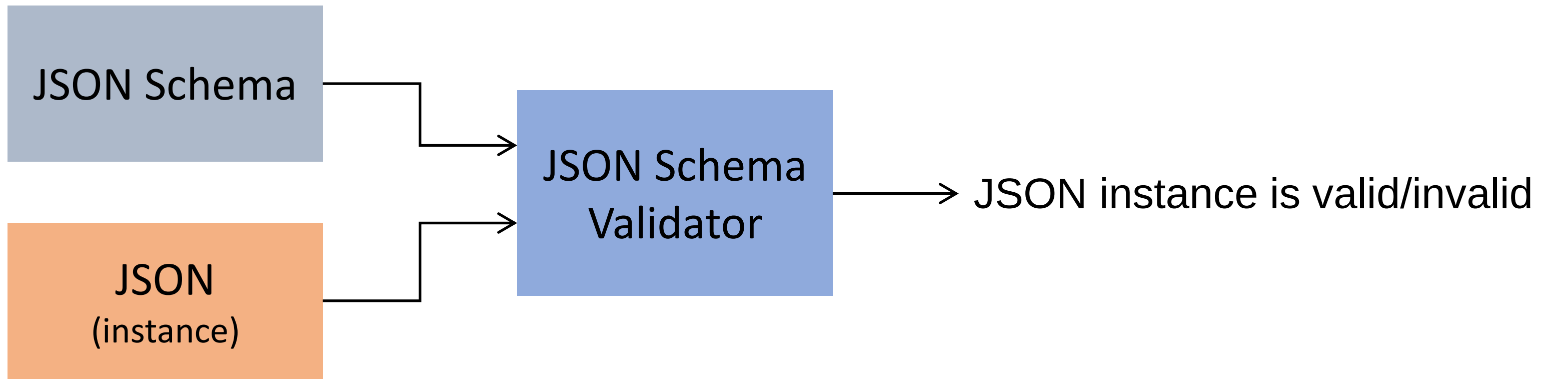
Bootstrap the Schema Language

- An XML Schema is written in XML.
- A JSON Schema is written in JSON.

Validate XML Docs Against XML Schema

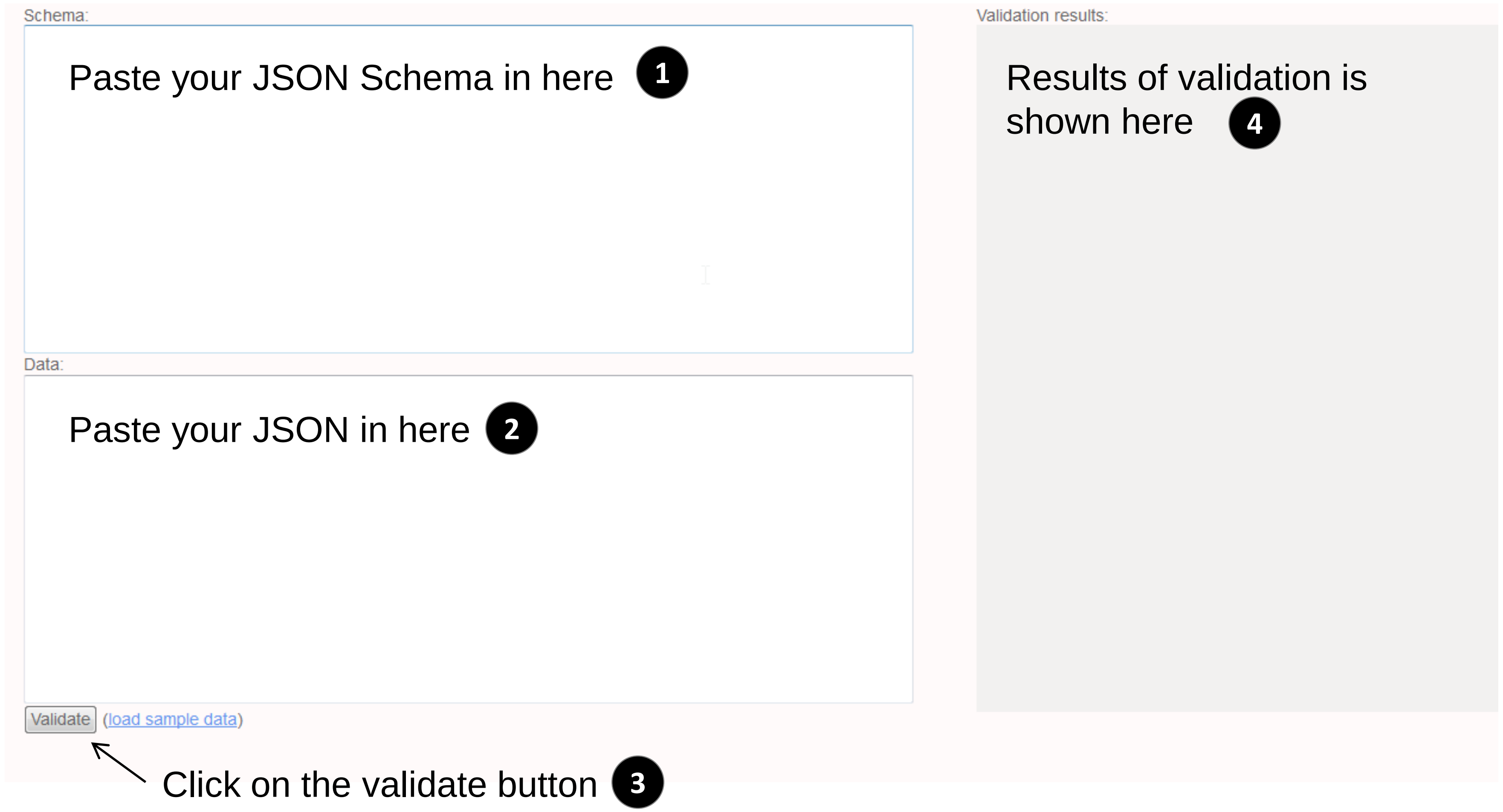


Validate JSON docs against JSON Schema



ONLINE JSON SCHEMA VALIDATOR

http://json-schema-validator.herokuapp.com/index.jsp



Paste your JSON Schema in here 1

Paste your JSON in here 2

Click on the validate button 3

Results of validation is shown here 4

Command-Line JSON Schema Validator

Want to validate a JSON instance against a JSON Schema from the command line? Download:

json-schema-validator-2.2.5-lib.jar

from:

https://bintray.com/fge/maven/json-schema-validator/view

and then, from a command line, type:

```
java  -jar  json-schema-validator-2.2.5-lib.jar  schema.json   instance.json
```

USE A JSON SCHEMA VALIDATOR FROM A JAVA PROGRAM

Want to validate a JSON instance against a JSON Schema from a Java program? Download:

json-schema-validator-2.2.5-lib.jar

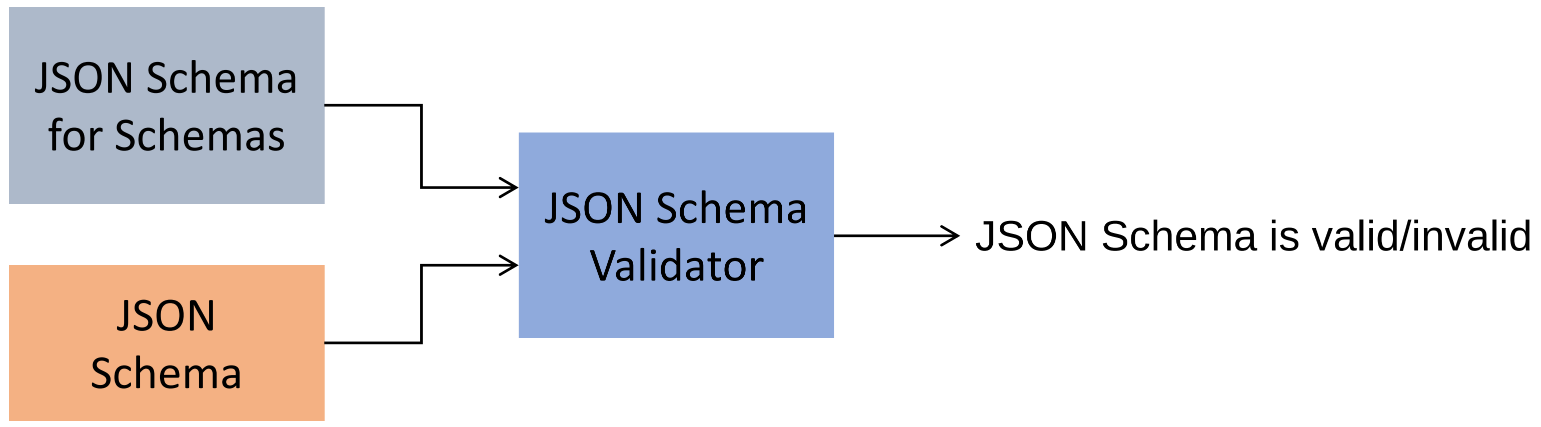
from:

https://bintray.com/fge/maven/json-schema-validator/view

and then read the documentation on the Java API.

Schema-for-Schemas

At the following URL is a JSON Schema. It is a schema for validating JSON Schemas … it is a schema for schemas!
http://json-schema.org/draft-04/schema#



Status of JSON Schema Specifications

- JSON Schema is being developed under the auspicies of the IETF standards organization.
- The JSON Schema specification is currently at draft #4.
- Work is proceeding on draft #5.
- Draft #5 will contain a few new things, but will be mostly the same as draft #4 (draft #5 is backward compatible with draft #4).
- Here are the draft #5 changes: https://groups.google.com/forum/#!topic/json-schema/FSJmct8crXk

Example of XML-to-JSON

The online freeformatter.com tool converts this XML:

```
<Book id="MCD">
     <Title>Modern Compiler Design</Title>
     <Author>Dick Grune</Author>
     <Publisher>Springer</Publisher>
</Book>
```

to this JSON:

```
{
   "@id": "MCD",
   "Title": "Modern Compiler Design",
   "Author": "Dick Grune",
   "Publisher": "Springer"
}
```

I like that it encodes XML attributes by prefixing the attribute name with the @ symbol.

Auto-converting XML to JSON: a bad idea?

Should you devise a way to auto-convert XML to JSON? In the below message the person says:

> Based on our real-world experience, it is best to create the JSON design from scratch.
> Do not auto-generate it from XML.

Hi all,

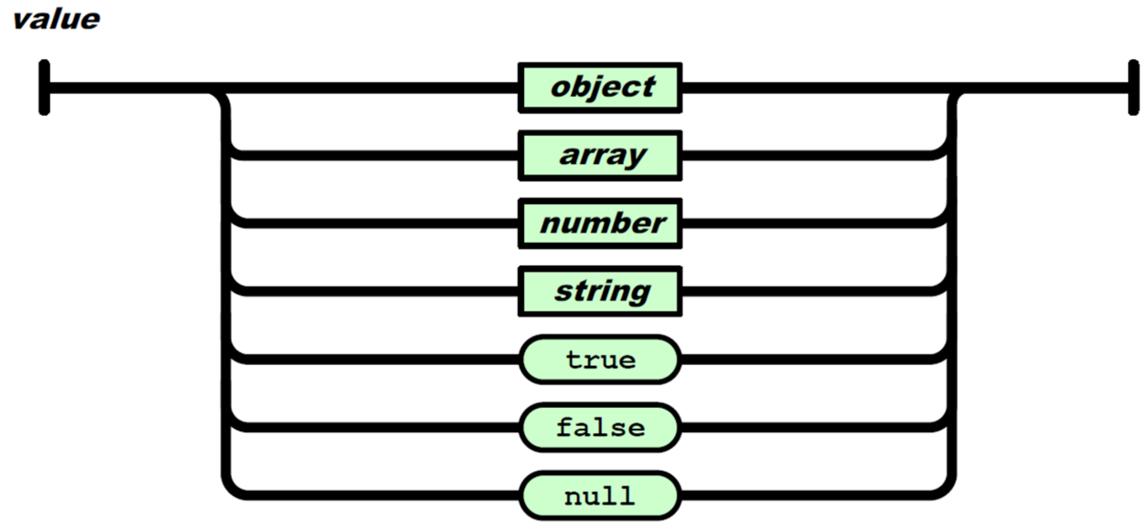
To throw in a view from a long-time XML user:
IPTC - www.iptc.org - builds XML-based news exchange formats for 17 years
now and was also challenged to do the same in JSON. After a long discussion
we refrained from automatically converting an existing XML data model to
JSON:
- currently no shared/common way to deal with namespaces in JSON
- designs like inline elements don't exist in JSON
- the element/attribute model has no corresponding design in JSON
- and a basic requirement of JSON users is: no complex data model, please!

Therefore we created
- a simplified data model for the news exchange in JSON - www.newsinjson.org
- compared to the richer but also more complex XML format www.newsml-g2.org
- a highly corresponding JSON model to an initial XML model for the rights
expression language ODRL as this is a set of data which cannot be
simplified: http://www.w3.org/community/odrl/work/json/ vs
http://www.w3.org/community/odrl/work/2-0-xml-encoding-constraint-draft-changes/

Both approaches were welcome and are used - and we learned: **an XML-to-JSON
tool only is of limited help**.

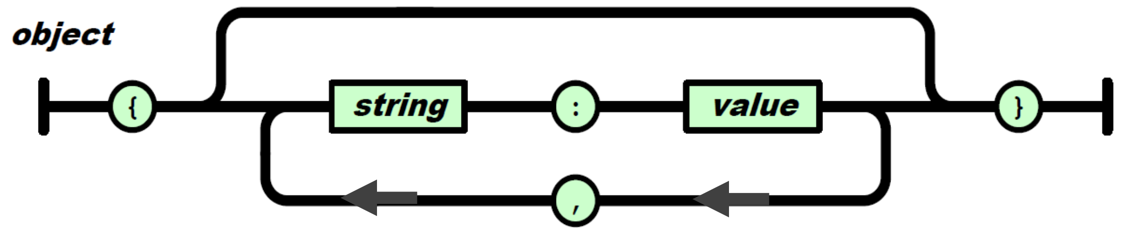
JSON Value

A JSON instance contains a single JSON value. A JSON value may be either an *object*, *array*, *number*, *string*, true, false, or null:



JSON OBJECT

A JSON object is zero or more *string-colon-value* pairs, separated by comma and wrapped within curly braces:

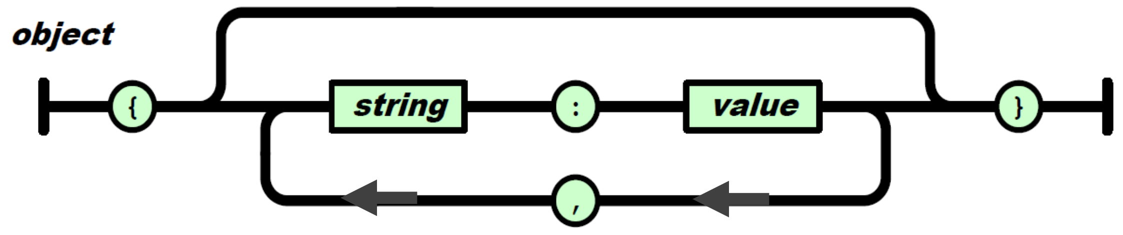


Example of a JSON object:

```
{ "name": "John Doe",
  "age": 30,
  "married": true }
```

Empty Object

A JSON object may be empty.



This is a JSON object: { }

NO DUPLICATE KEYS

- You should consider JSON objects as containing key/value pairs.
- Just as in a database the primary keys must be unique, so too in a JSON object the keys must be unique.
- This JSON object has duplicate keys:

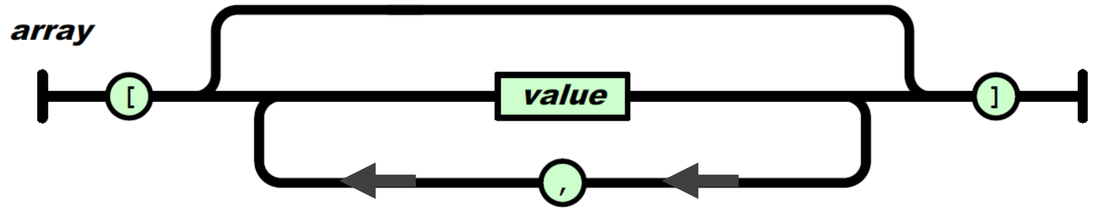
```
{ "Title": "A story by Mark Twain",
  "Title": "The Adventures of Huckleberry Finn" }
```

JSON PARSERS HAVE UNPREDICTABLE BEHAVIOR ON JSON OBJECTS WITH DUPLICATE KEYS

A JSON object whose names are all unique is interoperable in the sense that all software implementations receiving that object will agree on the name-value mappings. When the names within an object are not unique, the behavior of software that receives such an object is unpredictable. Many implementations report the last name/value pair only. Other implementations report an error or fail to parse the object and some implementations report all of the name/value pairs, including duplicates.

JSON ARRAY

A JSON array is used to express a list of values. A JSON array contains zero or more values, separated by comma and wrapped within square brackets:



Example of a JSON array

```
{ "name": "John Doe",
  "age": 30,
  "married": true,
  "siblings": ["John", "Mary", "Pat"] }
```

EMPTY ARRAY VS. ARRAY WITH A NULL VALUE

[ ]

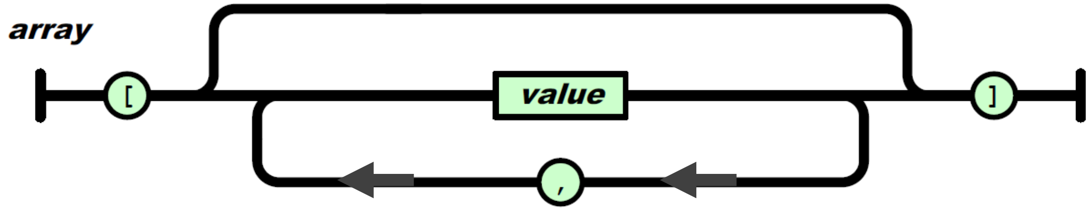
Array with no items in it.

[null]

Array with one item in it.

Array of Objects

Each item in an array may be any of the seven JSON values.

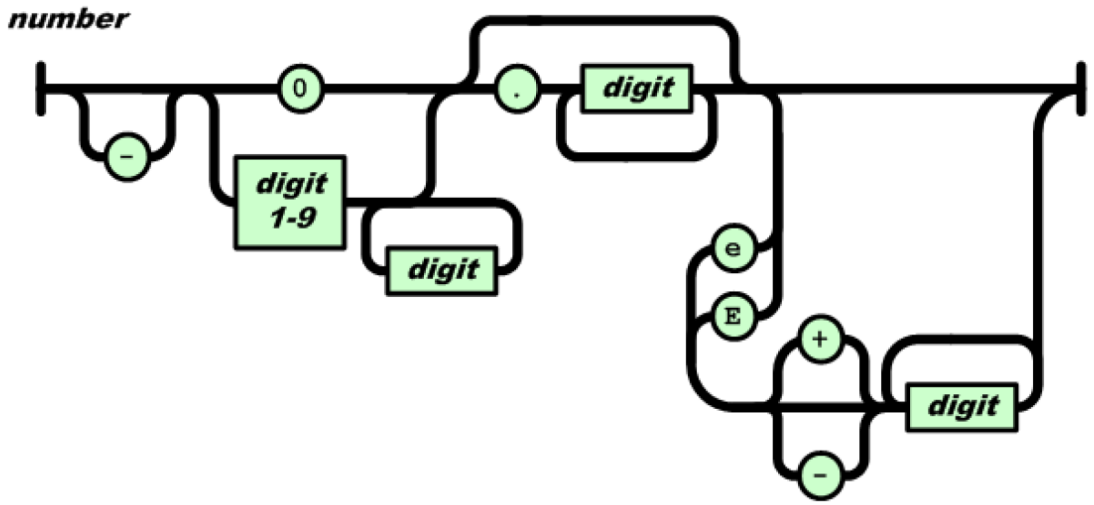


```
{
  "name": "John Doe",
  "age": 30,
  "married": true,
  "siblings": [
    {"name": "John", "age": 25},
    true,
    "Hello World"
  ]
}
```

The array contains 3 items.
The first item is an object,
the second item is a boolean,
and the third item is a string.

JSON NUMBER

A number is an integer or a decimal and it may have an exponent:



```
{
  "name": "John Doe",
  "age": 30,
  "married": true,
  "siblings": ["John", "Mary", "Pat"]
}
```

Example of a JSON number

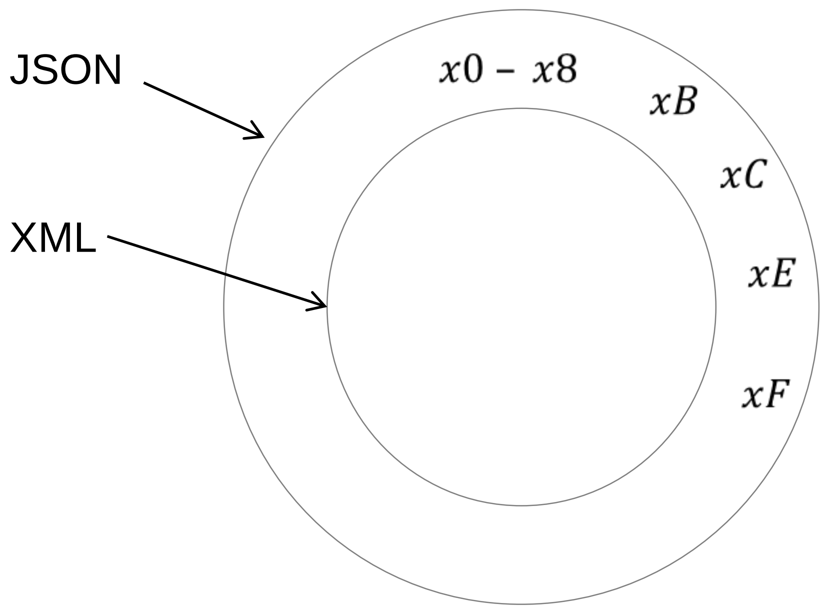
JSON STRING

A string is a sequence of Unicode characters wrapped within quotes (").

```
{
  "name": "John Doe",
  "age": 30,
  "married": true,
  "siblings": ["John", "Mary", "Pat"]
}
```

$\longleftarrow$ Example of a JSON string

JSON chars are a superset of XML chars



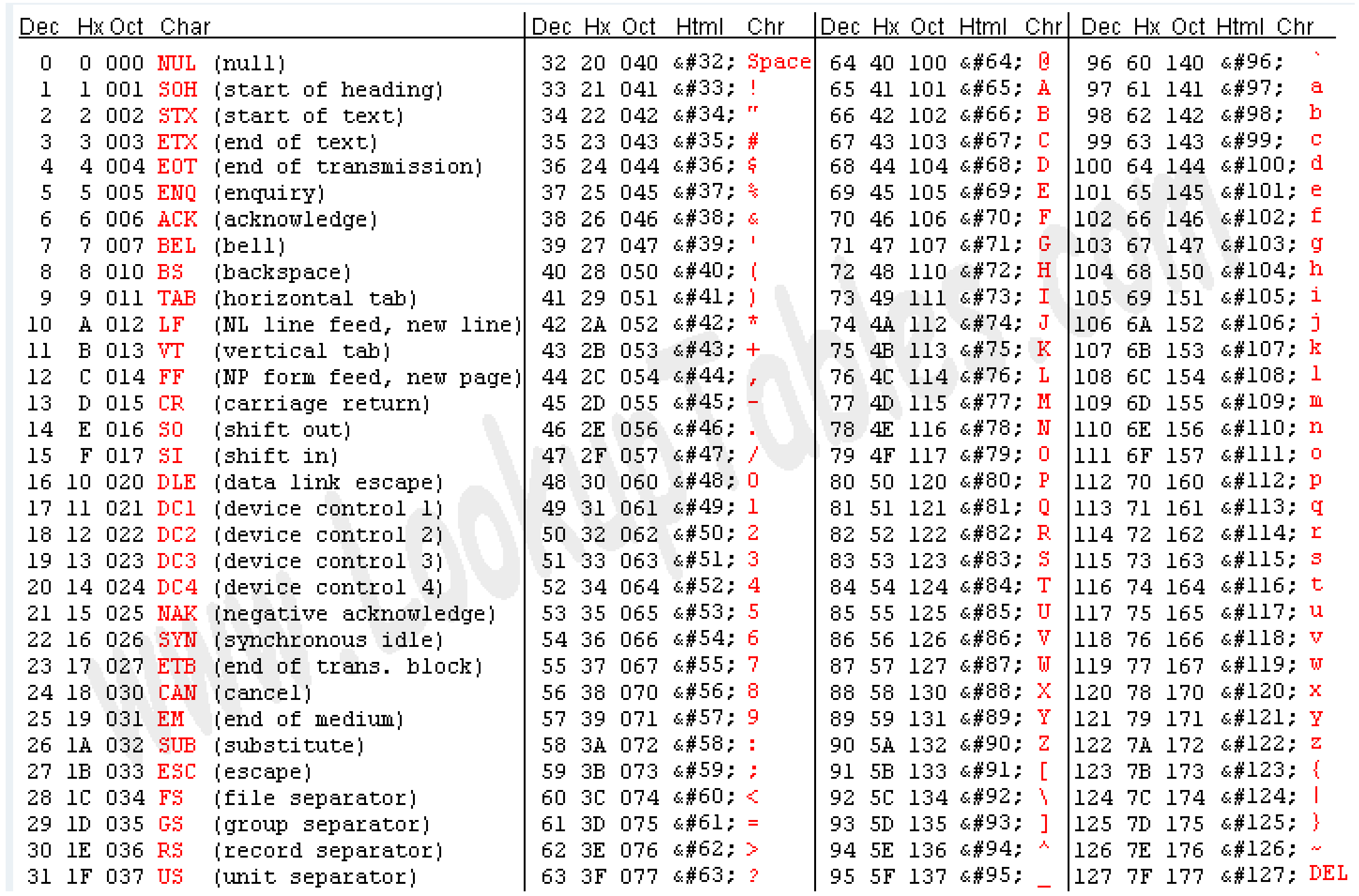
Lesson Learned: be careful converting JSON to XML as the result may be a non-well-formed XML document.

These characters must be escaped

If any of the following characters occur within a string, they must be *escaped* by preceding them with a backslash (\):

- quotation mark ("),
- backslash (\),
- the control characters U+0000 to U+001F

EACH CHARACTER CORRESPONDS TO A NUMBER



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

www.LookupTables.com

ASCII table

EXPRESSING CHARACTERS IN HEX FORMAT

A character may be represented as a hexadecimal number using this notation: \uXXXX

For example, instead of using 'j' in your JSON document, you can use either \u006A or \u006a.

Unicode Details

- Use the \uXXXX notation for Unicode code points in the Basic Multilingual Plane.
- Use a 12-character sequence \uYYYY\uZZZZ to encode code points above the Basic Multilingual Plane.
  - (0xD800 ≤ YYYY < 0xDC00 and 0xDC00 ≤ ZZZZ ≤ 0xDFFF)

NO MULTILINE STRINGS

JSON does not allow multiline strings.

Legal:

```
{
  "comment": "This is a very, very long comment"
}
```

Not legal:

```
{
  "comment": "This is a very,
   very long comment"
}
```

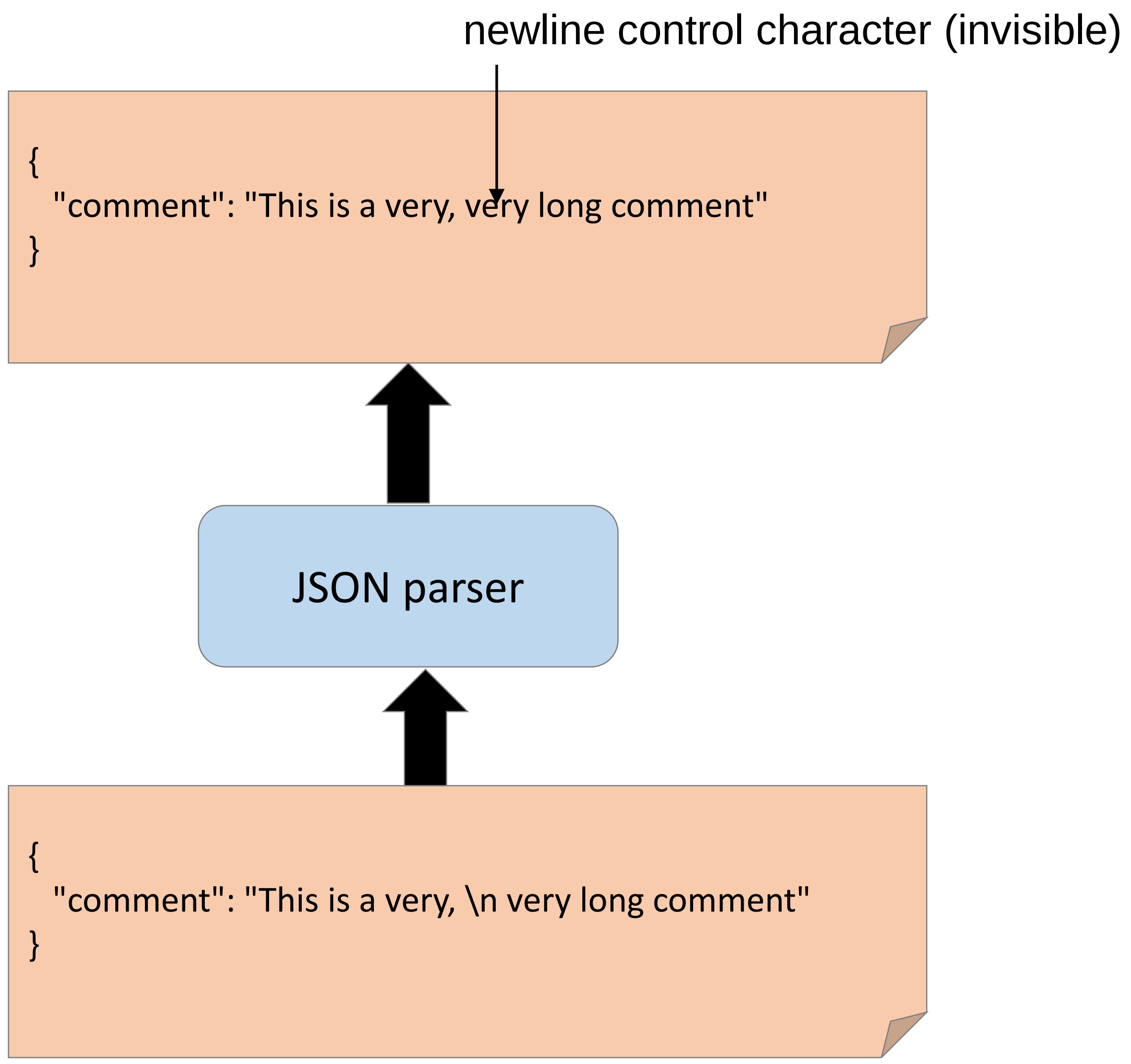
NO MULTILINE STRINGS (CONT.)

Legal:

```
{
  "comment": "This is a very, \n very long comment"
}
```

Just 2 symbols

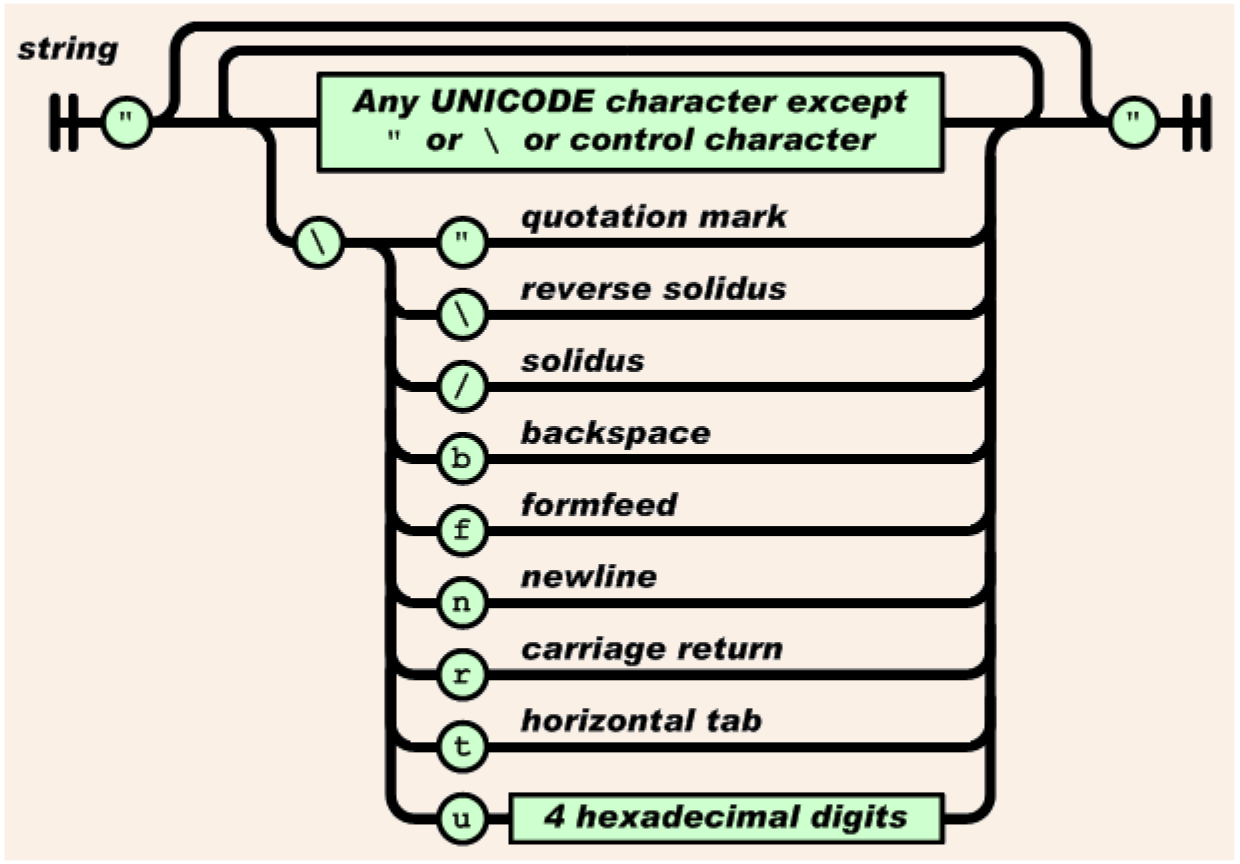
JSON parser converts \n to the newline character



```
{
  "comment": "This is a very, very long comment"
}
```

```
{
  "comment": "This is a very, \n very long comment"
}
```

JSON Strings



http://json.org/

http://stackoverflow.com/questions/2392766/multiline-strings-in-json

Other JSON Values

The values `true`, `false`, and `null` are literal values; they are not wrapped in quotes.

```
{
  "name": "John Doe",
  "age": 30,
  "married": true,    ← Example of a JSON boolean
  "siblings": ["John", "Mary", "Pat"]
}
```

Good use-case for `null`

Some people do not have a middle name, we can use `null` to indicate “no value”:

```
{
  “first-name": "John",
  “middle-name": null,
  “last-name": “Doe"
}
```

Whitespace is irrelevant

```
{
  "name": "John Doe",
  "age": 30,
  "married": true
}
```

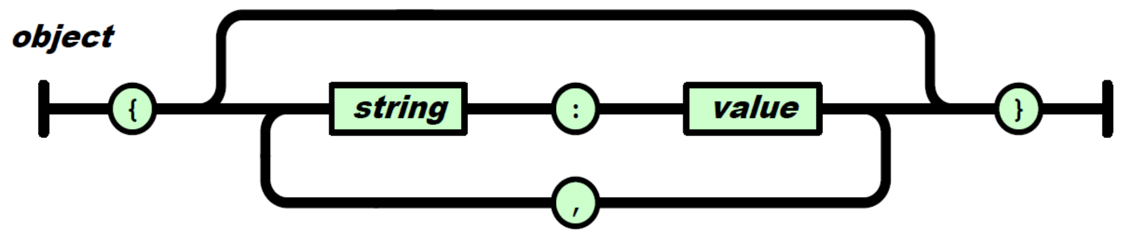
equivalent

```
{"name":"John Doe","age":30,"married":true}
```

STRING DELIMITERS: JSON VS. XML

- JSON strings are always delimited by double quotes.
- XML strings (such as attribute values) may be delimited by either double quotes or single quotes.

JSON is recursively defined



```
{
  "foo": json-value
}
```

The above JSON instance is an object. The object has a single property, "foo". Its value is any JSON value – recursive definition!

Using JSON you can define arbitrarily complex structures

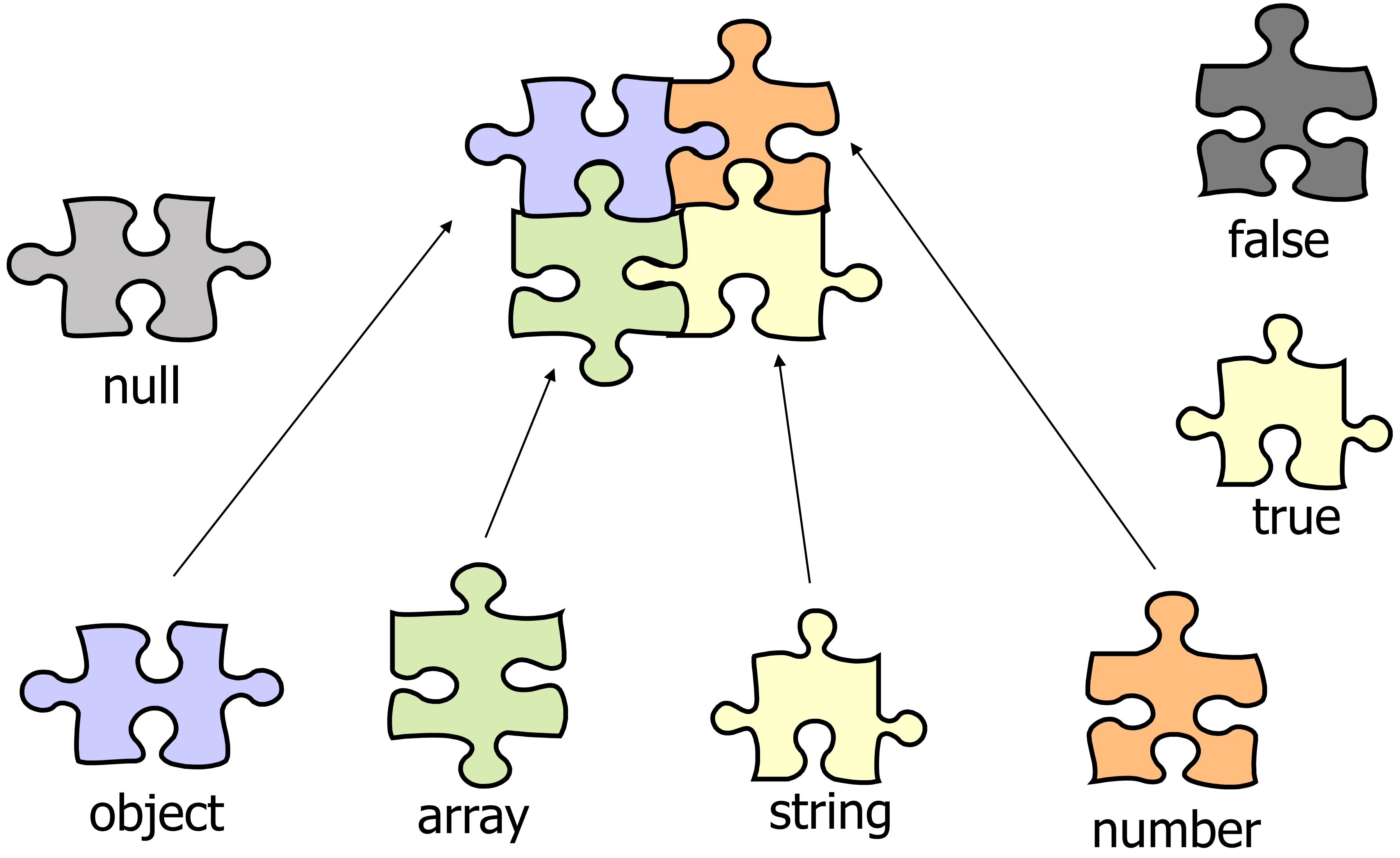
```
{
    "Book":
        {
            "Title": "Parsing Techniques",
            "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ]
        }
}
```

```
{
    "Book":
        {
            "Title": "Parsing Techniques",
            "Authors": [
                {"name":"Dick Grune", "university": "Vrije Universiteit"},
                {"name":"Ceriel J.H. Jacobs", "university": "Vrije Universiteit"}
            ]
      }
}
```

EXTEND, AD INFINITUM

```
{
    "Book":
        {
            "Title": "Parsing Techniques",
            "Authors": [
                {"name": {"first":"Dick", "last":"Grune"},
                 "university": "Vrije Universiteit"},
                {"name": {"first":"Ceriel", "last":"Jacobs"},
                 "university": "Vrije Universiteit"}
            ]
      }
}
```

7 Simple JSON Components, Assemble to Generate Unlimited Complexity



JSON PROVIDES THE STRUCTURES AND ASSEMBLY POINTS, YOU CUSTOMIZE THEM FOR YOUR NEEDS

object

```
{
    "___": json-value,
    "___": json-value,
    "___": json-value,
    …
}
```

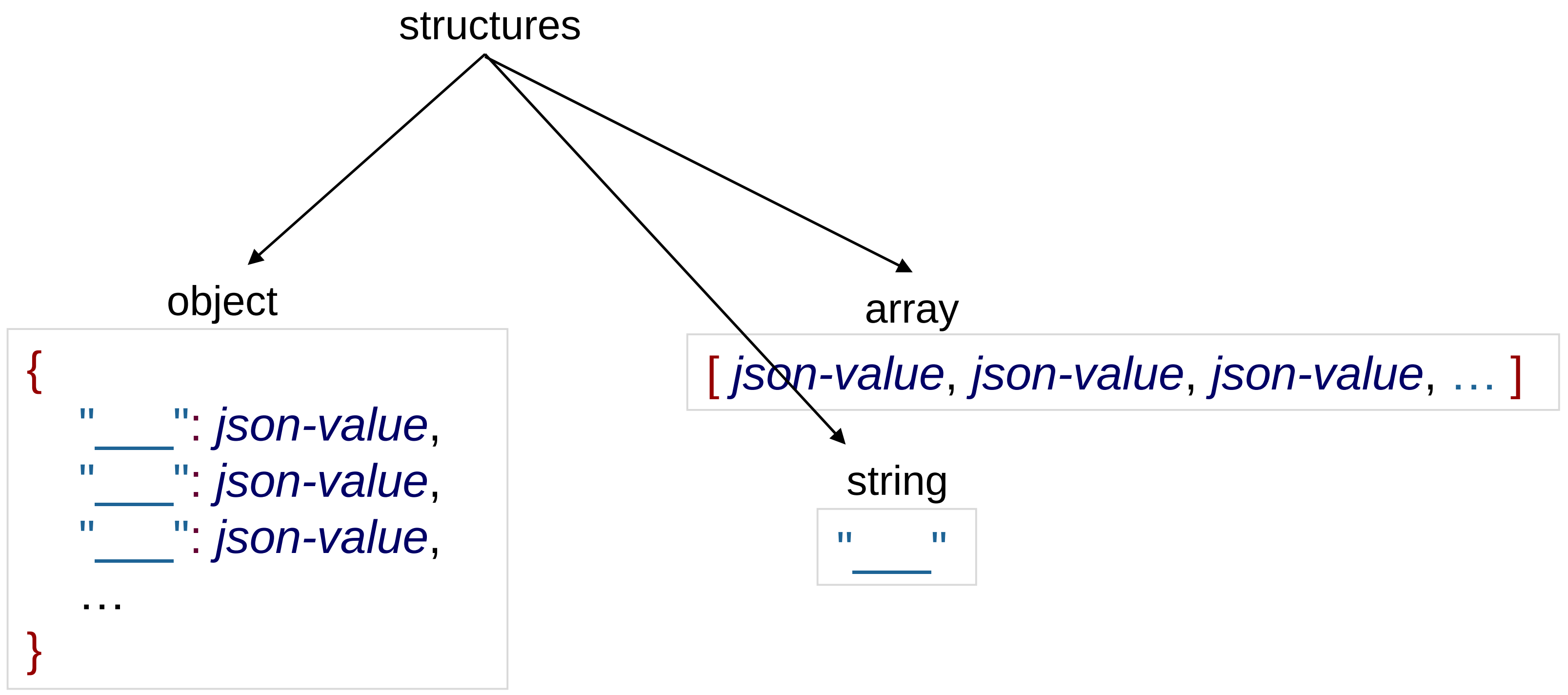
array

```
[ json-value, json-value, json-value, … ]
```

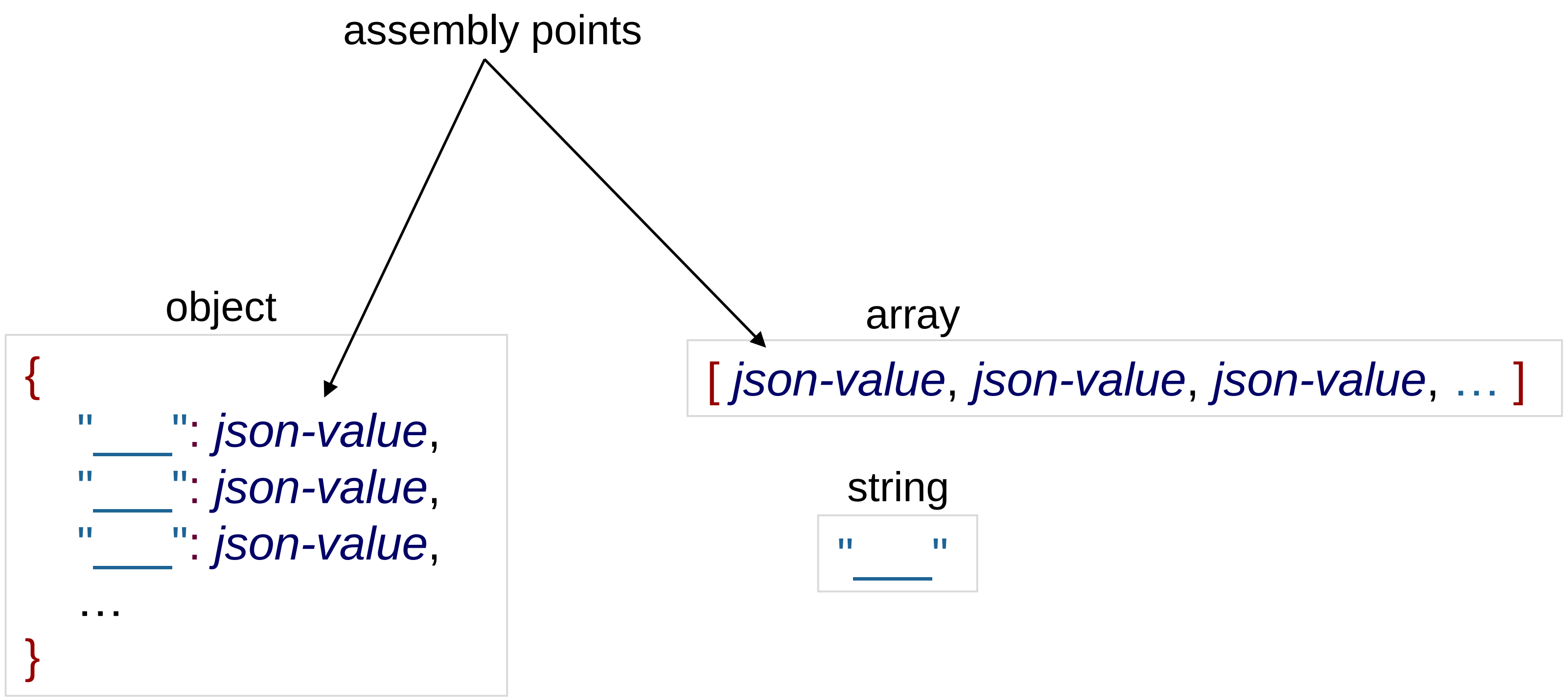
string

```
"___"
```

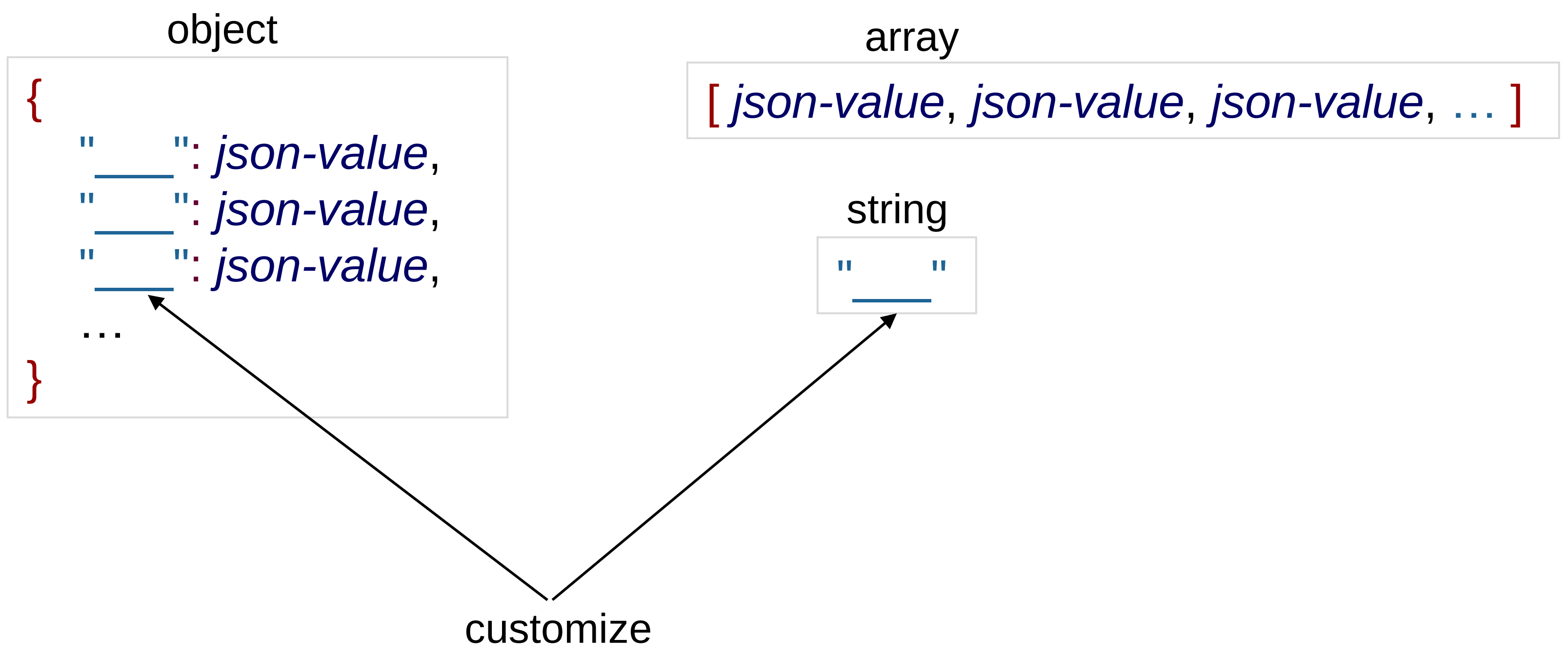
JSON provides the structures and assembly points, you customize them for your needs



JSON provides the structures and assembly points, you customize them for your needs



JSON PROVIDES THE STRUCTURES AND ASSEMBLY POINTS, YOU CUSTOMIZE THEM FOR YOUR NEEDS



COMMENTS NOT ALLOWED!

- You cannot comment a JSON instance document.
- There is no syntax for commenting JSON instances.
- Bummer.

OXYGEN XML SUPPORTS JSON

You can create and edit JSON instances and JSON Schemas using oXygen XML and it will check that the document is correctly formatted. Nice!

Evaluation

- JSON is simpler than XML and more compact
  - No closing tags, but if you compress XML and JSON the difference is not so great
  - XML parsing is hard because of its complexity
- JSON has a better fit for OO systems than XML
- JSON is not as extensible as XML
- Preferred for simple data exchange by many
- Less syntax, no semantics
- Schemas? We don’t need no stinkin schemas!
- Transforms? Write your own.

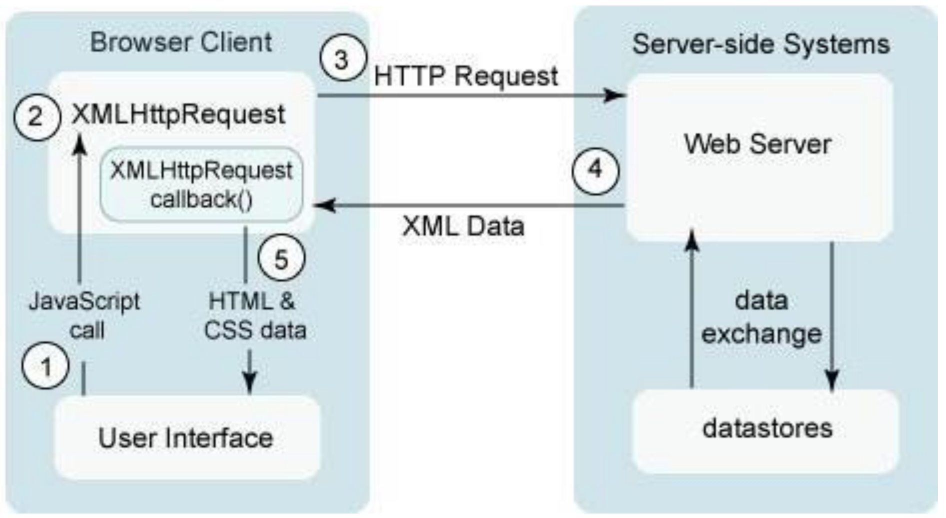
SUMMARY REVIEW

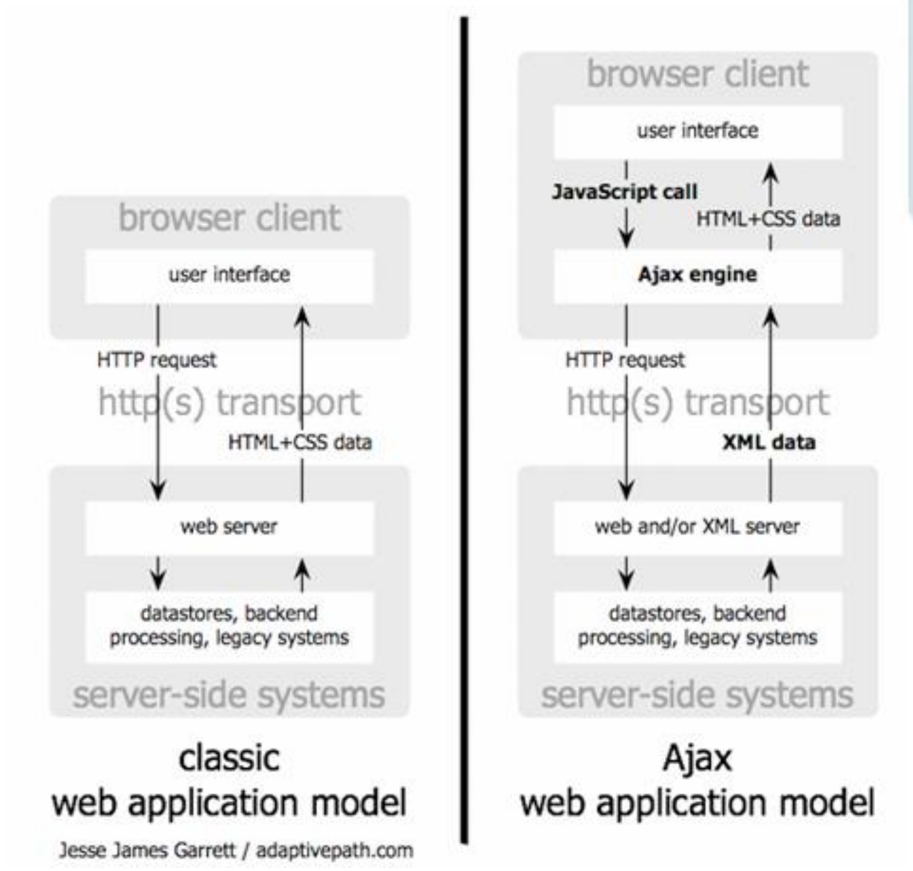
What is AJAX

- Asynchronous JavaScript and XML
- Combination of technologies, that can transfer data (using HTTP protocol) and change page without reloading of whole page.
- Based on earlier ideas (IFRAME, LAYER, Applets, etc)
- Advantages
  - Greater user comfort and effective using of web applications
  - Lower requirements to amount of transferred data
- Disadvantages
  - Elimination of the „back functionality“ in browser
  - Changes inside page does not affect as it is (URL)

COMMUNICATION MODEL

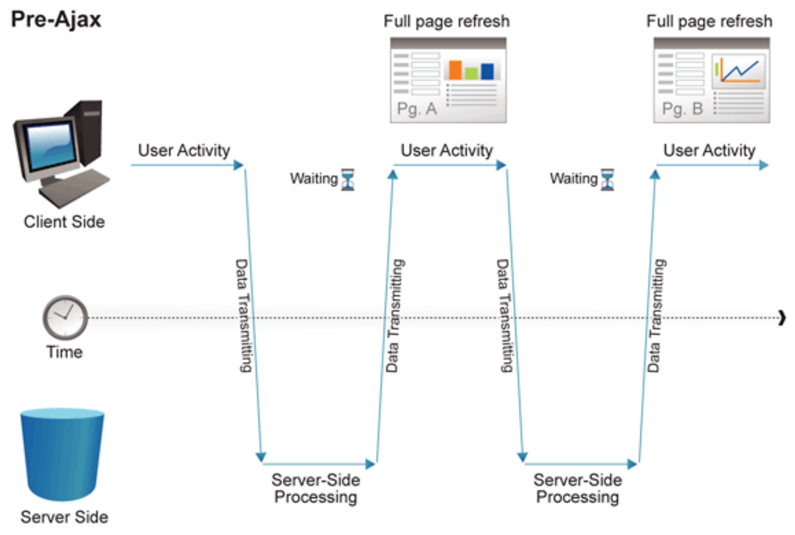
Zdroj: http://www.eioba.com

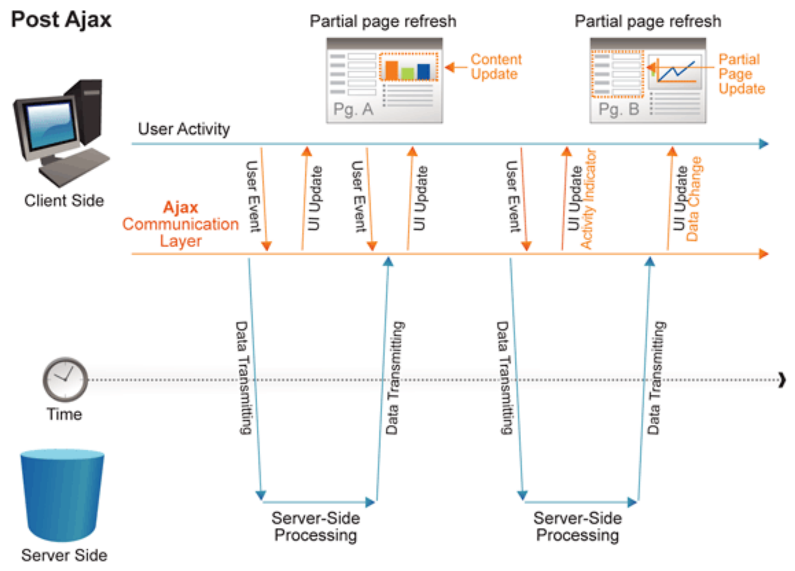




COMMUNICATION MODEL

Zdroj: http://www.websiteoptimization.com





IMPLEMENTATION OF AJAX – OLD BROWSERS

- DOM and XmlHttpRequest

```
if (window.XMLHttpRequest) {
   http_request = new XMLHttpRequest();
 } else if (window.ActiveXObject) {
    try {
      http_request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (eror) {
      http_request = new ActiveXObject("Microsoft.XMLHTTP");
    }
 }
```

Object creation

```
http_request.onreadystatechange = function() { process(http_request); };

http_request.open('POST', 'synonyma.php', true);
http_request.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
http_request.send(/* "any data" */);



function process(http_request) {
      if (http_request.readyState == 4) {
          if (http_request.status == 200) {
              alert(http_request.responseText);
          } else {
              alert("ERROR!");
          }
      }
  }
```

Dispatch AJAX request

IMPLEMENTATION OF AJAX – MODERN BROWSERS

```
var req = new XMLHttpRequest();
req.addEventListener("load", function(){
        console.log(req.responseText);
});
req.open("GET", "http://www.example.org/example.txt");
req.send("lorem=ipsum&name=janousek");
```

```
req.addEventListener("progress", updateProgress);
req.addEventListener("load", transferComplete);
req.addEventListener("error", transferFailed);
req.addEventListener("abort", transferCanceled);
```

Events

```
var data = new FormData();
data.append('myVariableName', 12345);
data.append('file', input.files[0]);

var req = new XMLHttpRequest();
req.addEventListener("load", function(){
        console.log(req.responseText);
});
req.setRequestHeader('Content-Type', 'multipart/form-data');
req.open("POST", "http://www.example.org/example.txt");
req.send(data);
```

File upload

AJAX and jQuery

Zdroj: http://www.ibm.com

```
$('#stats').load('stats.html');
```

Load HTML content

```
$.post('save.php', {
    text: 'my string',
    number: 23
}, function(data) {
    alert('Your data has been saved.');
    console.log(“Server data:“, data);
});
```

Send data to the server (POST method)

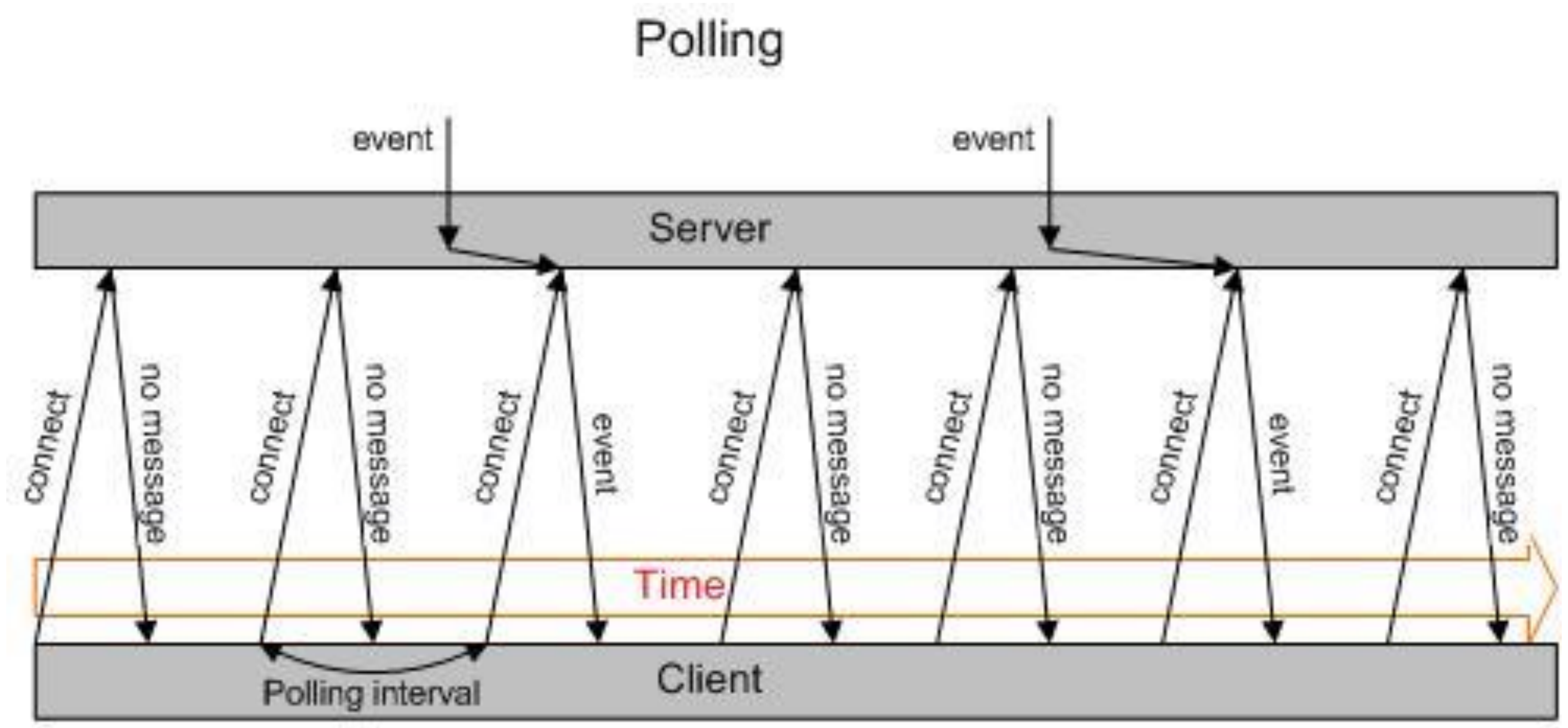
```
$.ajax({
    url: 'document.xml',
    type: 'GET',
    dataType: 'xml',
    timeout: 1000,
    error: function(){
        alert('Error loading XML document');
    },
    success: function(xml){
        $(xml).find('item').each(function(){
        var item_text = $(this).text();

        $('<li></li>')
            .html(item_text)
            .appendTo('ol');
    });
    }
});
```

Complete example of parsing XML in AJAX request

ASYNCHRONOUS APPROACHES

Zdroj: http://pic.dhe.ibm.com/infocenter

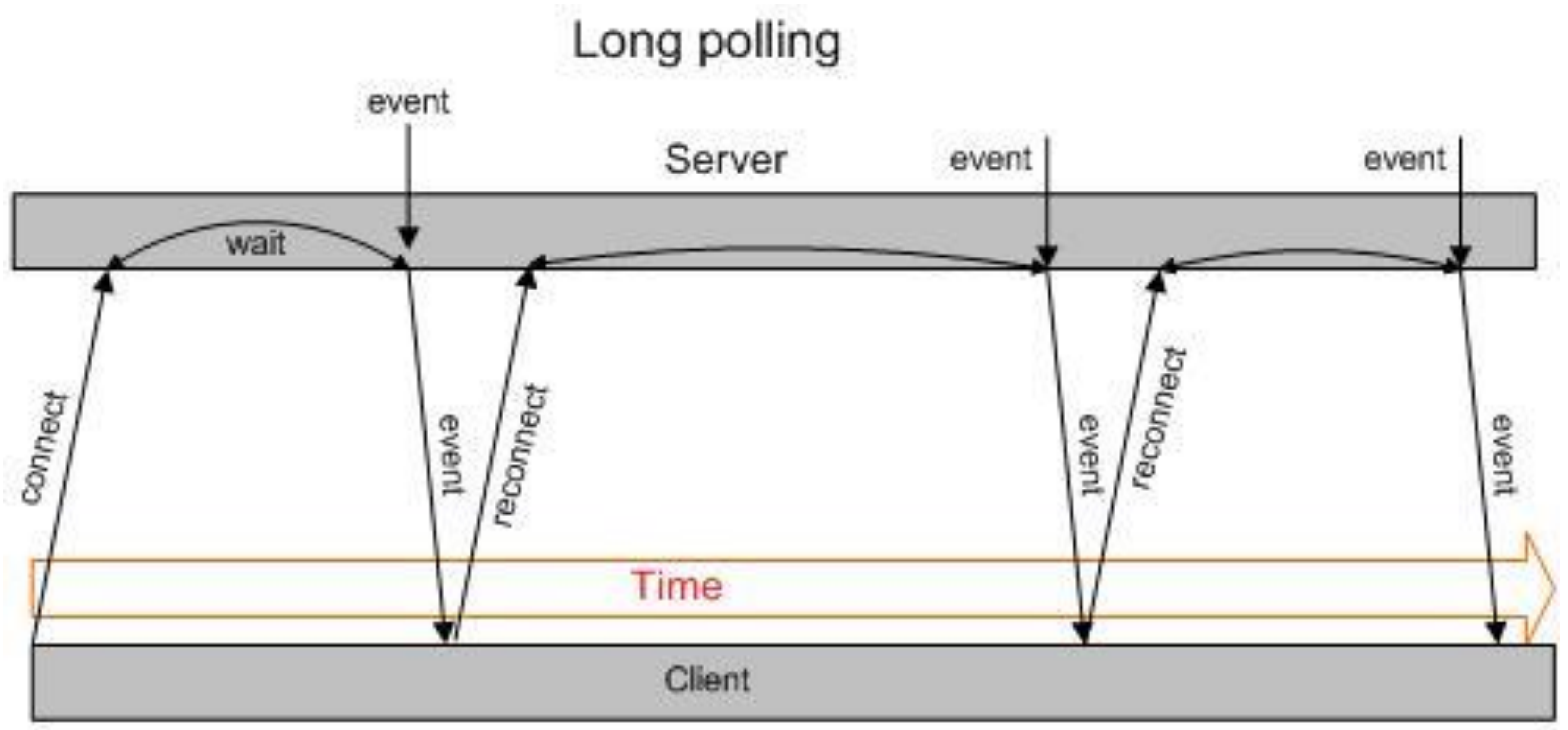
- Polling



Asynchronous approaches

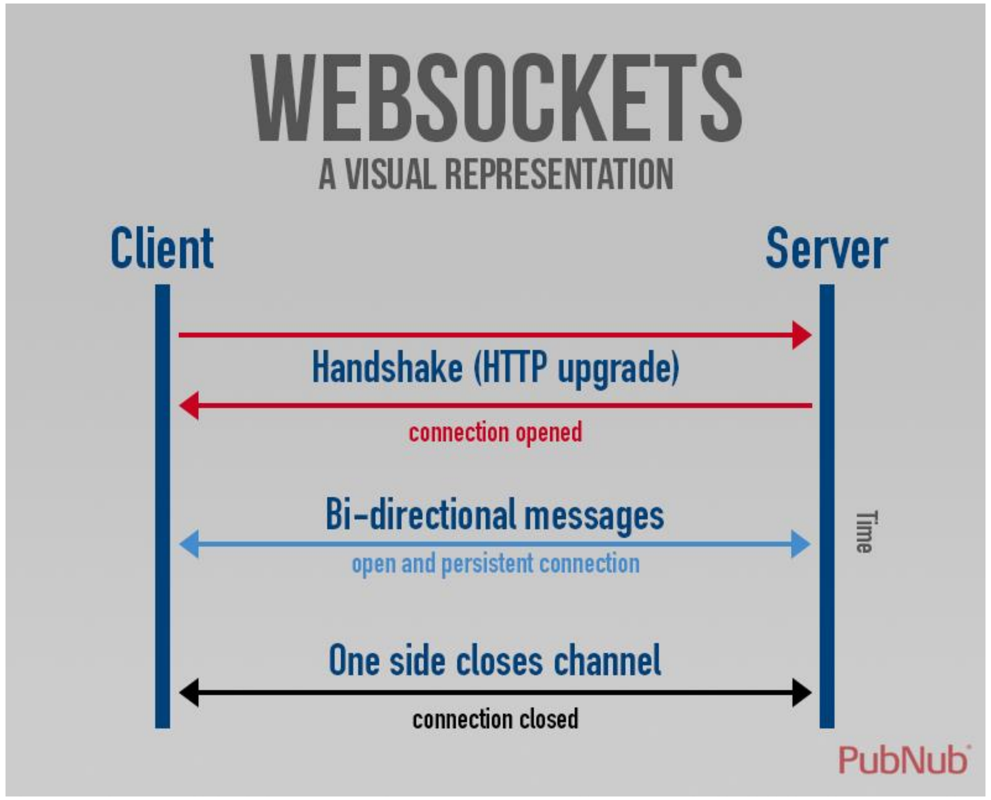
Zdroj: http://pic.dhe.ibm.com/infocenter

- Long - polling



WebSockets

- Persistent bidirectional communication channel
- Object WebSocket in JS
- send, onmessage, onopen, onerror, readyState



WHAT IS XML

- eXtensible Markup Language
- a set of rules
  - semantic tags (tags, elements)
  - divide the document based on structure
  - identification of a part of document
- based on SGML (Standard Generalized Markup Language)
  - same possibilities
  - simplicity
- it is not another markup language
  - it is meta-language
  - the names of tags, attributes, etc. are based only on author or standard

Why use XML?

- data + tags = structured data with
- semantic
- enables definition of relations between elements
- it can be 100% ASCII text
- it is documented by W3C
- it is not patented, without copyright or another limitations
- there are not versions of XML (XML itself)
- support in programming language
- support in many tools
- simple processing - parsing

Format of XML

- Elements/Tags
  - The most of the markup are tags
    - t ag is everything what starts with ‘<‘ and is finished by ‘>’
    - tag has a name
      - has to start [a-z,A-Z,_]
      - it is case sensitive (<B> vs. <b>)
  - Empty tag
    - without content, can have attributes
    - it is possible to use shortcut ‘/>‘

      ```
      <empty />
      <empty></empty>
      ```
  - beware of the character entities

```
<tag atribut=„value">
  data
</tag>
```

Znaková entita	znak
`&amp;`	&
`&lt;`	<
`&gt;`	>
`&quot;`	"
`&apos;`	'
`&#37;`	%
...	...

```
<section>
  <headline>Markup</headline>
  <text>
    Lower then(&lt;)
    and ampersand (&amp;)
have to be used as entities
  </text>
</section>
```

Format of XML

- Attributes
  - Pair **`name = value`**
  - name
    - Have to begin with character **`[a-z,A-Z,_]`**
    - Can be used only once per tag
  - value
    - *String in quotes*
    - Can be any character
    - Quotes in text have to be used as entity (&quot;)

DATA PLACEMENT

- XML data can be
  - Within tag´s attributes
  - Within tags
- recommendation
  - own data within tags
  - meta-data within attributes
  - Common information for attributes
    - IDs
    - URLs
    - not so important data for reader

```
<activity creation="06/08/2000">
```

```
<activity>
  <creation day="08" month="06" year="2000" />
  ...
```

```
<activity>
  <creation>
    <day>08</day>
    <month>06</month>
    <year>2000</year>
  </creation>
  ...
```

ANOTHER SPECIFICATIONS

- Comments
  - Starts with “**<!--**” adn ends with “-
- Text withou interpretation
  - Section **CDATA**

```
<![CDATA[
for (int i = 0; i < array.length && error
== null; i++)
]]>
```

-
- XML

```
<?xml version="1.0" encoding=„UTF-8”?>
```

- MIME-type specification
  - application/xml, text/xml
  - application/mathml+xml, application/XSLT+xml, image/svg+xml

NAMESPACE

- Namespace
  - is used for separation a different sets of elements based on prefix
  - specification and usage in the form **xmlns:name**
  - also valid for child elements
- definition of NS refers to URI (it do not have to exists)

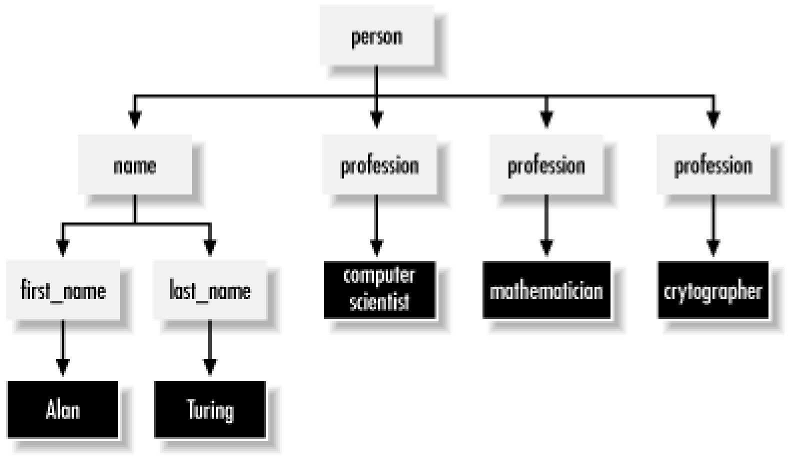
```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="keyword">
  ...
  </xsl:template>
</xsl:stylesheet>
```

```
<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform">
  <template match="keyword">
    <!-- undeclare default namespace -->
    <content-item xmlsn="">
      ...
```

PARENT, CHILDS, ROOT, ...

Zdroj: http://docstore.mik.ua/orelly/xml/xmlnut

- XML document is based on tree structure
- Just one root element is allowed
- No element crossing
- It is possible to define parent or childs of each element (parent max. 1, childs 0 or more)



```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

DTD

- Document Type Definition
- Specification of language for description of rules and facilities of XML document creation
- It enables correct validation of XML document
- Defines
    - a list of elements, attributes, notices and entities
    - type of content of elements
    - relations between elements
    - structure
- It is located
    - within prolog behind declaration
    - before first element
- Tutorial: http://www.w3schools.com/xml/xml_dtd_intro.asp

```
<!DOCTYPE person[
   ...
]>
```

```
<!DOCTYPE person SYSTEM
  "http://abc.com/xml/dtds/person.dtd">
```

DTD – ELEMENT DECLARATION

```
<!ELEMENT element_name content_specification>
```

- ANY
  - any value is allowed (other elements or #PCDATA)
- EMPTY
  - element has no content
- (#PCDATA)
  - parsed character data (text node)
- (child1, child2, …)
  - declaration of child elements list
  - it is able to use regular count definition (child1?, child2+, child3*)
- (child1 | child2)
  - declaration of OR operation
  - It is able to combine definition thanks to brackets

```
<!ELEMENT html (head, body)>
<!ELEMENT name (last_name
               | (first_name, ( (middle_name+, last_name) | (last_name?) )
               ) >
```

DTD – Attribute Declaration

```
<!ATTLIST element_name attribute_name
content_specification default_value>
```

- CDATA
  - text for parsing
- NMTOKEN, NMTOKENS
  - value for name specification (without spaces, etc.), e.g. name in HTML
- (Monday|Thursday|Friday)
- – ennumaration of allowed values
- ID
  - unique identification in whole document, max. one attribute of this type
- IDREF, IDREFS
  - attribute to define relation to element with ID
- ENTITY, ENTITIES
  - value refers to specific entity
- „value“
  - default value
- #IMPLIED
  - attribute is optional
- #REQUIRED
  - attribute is neccessary

```
<!ATTLIST img
   src     CDATA              #REQUIRED
   id      ID                 #IMPLIED
   print   (yes | no)  "yes"
>
```

DTD and sample XML

Zdroj: http://www.idevelopment.info/data/Programming/java/xml/ExampleXMLandDTDFile.html

XML

```xml
<?xml version="1.0"?>
<!DOCTYPE DatabaseInventory SYSTEM "DatabaseInventory.dtd">

<DatabaseInventory>

 <DatabaseName>
  <GlobalDatabaseName>production.iDevelopment.info</GlobalDatabaseName>
  <OracleSID>production</OracleSID>
  <DatabaseDomain>iDevelopment.info</DatabaseDomain>
  <Administrator EmailAlias="jhunter" Extension="6007">Jeffrey Hunter</Administrator>
  <DatabaseAttributes Type="Production" Version="9i"/>
  <Comments>
   The following database should be considered the most stable for
   up-to-date data. The backup strategy includes running the database
   in Archive Log Mode and performing nightly backups. All new accounts
   need to be approved by the DBA Group before being created.
  </Comments>
 </DatabaseName>

 <DatabaseName>
  <GlobalDatabaseName>development.iDevelopment.info</Glob
  <OracleSID>development</OracleSID>
  <DatabaseDomain>iDevelopment.info</DatabaseDomain>
  <Administrator EmailAlias="jhunter" Extension="6007">Jeffrey
  <Administrator EmailAlias="mhunter" Extension="6008">Melo
  <DatabaseAttributes Type="Development" Version="9i"/>
  <Comments>
   The following database should contain all hosted applications.
   data will be exported on a weekly basis to ensure all developm
   have stable and current data.
  </Comments>
 </DatabaseName>

 <DatabaseName>
  <GlobalDatabaseName>testing.iDevelopment.info</GlobalData
  <OracleSID>testing</OracleSID>
  <DatabaseDomain>iDevelopment.info</DatabaseDomain>
  <Administrator EmailAlias="jhunter" Extension="6007">Jeffrey
  <Administrator EmailAlias="mhunter" Extension="6008">Melo
  <Administrator EmailAlias="ahunter">Alex Hunter</Administra
  <DatabaseAttributes Type="Testing" Version="9i"/>
  <Comments>
   The following database will host more than half of the testing
   for our hosting environment.
  </Comments>
 </DatabaseName>

</DatabaseInventory>
```

DTD

```dtd
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT DatabaseInventory (DatabaseName+)>
<!ELEMENT DatabaseName (    GlobalDatabaseName
                          , OracleSID
                          , DatabaseDomain
                          , Administrator+
                          , DatabaseAttributes
                          , Comments)
>
<!ELEMENT GlobalDatabaseName (#PCDATA)>
<!ELEMENT OracleSID          (#PCDATA)>
<!ELEMENT DatabaseDomain     (#PCDATA)>
<!ELEMENT Administrator      (#PCDATA)>
<!ELEMENT DatabaseAttributes EMPTY>
<!ELEMENT Comments           (#PCDATA)>

<!ATTLIST Administrator       EmailAlias CDATA #REQUIRED>
<!ATTLIST Administrator       Extension  CDATA #IMPLIED>
<!ATTLIST DatabaseAttributes  Type       (Production|Development|Testing)
#REQUIRED>
<!ATTLIST DatabaseAttributes  Version    (7|8|8i|9i)  "9i">
```

XML Schema Definition (XSD)

- DTD disadvantages
  - does not support namespaces
  - does not support data types for elements content and attributes
  - DTD is not XML
- XML Schema
  - specification language based on XML
  - standardized by W3C
  - Defines
    - structure of XML document
    - elements and attributes of XML document
    - child elements, their number and order
    - content of element (empty/nonempty)
    - data types of elements and attributes (over 40 types)
    - default and fixed values
  - everything is covered by namespace xs:
- Tutorial: http://www.w3schools.com/schema/default.asp

XSD

Zdroj: http://www.kosek.cz

```
<?xml version="1.0" encoding="UTF-8"?>
<employees>
   <employee id="101">
      <name>Jan</name>
      <surname>Novák</surname>
      <email>jan@novak.cz</email>
      <email>jan.novak@firma.cz</email>
      <fee>25000</fee>
      <born>1965-12-24</born>
   </employee>
   <employee id="101">
      <name>Petra</name>
      <surname>Mal8</surname>
      <email>petra.mala@aaa.cz</email>
      <fee>45000</fee>
      <born>1975-10-21</born>
   </employee>
</employees>
```

XML

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="employees">
    <xs:complexType>
      <xs:sequence>
        <xs:element name=" employee"
                    maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="surname" type="xs:string"/>
              <xs:element name="email" type="xs:string"
                          maxOccurs="unbounded"/>
              <xs:element name="fee" type="xs:decimal"
                          minOccurs="0"/>
              <xs:element name="born" type="xs:date"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:int"
                          use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

W3C XML Schema

```
<!ELEMENT employees (employee+)>
<!ELEMENT employee (name, surname, email+,
                     fee?, born)>
<!ELEMENT name        (#PCDATA)>
<!ELEMENT surname     (#PCDATA)>
<!ELEMENT email        (#PCDATA)>
<!ELEMENT fee         (#PCDATA)>
<!ELEMENT born      (#PCDATA)>
<!ATTLIST employee
          id     CDATA #REQUIRED>
```

DTD

XSD – Element declaration

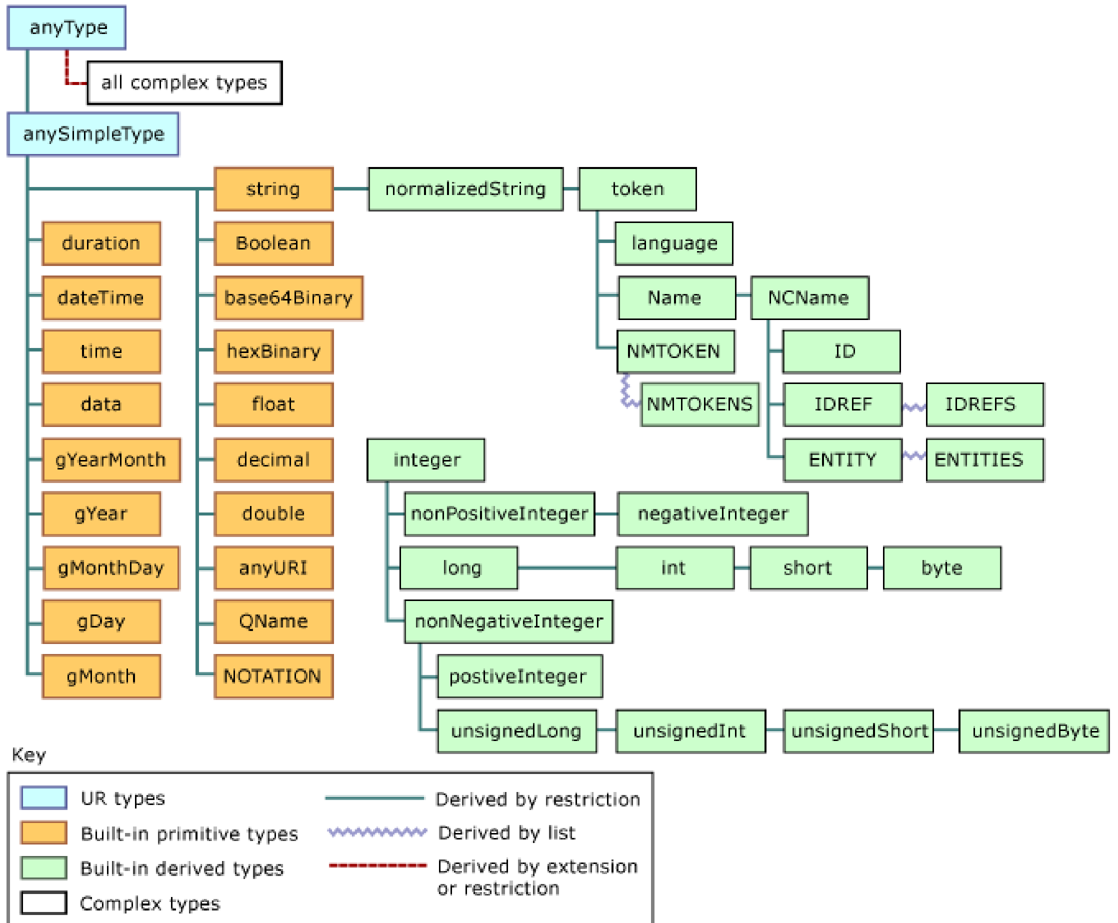
```
<xs:element name=„name" type=„type"/>
```

simple element

- Name corresponds to general rules for names
- On of many available types
- New inherited types

```
<xs:simpleType name=„nameType">
  <xs:restriction base="xs:string">
    <xs:minLength value="1"/>
    <xs:maxLength value="15"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name=„currencyType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="CZK"/>
    <xs:enumeration value="EUR"/>
    <xs:enumeration value="USD"/>
  </xs:restriction>
</xs:simpleType>
```



XSD – Attributes declaration

- Attribute itself is simple-element as a part of complex-elementu

complex element

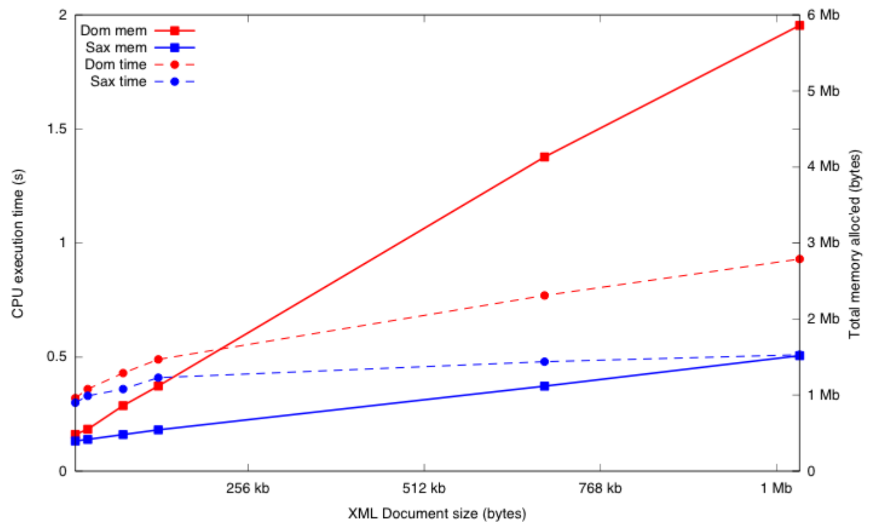
```
<xs:element name=„name“>
   <xs:complexType>
      <xs:sequence>
              <xs:element …/>
      </xs:sequence>
      <xs:attribute name=„name" type=„type"
                          use="required"/>
   </xs:complexType>
</xs:element>
```

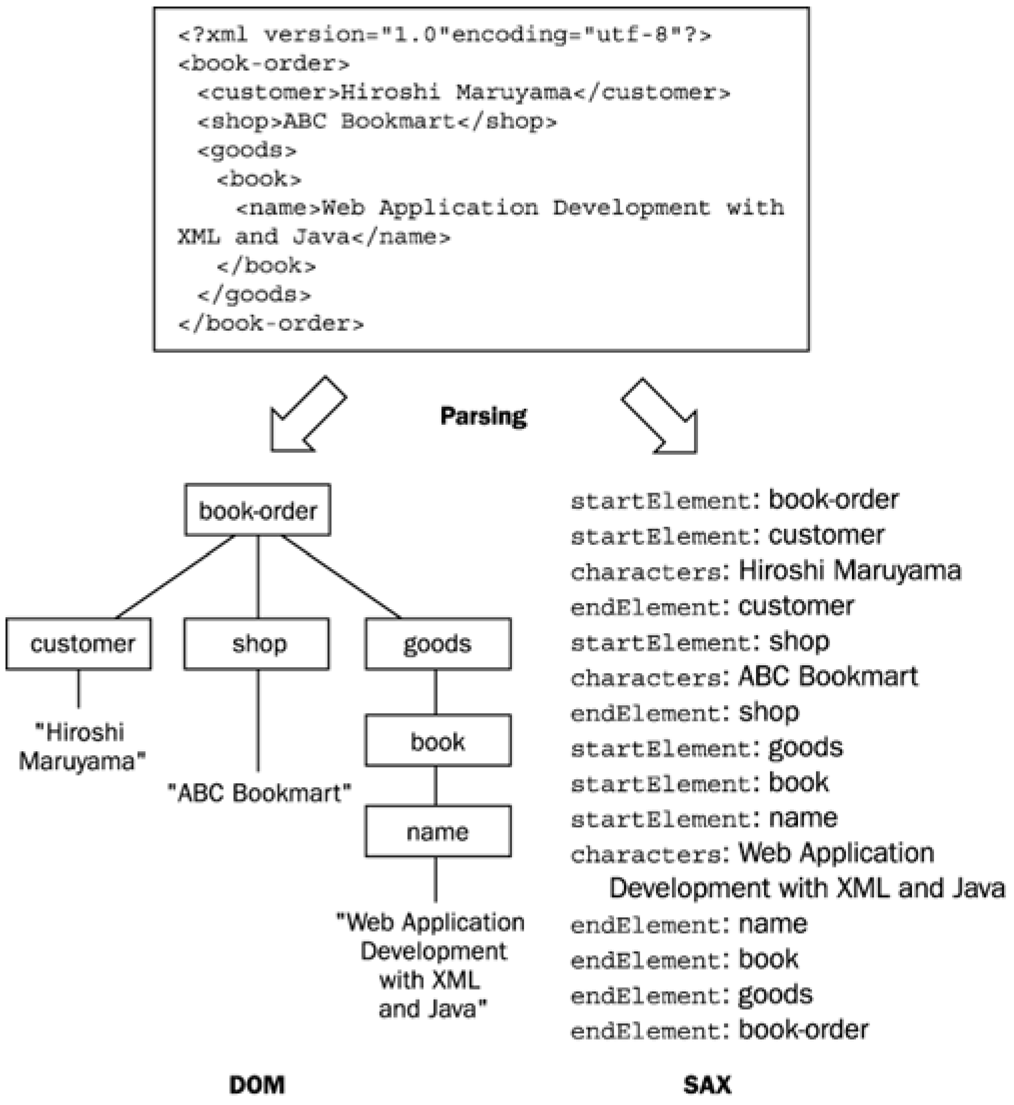
API

- DOM
  - Document Object Model
  - represents tree structure of XML document by objects in memory
- SAX
  - Simple API for XML – event-driven model
  - processing of XML during document loading
  - calling methods/parsing loaded data at the beginning and the end of element, during text parsingm etc.
  - it is not a standard
  - fast, higher claims to application logic
- Parser
  - application, class, algorithm
  - transfer XML from text form to another form for next processing (for example. DOM)
  - check syntax and DTD and validation

DOM vs. SAX

Zdroj: http://tech.inhelsinki.nl/2007-08-29/, http://book.javanb.com





```
<?xml version="1.0"encoding="utf-8"?>
<book-order>
  <customer>Hiroshi Maruyama</customer>
  <shop>ABC Bookmart</shop>
  <goods>
    <book>
      <name>Web Application Development with
XML and Java</name>
    </book>
  </goods>
</book-order>
```

```
startElement: book-order
startElement: customer
characters: Hiroshi Maruyama
endElement: customer
startElement: shop
characters: ABC Bookmart
endElement: shop
startElement: goods
startElement: book
startElement: name
characters: Web Application
   Development with XML and Java
endElement: name
endElement: book
endElement: goods
endElement: book-order
```

JAVASCRIPT

- Transform XML to DOM

```
var parser = new DOMParser();
var doc = parser.parseFromString(xmlString, "application/xml");
```

AJAX variant (old)

```
function verifyfunc() {
    if (xmlDoc.readyState != 4) {
        return false;
    }
}
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
xmlDoc.onreadystatechange=verifyfunc;
xmlDoc.load('xmltest.xml');
var xmlObj=xmlDoc.documentElement;
```

- Work with DOM

```
function WriteXML() {
    var t= „Father: " + xmlObj.childNodes(0).text + " (born " +
xmlObj.childNodes(0).getAttribute(„date") + ")\n"
    t += „Mother: " + xmlObj.childNodes(1).text + " (born " +
xmlObj.childNodes(1).getAttribute(„date") + ")\n\n"
    t += „Children:\n"
    var i;
    for(i=0; i<xmlObj.childNodes(2).childNodes.length; i++ ) {
        t += " " + xmlObj.childNodes(2).childNodes(i).text + " (born " +
xmlObj.childNodes(2).childNodes(i).getAttribute(„date") + ")\n"
    }
    alert(t);
}
```

XPath - Path

- Path (Path Expression) is the fundamental construction element for query specification
- Similar to the paths in OS
- Separators „/“ or „//“
- More than one sequence thanks to OR or „|“
- Each step includes
  - axes identificator
  - node condition
  - predicate
- Path is evaluated from the left to right and relatively to the current node

```
axisname::nodetest[predicate]
```

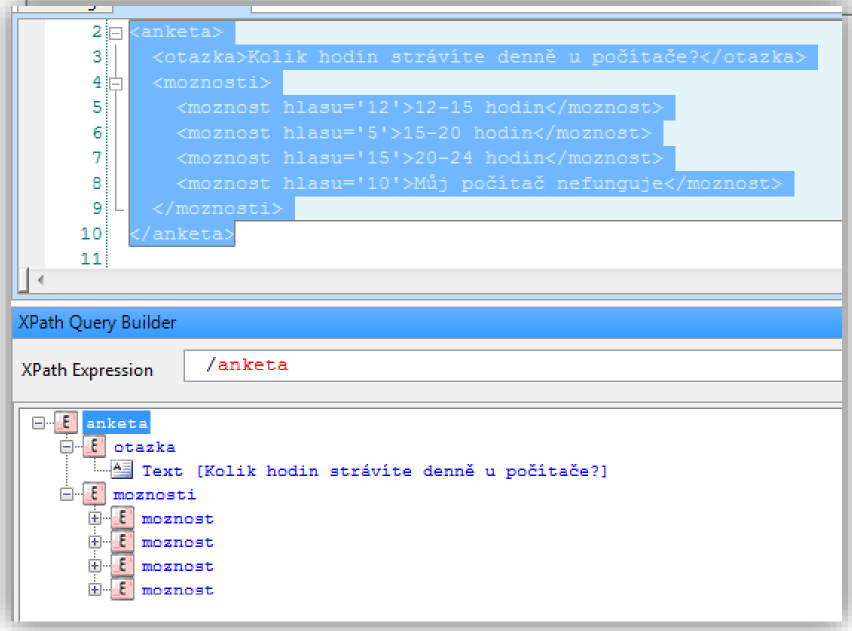
```
/pool/descendant::option[@votes>10]/text()
```

XPATH – STEPS

Zdroj: http://interval.cz/clanky/zaklady-jazyka-xpath/

Source XML file

```
<pool>
  <question>Kolik hodin strávíte denně u počítače?</question>
  <options>
    <option votes='12'>12-15 hodin</option>
    <option votes='5'>15-20 hodin</option>
    <option votes='15'>20-24 hodin</option>
    <option votes='10'>Můj počítač nefunguje</option>
  </options>
</pool>
```

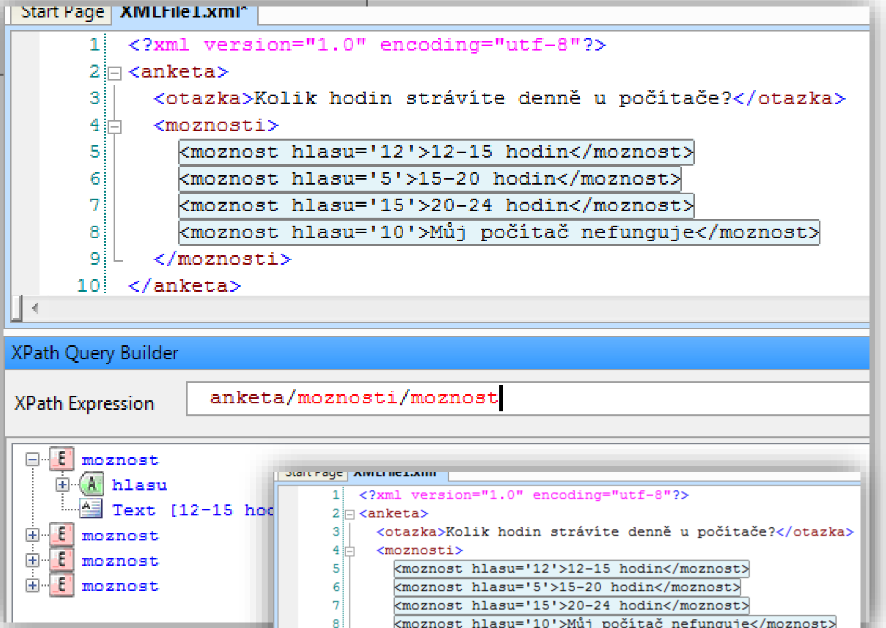


```
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/anketa`

```
anketa
  otazka
    Text [Kolik hodin strávíte denně u počítače?]
  moznosti
    moznost
    moznost
    moznost
    moznost
```



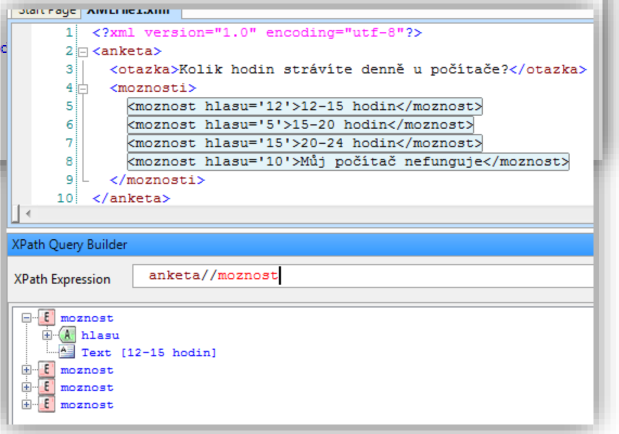
Start Page | XMLFile1.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `anketa/moznosti/moznost`

```
moznost
  hlasu
  Text [12-15 hod
moznost
moznost
moznost
```



```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

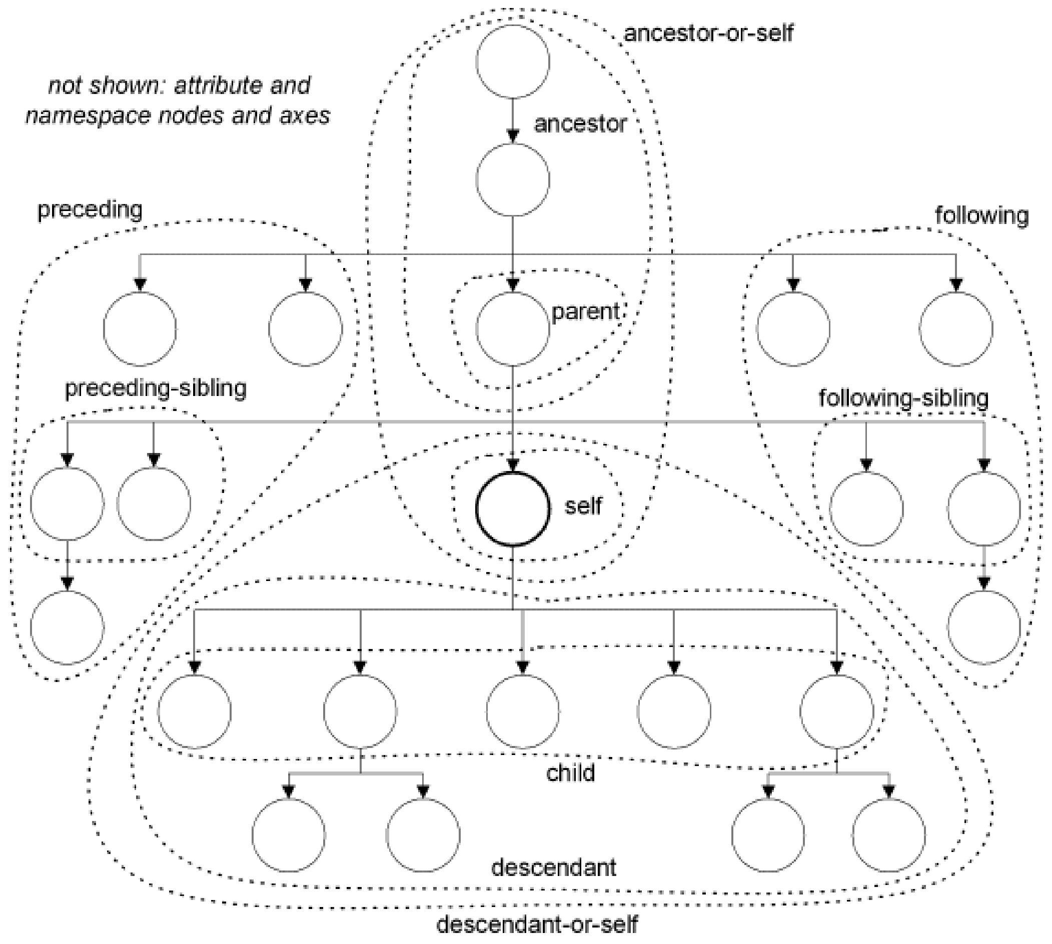
XPath Expression `anketa//moznost`

```
moznost
  hlasu
  Text [12-15 hodin]
moznost
moznost
moznost
```

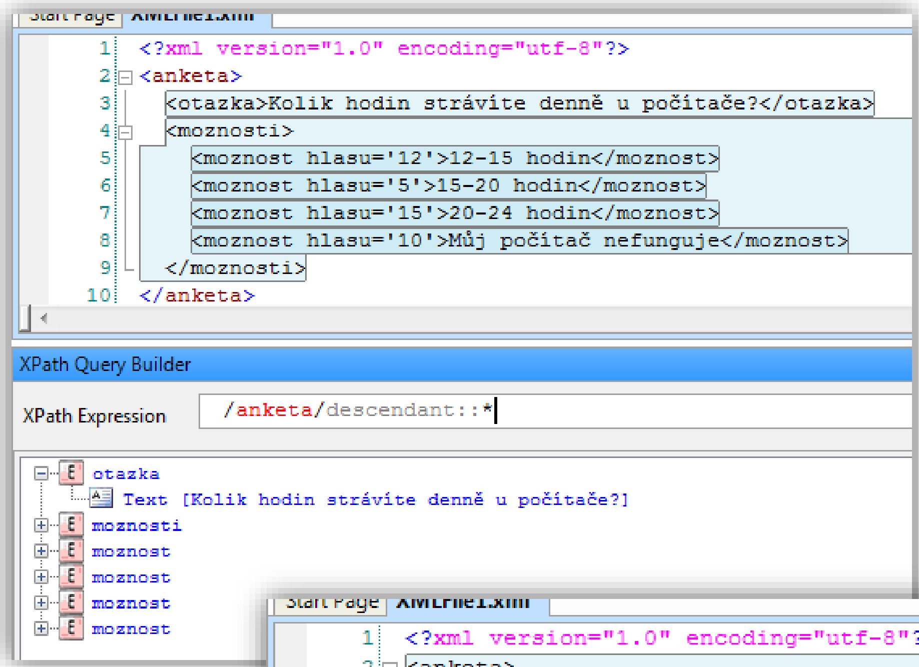
XPATH - AXES

Zdroj: http://www.georgehernandez.com

- Specifies a direction of searching within XML tree
- A set of relevant nodes incomming to the testing, default is child::
- Axes *ancestor*, *descendant*, *following*, *preceding* and *self* are disjoint and together cover all tree nodes



XPATH - AXES



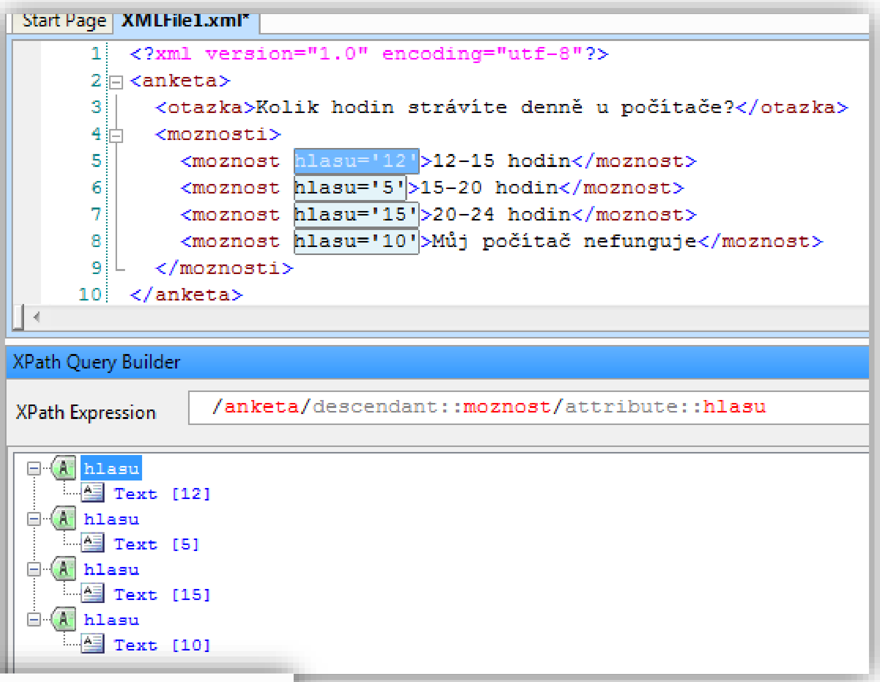
Start Page | XMLFile1.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/anketa/descendant::*`

```
⊟ E otazka
     Text [Kolik hodin strávíte denně u počítače?]
⊞ E moznosti
⊞ E moznost
⊞ E moznost
⊞ E moznost
⊞ E moznost
```



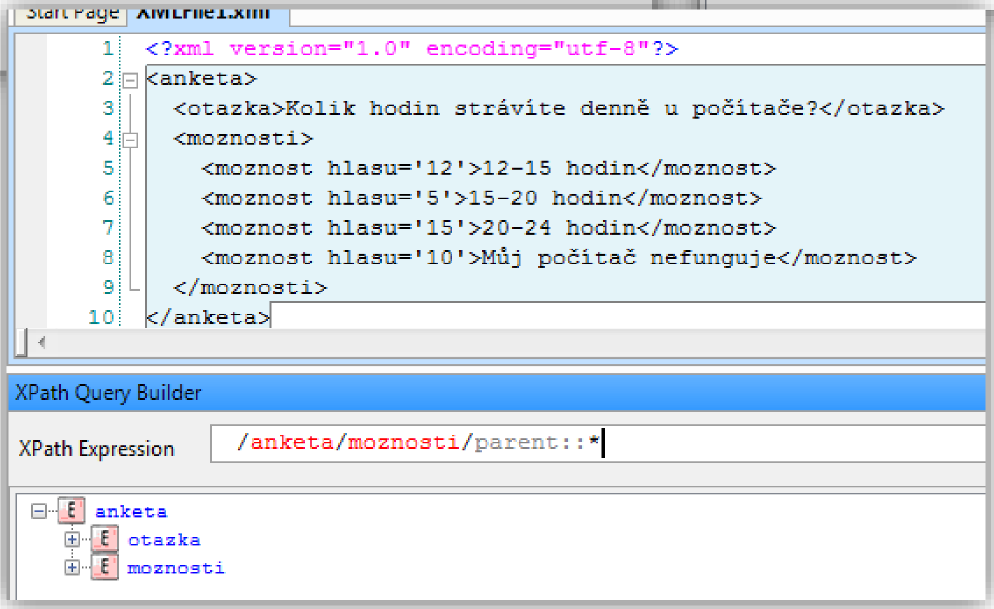
Start Page | XMLFile1.xml*

```xml
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/anketa/descendant::moznost/attribute::hlasu`

```
⊟ A hlasu
     Text [12]
⊟ A hlasu
     Text [5]
⊟ A hlasu
     Text [15]
⊟ A hlasu
     Text [10]
```



Start Page | XMLFile1.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

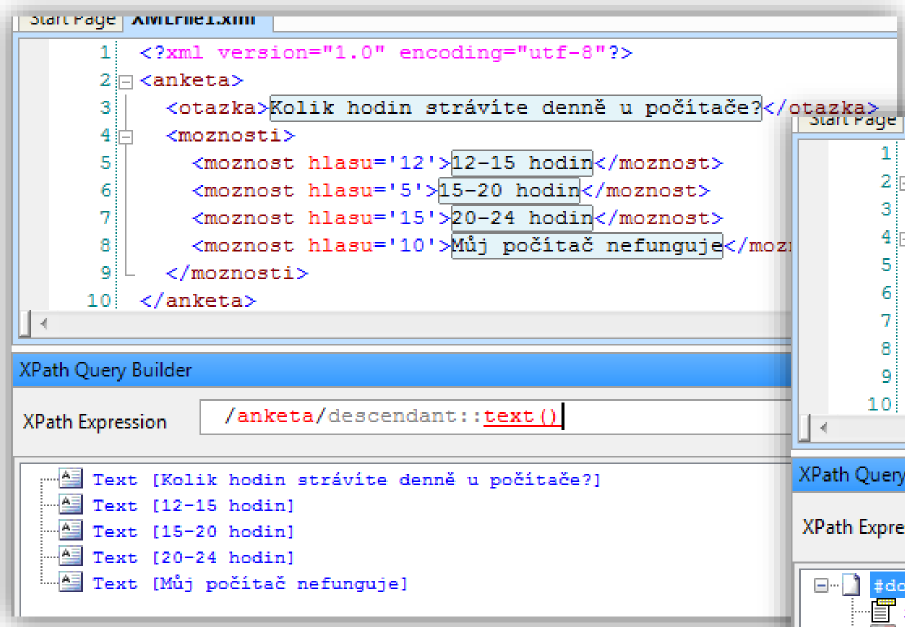
XPath Query Builder

XPath Expression `/anketa/moznosti/parent::*`

```
⊟ E anketa
   ⊞ E otazka
   ⊞ E moznosti
```

XPath – Node tests

- Node specification
    - by name (namespace prefix included)
    - by type (text(), node(), comment(),
    - processing-instruction()

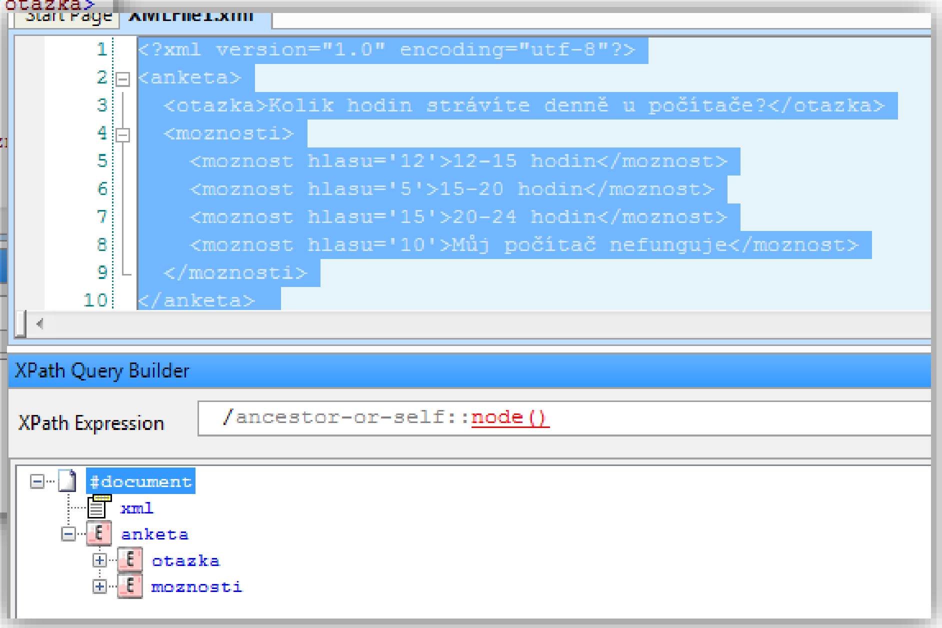


```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moz
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/anketa/descendant::text()`

```
Text [Kolik hodin strávíte denně u počítače?]
Text [12-15 hodin]
Text [15-20 hodin]
Text [20-24 hodin]
Text [Můj počítač nefunguje]
```



```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

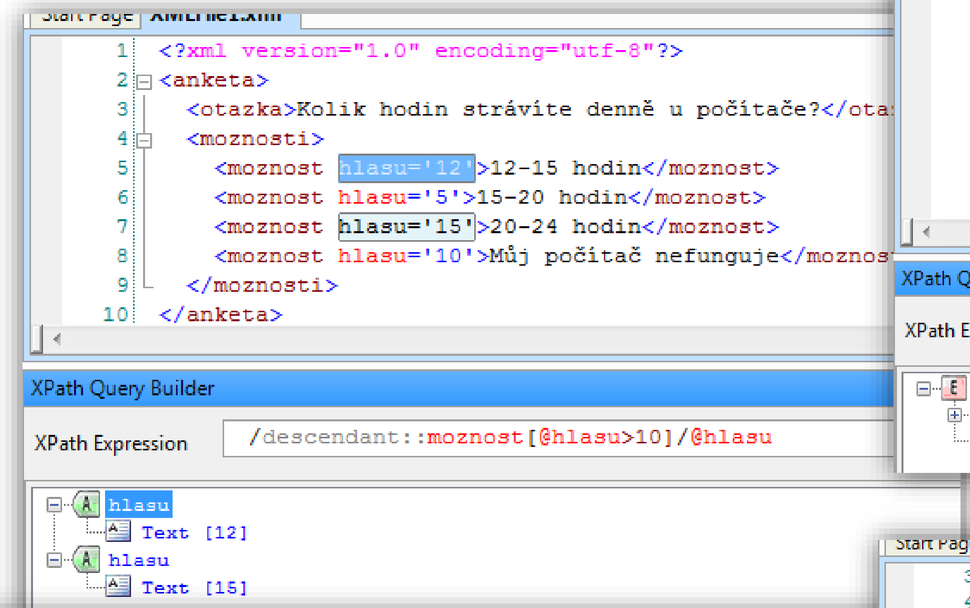
XPath Expression `/ancestor-or-self::node()`

```
#document
  xml
  anketa
    otazka
    moznosti
```

XPATH – PREDICATS, ETC.

- It is possible to use
  - Characters „*“, „.“, „..“
  - Mathematics, relate and logic operators)
  - Shortcut character „@“ for attribute:: axe
  - Functions (approx. 100 functions) (last(),
  - position(), string(), concat(), atd.)
- Conditions can be constructed with respect to all parts of a given element (axes, nodes, attributes)

XPATH



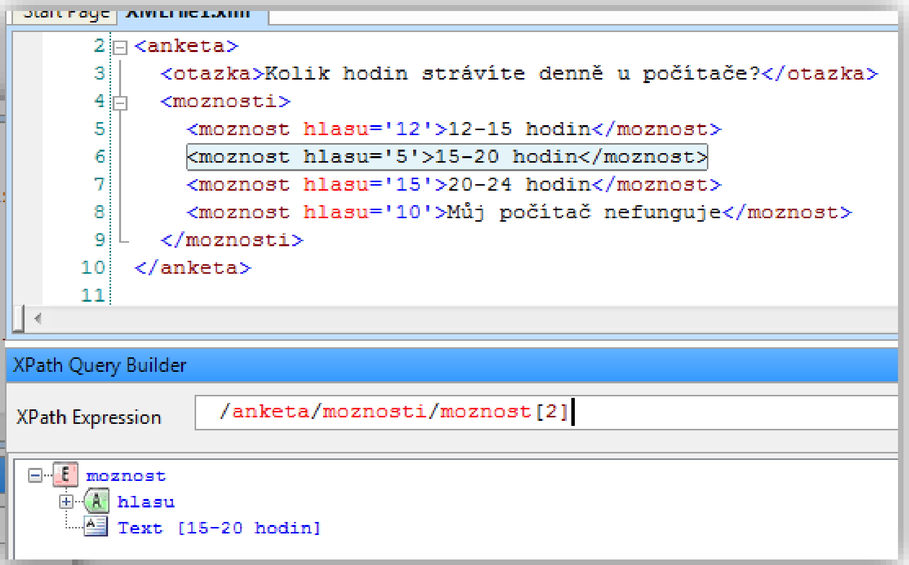
Start Page | XMLFile1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</ota
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznos
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/descendant::moznost[@hlasu>10]/@hlasu`

```
hlasu
  Text [12]
hlasu
  Text [15]
```



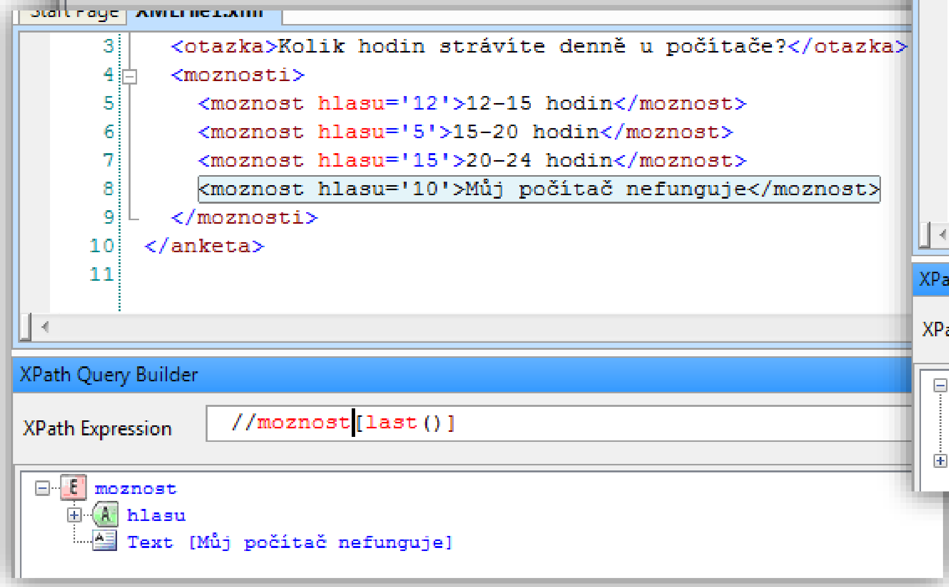
Start Page | XMLFile1.xml

```
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/anketa/moznosti/moznost[2]`

```
moznost
  hlasu
  Text [15-20 hodin]
```



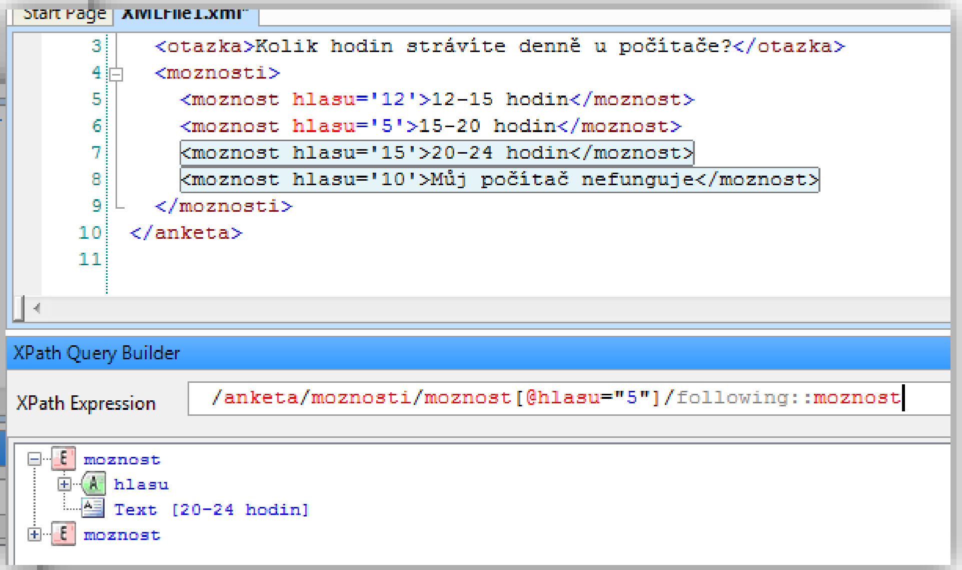
Start Page | XMLFile1.xml

```
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `//moznost[last()]`

```
moznost
  hlasu
  Text [Můj počítač nefunguje]
```



Start Page | XMLFile1.xml

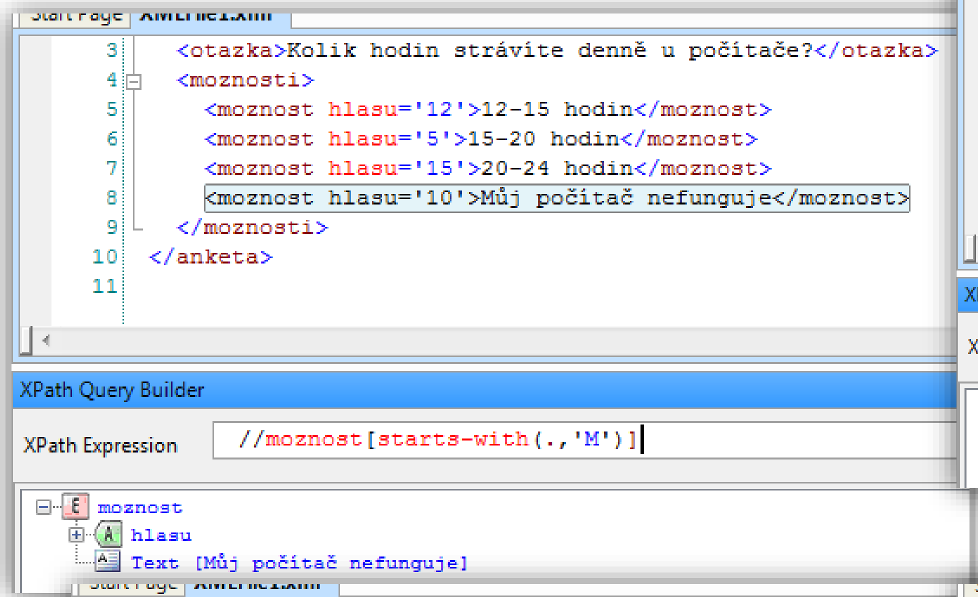
```
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `/anketa/moznosti/moznost[@hlasu="5"]/following::moznost`

```
moznost
  hlasu
  Text [20-24 hodin]
moznost
```

XPATH

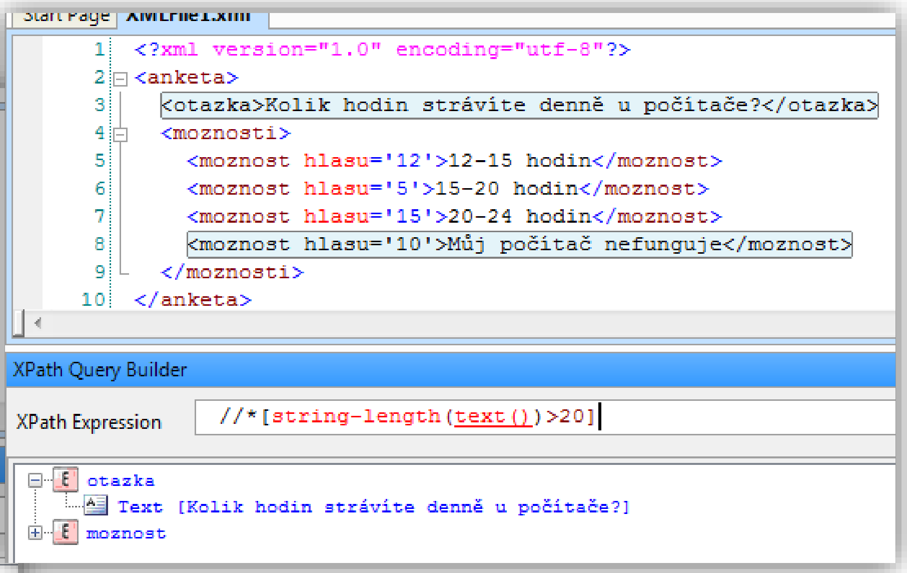


```
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `//moznost[starts-with(.,'M')]`

```
moznost
  hlasu
  Text [Můj počítač nefunguje]
```

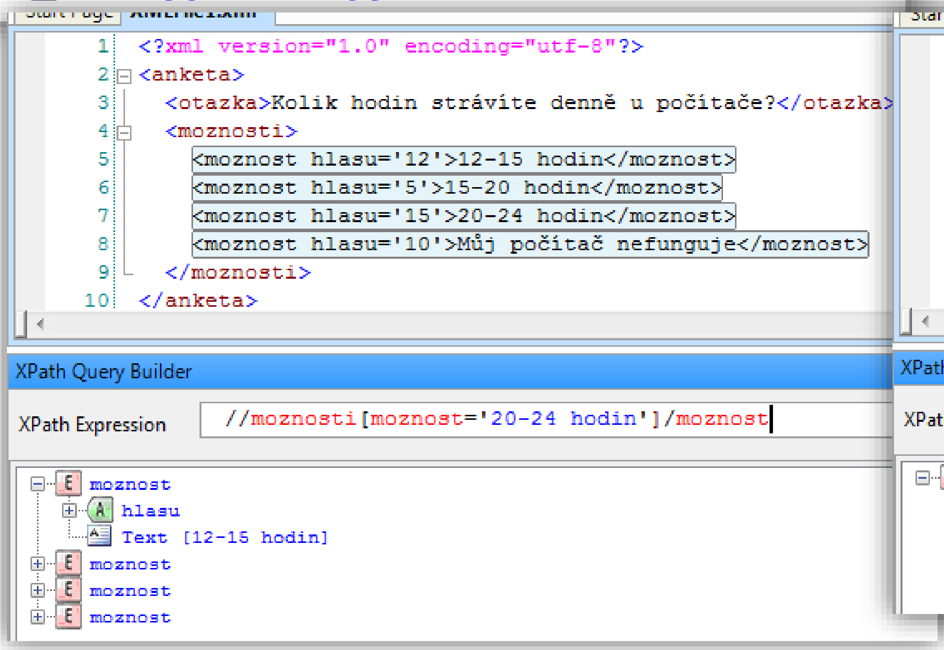


```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `//*[string-length(text())>20]`

```
otazka
  Text [Kolik hodin strávíte denně u počítače?]
moznost
```

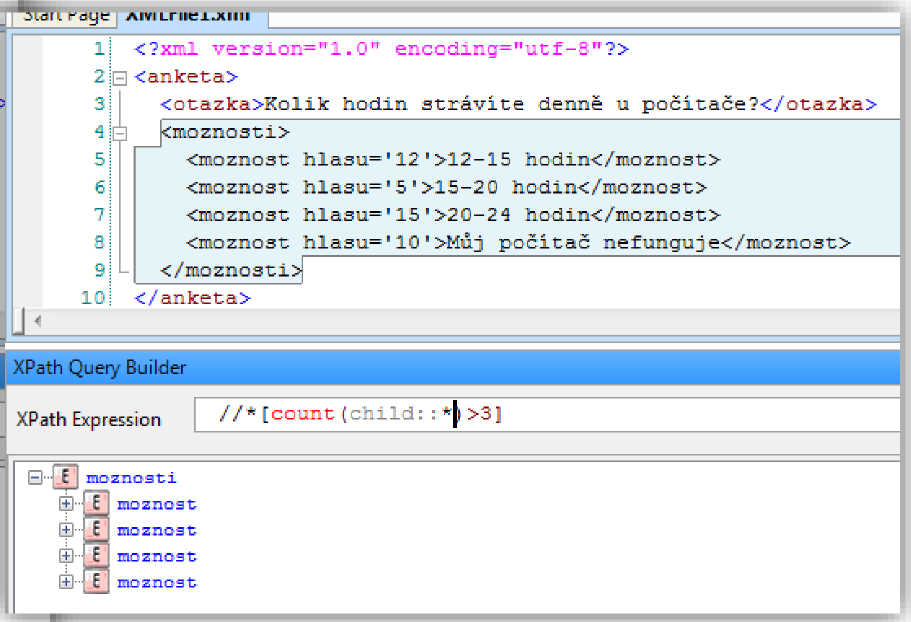


```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `//moznosti[moznost='20-24 hodin']/moznost`

```
moznost
  hlasu
  Text [12-15 hodin]
moznost
moznost
moznost
```



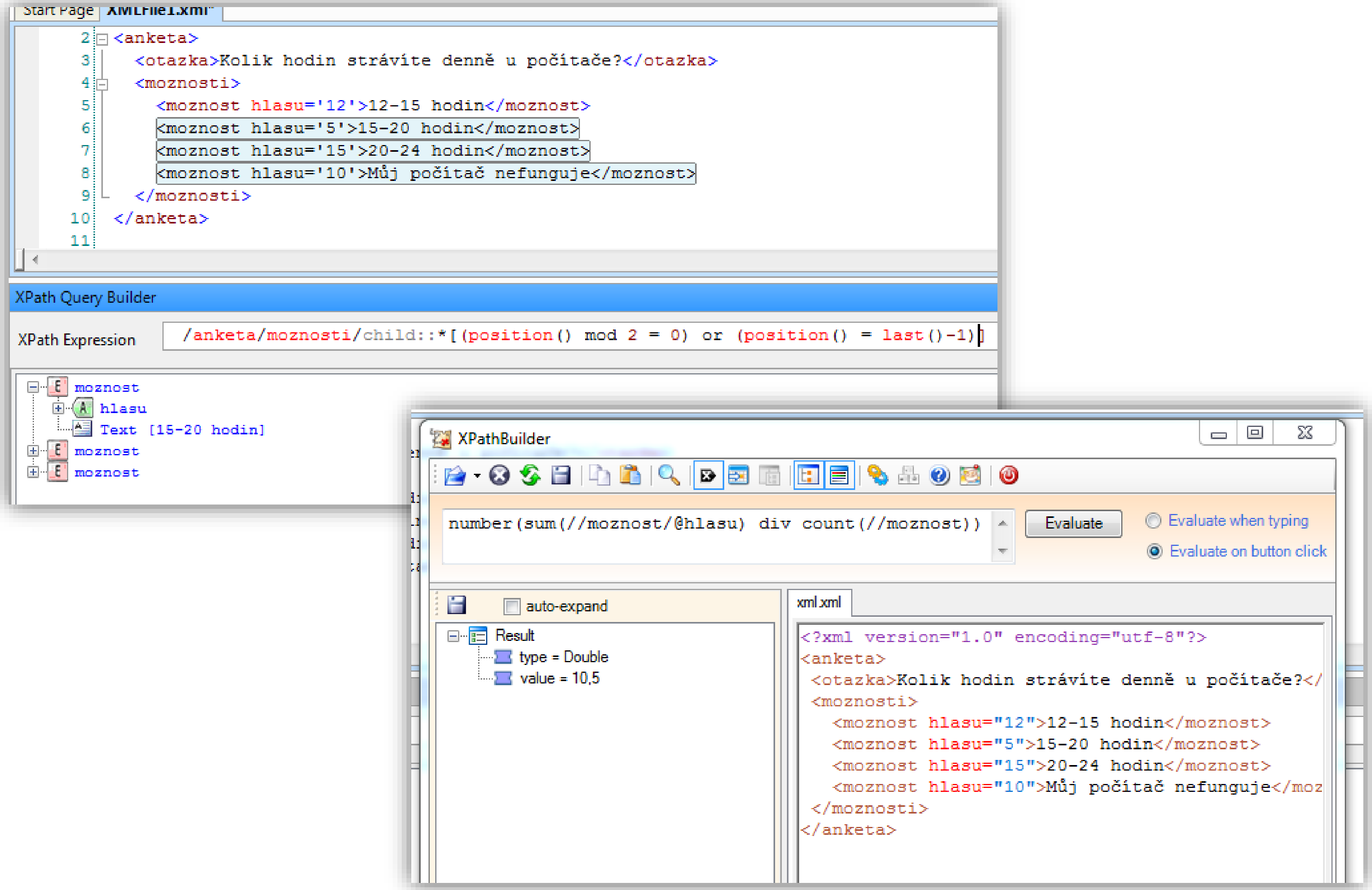
```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
  <otazka>Kolik hodin strávíte denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Query Builder

XPath Expression `//*[count(child::*)>3]`

```
moznosti
  moznost
  moznost
  moznost
  moznost
```

XPATH



```
<anketa>
  <otazka>Kolik hodin strávite denně u počítače?</otazka>
  <moznosti>
    <moznost hlasu='12'>12-15 hodin</moznost>
    <moznost hlasu='5'>15-20 hodin</moznost>
    <moznost hlasu='15'>20-24 hodin</moznost>
    <moznost hlasu='10'>Můj počítač nefunguje</moznost>
  </moznosti>
</anketa>
```

XPath Expression

```
/anketa/moznosti/child::*[(position() mod 2 = 0) or (position() = last()-1)]
```

```
moznost
  hlasu
  Text [15-20 hodin]
moznost
moznost
```

```
number(sum(//moznost/@hlasu) div count(//moznost))
```

( ) Evaluate when typing [✓ Evaluate on button click]

```
Result
  type = Double
  value = 10,5
```

```
<?xml version="1.0" encoding="utf-8"?>
<anketa>
 <otazka>Kolik hodin strávite denně u počítače?</
 <moznosti>
   <moznost hlasu="12">12-15 hodin</moznost>
   <moznost hlasu="5">15-20 hodin</moznost>
   <moznost hlasu="15">20-24 hodin</moznost>
   <moznost hlasu="10">Můj počítač nefunguje</moz
 </moznosti>
</anketa>
```

XPath – Online Testers

- http://www.freeformatter.com/xpath-tester.html
- http://codebeautify.org/Xpath-Tester
- http://www.xpathtester.com/xpath
- Etc…

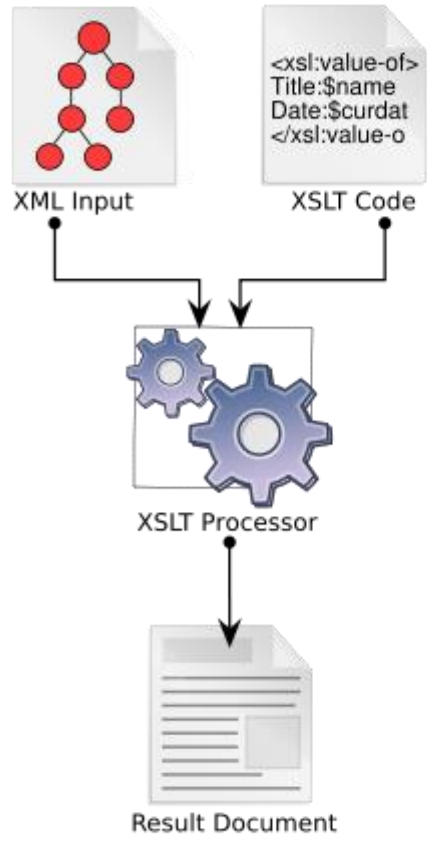
XPATH AND JAVASCRIPT

- Example for modern browsers.

```
function parseXml(xml){
          var path = "/bookstore/book/title";
          var nodes = document.evaluate(path, xml, null, XPathResult.ANY_TYPE, null);
          var result = nodes.iterateNext();
          while (result) {
                    console.log(result.childNodes[0].nodeValue);
                    result = nodes.iterateNext();
          }
}

function loadXMLDoc(fileName) {
          var xhttp = new XMLHttpRequest();
          xhttp.addEventListener("load", function(){
                    parseXml(xhttp.responseXML);
          });
          xhttp.open("GET", fileName);
          xhttp.send(null);
}

loadXMLDoc('test.xml');
```

XSLT

- Extensible Stylesheet Language Transformations
- Language for transforming XML documents into other documents.
- Uses XPath or XQuery.



XSLT

```xml
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
</persons>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/1999/xhtml">

  <xsl:output method="xml" indent="yes"
encoding="UTF-8"/>

  <xsl:template match="/persons">
    <html>
      <head> <title>Testing XML Example</title>
</head>
      <body>
        <h1>Persons</h1>
        <ul>
          <xsl:apply-templates select="person">
            <xsl:sort select="family-name" />
          </xsl:apply-templates>
        </ul>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="person">
    <li>
          <xsl:value-of select="family-name"/>
          <xsl:text>, </xsl:text>
          <xsl:value-of select="name"/>
    </li>
  </xsl:template>

</xsl:stylesheet>
```

```xml
<?xml version="1.0" encoding="UTF-
8"?>
<html
xmlns="http://www.w3.org/1999/xhtml">
  <head> <title>Testing XML
Example</title> </head>
  <body>
    <h1>Persons</h1>
      <ul>
        <li>Ismincius, Morka</li>
        <li>Smith, John</li>
      </ul>
  </body>
</html>
```

JSON

- JavaScript Object Notation
  - Collection of zero or more name/value pairs
  - List of values
  - Data types – JSONString, JSONNumber, JSONBoolean, JSONNull, etc.
- Suitable for exchange and transfer short structured data
  - [{"name": "Cerna sanitka", "tvname": "CT1"}, {"name": "Comeback", "tvname": "Nova"}, {"name": "Pratele", "tvname": "Prima"}]

- http://jsonlint.com/, http://braincast.nl/samples/jsoneditor/

JSON AND JAVASCRIPT

```
function loadJSON()
{
          var data_file = "http://www.tutorialspoint.com/json/data.json";
          var http_request = new XMLHttpRequest();
          http_request.addEventListener('load', function(){
                    // Javascript function JSON.parse to parse JSON data
                    var jsonObj = JSON.parse(http_request.responseText);

                    // jsonObj variable now contains the data structure and can
                    // be accessed as jsonObj.name and jsonObj.country.
                    document.getElementById("Name").innerHTML =  jsonObj.name;
                    document.getElementById("Country").innerHTML = jsonObj.country;
          });
          http_request.open("GET", data_file);
          http_request.send();
}
```

```
$(document).ready(function(){
  $("button").click(function(){

$.getJSON("demo_ajax_json.json",function(result)
{
      $.each(result, function(i, field){
        $("div").append(field + " ");
      });
    });
  });
});
```