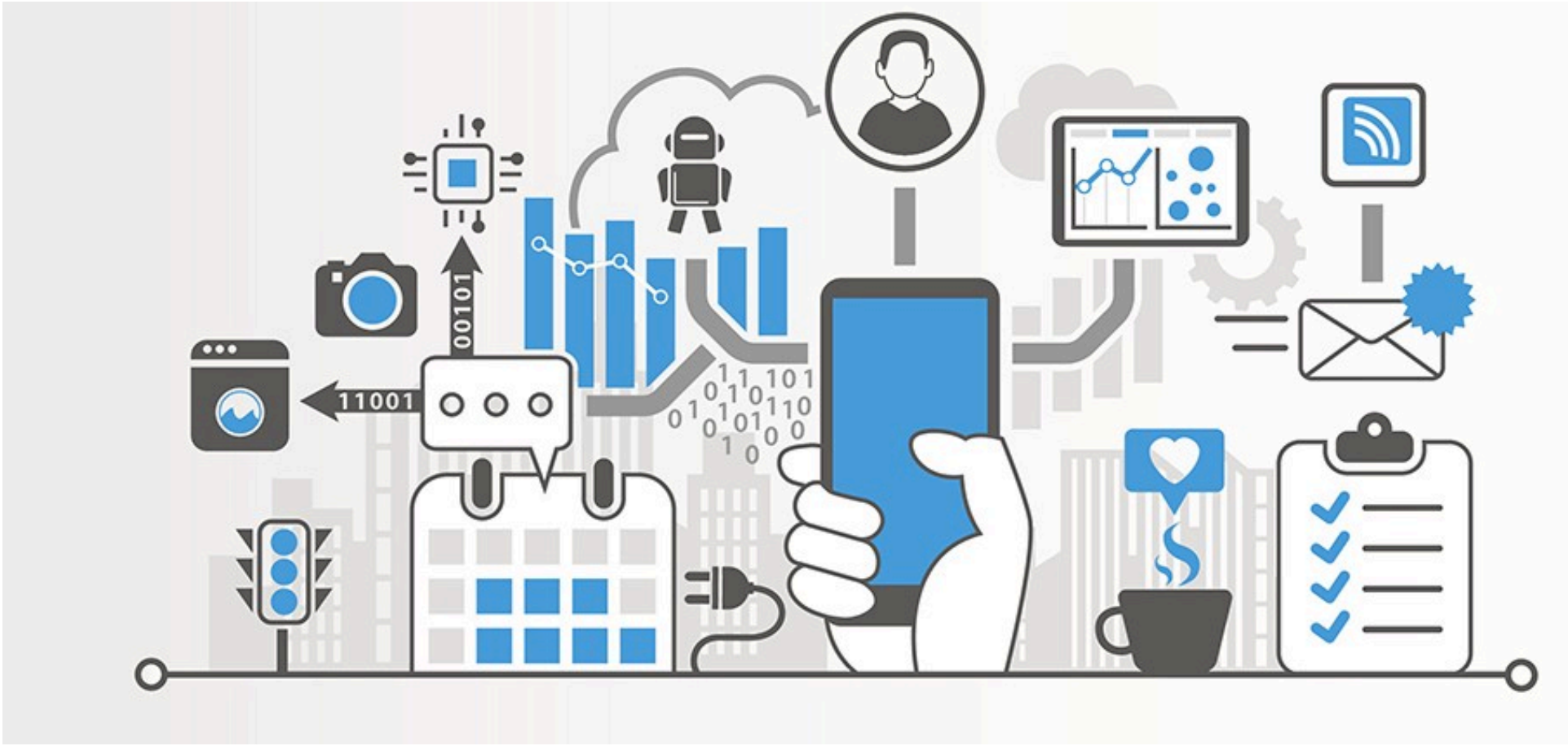
Programming for IoT Applications

Edoardo Patti

Lecture 4



DATA EXCHANGE IN WEB ENVIRONMENTS

Data exchange in web environments

The web consists of several heterogeneous systems that need to exchange data between them.

Requirements for data exchange

- A language to define abstract data types
- The system independent data representation for any abstract data type that can be defined
- The mechanisms that data receivers must use for correctly decoding data
  - Typically, receivers must know the (abstract) type of data they will receive for decoding them

What is a Markup Language?

*A **Markup Language** is a system for annotating a document in a way that is syntactically distinguishable from the text.*

Annotations describe structural presentation and semantic aspects of a document.

What is SGML?

SGML (Standard Generalized Markup Language) is a meta-language for describing markup languages for device/system independent documents

SGML enables the description of syntactic/structural aspects of markup languages (not their semantics)

- How markups are written and how they are distinguished from text and from each other.
- What markups are possible or required in different parts of the document.

What is SGML?

An SGML Document

- is a data object that can be described by the general markup language
- takes the form of a text conformant to the general SGML syntactic rules, including markups, textual data and a reference to a DTD (Document Type Definition)

The DTD is a formalism that describes the specific features of a markup language

HTML – HyperText Markup Language

What is HTML?

HTML is the standard markup language for creating web pages.

Tim Berners-Lee considered **HTML** to be an application of **SGML** when proposed its first draft.

It is maintained by World Wide Web Consortium (W3C)

Tim Berners-Lee

What is HTML?

HTML describes the structure of a website semantically along with cues for presentation.

Web browsers read HTML files and render them into visible or audible web pages.

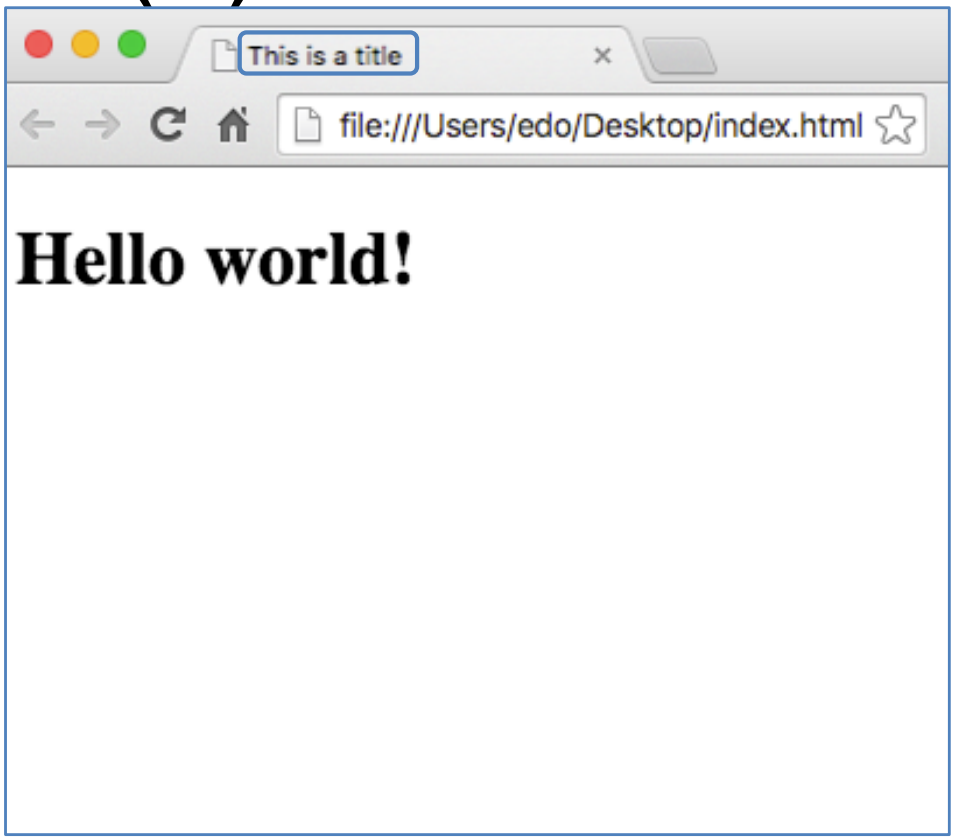
What is HTML?

The language consists of tags enclosed in angle brackets (like <html>), also called HTML elements.

Browsers do not display the HTML tags but use them to interpret the content of the page.

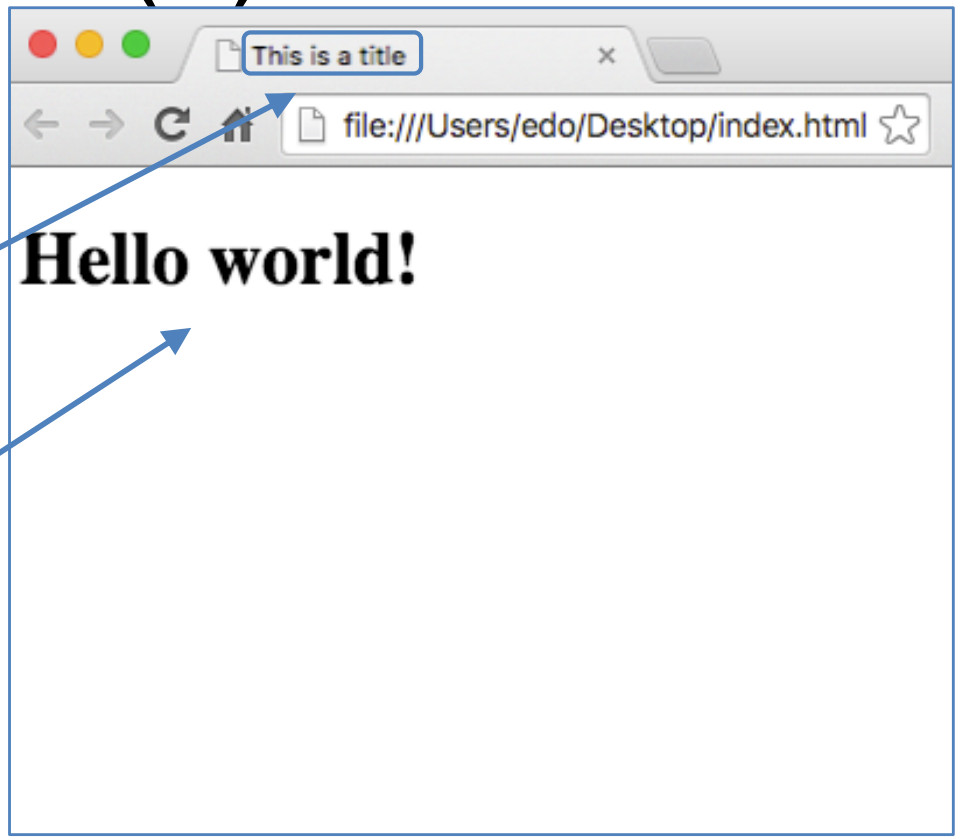
It is a markup language, rather than a programming language.

HTML: example (1)



HTML: example (1)

```
<!DOCTYPE html>
<html>
    <head>
        <title>This is a
title</title>
    </head>
    <body>
        <p>Hello world!</p>
    </body>
</html>
```



HTML: example (2)

HTML: example (2)


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="it-IT" lang="it-IT">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta http-equiv="Content-language" content="it-IT" />
        <title>Politecnico di Torino</title>
        <meta name="author" content="Politecnico di Torino" />
        <meta name="copyright" content="Politecnico di Torino" />
                                <meta name="description" content="Da oltre 150 anni, il Politecnico di Torino &egrave; una delle istituzioni pubbliche
pi&ugrave; prestigiose a livello italiano ed internazionale nella formazione, ricerca, trasferimento tecnologico e servizi in tutti i settori
dell'Architettura e dell'Ingegneria." />                                <meta name="apple-mobile-web-app-title" content="PoliTO" />


        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
                                <link rel="alternate" type="application/rss+xml" title="Polito Feed RSS" href="/news/?rss=1&amp;lang=it" />
<link href="/includes/css/generico.css?updated=201406251040" rel="stylesheet" type="text/css"/>
        <link href="/includes/css/jquery.cookiebar.css?updated=20150529" rel="stylesheet" type="text/css"/>
                    <link href="/includes/css/home.css?updated=201407041100" rel="stylesheet" type="text/css"/>
                                    <script src="/includes/js/jquery-1.9.1.min.js" type="text/javascript"></script>
        <script type="text/javascript">
```

XML - eXtensible Markup Language

What is XML?

XML is a language that enables the formal description of markup languages.

The information is sent in a "document":

- Human & machine readable
- Memory/bandwidth requirements not optimized

Data incorporates information about their type

- receivers do not need to know data types in advance

What is XML?

XML was designed to be:

- directly usable on the internet (via HTTP)
- largely open and compatible
- directly and simply usable by applications

It is one of the main standards for data exchange among heterogeneous (web-) applications

What is XML?

The document includes:

- contents structure (the organization of the contents)
- contents semantics (the meaning of the contents)

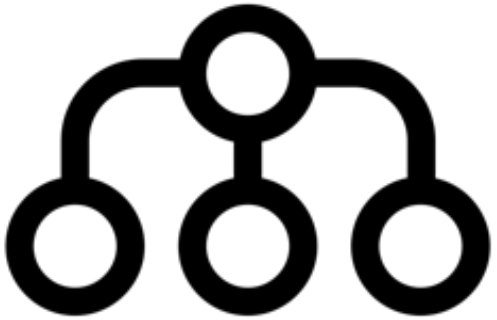
XML: document example


```
<?xml version="1.0"?>
<bibliography>
    <article>
        <author> J. W. Cooley </author>
        <author> J. A. Tukey </author>
        <title> An Algorithm for Machine Computation of Complex FFT </title>
        <journal volume="19" number="April 1965"> Math. Computation </journal>
    </article>
    <article>
        <author> T. G. Stockham </author>
        <title> High speed convolution and correlation </title>
        <proc year="1966"> Spring Joint Computer Conference </proc>
    </article>
    <book>
        <author> D. A. Chappel </author>
        <author> T. Jewell </author>
        <title> Java Web Services </title>
        <publisher> Hops Libri </publisher>
    </book>
</bibliography>
```

Logical Organization of the XML Document

An XML document have tree structure

- Each sub-tree (or node) is an element
- Elements can:
  - include data
  - have attributes



Logical Organization of the XML Document

An XML document may include

- declarations (e.g. reference to DTD)
- processing instructions
- comments

Logical Organization of the XML Document

A well formed XML document follows the general SGML syntax.

Main rules:

- Each non-empty element is delimited by an initial and a final tag

```
<?xml version="1.0"?>
<bibliography>
    <article>
        <author> J. W. Cooley </author>
        <author> J. A. Tukey </author>
        <title>
            An Algorithm for Machine Computation of Complex FFT
        </title>
        <journal volume="19" number="April 1965">
            Math. Computation
        </journal>
    </article>
    <article>
        <author> T. G. Stockham </author>
        <title>
            High speed convolution and correlation
        </title>
        <proc year="1966">
            Spring Joint Computer Conference
        </proc>
    </article>
    <book>
        <author> D. A. Chappel </author>
        <author> T. Jewell </author>
        <title> Java Web Services </title>
        <publisher> Hops Libri </publisher>
    </book>
</bibliography>
```

Logical Organization of the XML Document

A well formed XML document follows the general SGML syntax.

Main rules:

- Each non-empty element is delimited by an initial and a final tag
- There is a single root element (i.e. element that contains all the other elements)

```
<?xml version="1.0"?>
<bibliography>
        <article>
                <author> J. W. Cooley </author>
                <author> J. A. Tukey </author>
                <title>
                        An Algorithm for Machine Computation of
                        Complex FFT
                </title>
                <journal volume="19" number="April 1965">
                        Math. Computation
                </journal>
        </article>
        <article>
                <author> T. G. Stockham </author>
                <title>
                        High speed convolution and correlation
                </title>
                <proc year="1966">
                        Spring Joint Computer Conference
                </proc>
        </article>
        <book>
                <author> D. A. Chappel </author>
                <author> T. Jewell </author>
                <title> Java Web Services </title>
                <publisher> Hops Libri </publisher>
        </book>
</bibliography>
```

Logical Organization of the XML Document

A well formed XML document follows the general SGML syntax.

Main rules:

- Each non-empty element is delimited by an initial and a final tag
- There is a single root element (i.e. element that contains all the other elements)
- Attribute values are always enclosed in quotes

```
<?xml version="1.0"?>
<bibliography>
        <article>
                <author> J. W. Cooley </author>
                <author> J. A. Tukey </author>
                <title>
                        An Algorithm for Machine Computation of
                        Complex FFT
                </title>
                <journal volume="19" number="April 1965">
                        Math. Computation
                </journal>
        </article>
        <article>
                <author> T. G. Stockham </author>
                <title>
                        High speed convolution and correlation
                </title>
                <proc year="1966">
                        Spring Joint Computer Conference
                </proc>
        </article>
        <book>
                <author> D. A. Chappel </author>
                <author> T. Jewell </author>
                <title> Java Web Services </title>
                <publisher> Hops Libri </publisher>
        </book>
</bibliography>
```

Logical Organization of the XML Document

A well formed XML document follows the general SGML syntax.

Main rules:

- Each non-empty element is delimited by an initial and a final tag
- There is a single root element (i.e. element that contains all the other elements)
- Attribute values are always enclosed in quotes
- Attribute names are unique inside each element (XML is case-sensitive)

```
<?xml version="1.0"?>
<bibliography>
        <article>
                <author> J. W. Cooley </author>
                <author> J. A. Tukey </author>
                <title>
                        An Algorithm for Machine Computation of
                        Complex FFT
                </title>
                <journal volume="19" number="April 1965">
                        Math. Computation
                </journal>
        </article>
        <article>
                <author> T. G. Stockham </author>
                <title>
                        High speed convolution and correlation
                </title>
                <proc year="1966">
                        Spring Joint Computer Conference
                </proc>
        </article>
        <book>
                <author> D. A. Chappel </author>
                <author> T. Jewell </author>
                <title> Java Web Services </title>
                <publisher> Hops Libri </publisher>
        </book>
</bibliography>
```

Syntactic Structures

A document is made up of data (character sequences) and markups

A markup can be:

- the begin/end tag of an element
- a reference to an entity
- a comment
- a DTD declaration
- an XML declaration
- a processing instruction

The XML declaration

It is placed at the beginning of the document

It specifies the XML version and character encoding used in the document

Syntax:

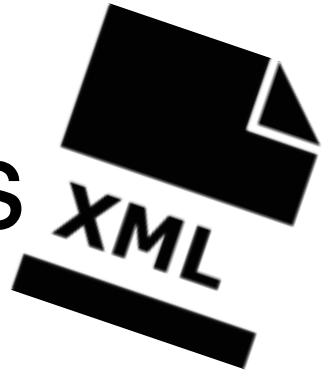
<?xml version="..." encoding="..." ?>

Examples:

<?xml version = "1.0"?>

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

Validating XML Documents



An XML well-formed document is valid if:

- It contains a DTD declaration
- It satisfies the constraints expressed by its DTD

What is DTD?

DTD (Document Type Definition) defines a language.

It defines a sequence of rules (element declarations and attribute declarations)

Rules are written in SGML syntax

DTD example


```
<!-- TV Schedule DTD -->
<!DOCTYPE TVSCHEDULE [

        <!ELEMENT TVSCHEDULE (CHANNEL+)>
        <!ELEMENT CHANNEL (BANNER,DAY+)>
        <!ELEMENT BANNER (#PCDATA)>
        <!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
        <!ELEMENT HOLIDAY (#PCDATA)>
        <!ELEMENT DATE (#PCDATA)>
        <!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
        <!ELEMENT TIME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT DESCRIPTION (#PCDATA)>

        <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
        <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
        <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
        <!ATTLIST TITLE RATING CDATA #IMPLIED>
        <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

DTD: element declaration

DTD specifies:

- Element name
- Content model

The syntax is:

<!ELEMENT *name model*>

Model Types are:

- EMPTY. The element must be empty
- ANY. Any contents are admitted (no check)
- Element. The element must contain only elements
- Mixed Content. The element may contain both elements and data

DTD: element models

Element Model let specify name, order, optionality and multiplicity of nested elements by a simple grammar

```
<!ELEMENT meal (course*)>
<!ELEMENT course (first|second|dessert)>
<!ELEMENT fixedPrficeMeal (first,second,dessert)>
<!ELEMENT first EMPTY>
<!ELEMENT second EMPTY>
<!ELEMENT dessert EMPTY>
```

DTD: element models

Element Model let specify name, order, optionality and multiplicity of nested elements by a simple grammar

A model is either an **element** or **sequence of models** or **alternative of models**:

- A comma denotes a sequence

```
<!ELEMENT meal (course*)>
<!ELEMENT course (first|second|dessert)>
<!ELEMENT fixedPrficeMeal (first,second,dessert)>
<!ELEMENT first EMPTY>
<!ELEMENT second EMPTY>
<!ELEMENT dessert EMPTY>
```

DTD: element models

Element Model let specify name, order, optionality and multiplicity of nested elements by a simple grammar

A model is either an **element** or **sequence of models** or **alternative of models**:

- A comma denotes a sequence
- A vertical bar denotes an alternative

```
<!ELEMENT meal (course*)>
<!ELEMENT course (first|second|dessert)>
<!ELEMENT fixedPrficeMeal (first,second,dessert)>
<!ELEMENT first EMPTY>
<!ELEMENT second EMPTY>
<!ELEMENT dessert EMPTY>
```

DTD: element models

Multiplicity is specified by the postfix operators:

- `+` the model must occur 1 or more times
- `*` the model must occur 0 or more times
- `?` the model must occur 0 or 1 times (is optional)

```
<!ELEMENT meal (course*)>
<!ELEMENT course (first|second|dessert)>
<!ELEMENT fixedPrficeMeal (first,second,dessert)>
<!ELEMENT first EMPTY>
<!ELEMENT second EMPTY>
<!ELEMENT dessert EMPTY>
```

```
<!ELEMENT laboratory (name, head, secretary?,
                      (technician|operator)+)>
```

DTD: attribute declaration

Each declaration specifies the features of one or more attributes of an element type

The syntax is:

<!ATTLIST element-name attribute-name attribute-type attribute-value>

For each attribute, **name, value type and default declaration** can be specified.

<!ATTLIST onlyData id ID #REQUIRED type (vector | matrix) "vector">

DTD: value type specification

Type	Value	Syntax	Example
String	A string without the special characters < >&’”	CDATA	CDATA
Token	a token or a sequence of token	ID IDREF ENTITY NMTOKEN IDREFS ENTITIES NMTOKENS	ID
Enumaration	One of the specified strings	A list of strings separated by |	(Mr|Mrs|Miss)

DTD: meaning of Token Types

Type	Value
ID	A name that **uniquely** identifies the element in the whole XML document
IDREF	The ID of an element in the XML document (a reference to an element)
ENTITY	The name of an entity declared in the DTD
NMTOKEN	A generic name
IDREFS	A sequence of IDREF tokens
ENTITIES	A sequence of ENTITY tokens
NMTOKENS	A sequence of NMTOKEN tokens

DTD: Default declaration

Indicates if the attribute is compulsory or what is its default value. It may take 4 different forms:

- #REQUIRED. The attribute is compulsory (no default)
- “default ”. The attribute is optional. If absent, the indicated default value is used.
- #IMPLIED. The attribute is optional, and the default value is undefined. Any value can be used if the attribute is absent.
- #FIXED default. The attribute is optional, but fixed. If present, it must have the indicated default value.

DTD: example


```
<!ATTLIST course
                    code        ID          #REQUIRED
                    name        CDATA       #IMPLIED
                    double      (yes I no)  “no”
>
```

XML examples of valid start element tags:

<course code=“A10” name=“Spaghetti”>

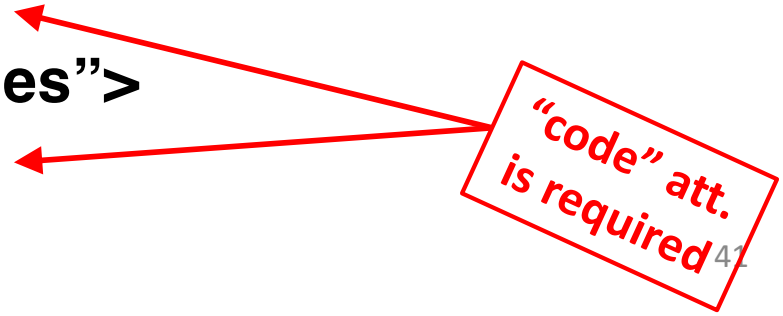
<course code=“A10” double=“yes”>

XML examples of invalid start element tags:

<course name=“Steak”>

<course code=“A10” double=“nes”>

<course>



DTD: example


```
<!ATTLIST course
                    code        ID          #REQUIRED
                    name        CDATA       #IMPLIED
                    double      (yes | no)  “no”
>
```

XML examples of valid start element tags:

<course code=“A10” name=“Spaghetti”>

<course code=“A10” double=“yes”>

XML examples of invalid start element tags:

<course name=“Steak”>

<course code=“A10” double=“nes”>

<course>

JSON - JavaScript Object Notation


JSON
JSON EVERYWHERE
quickmeme.com

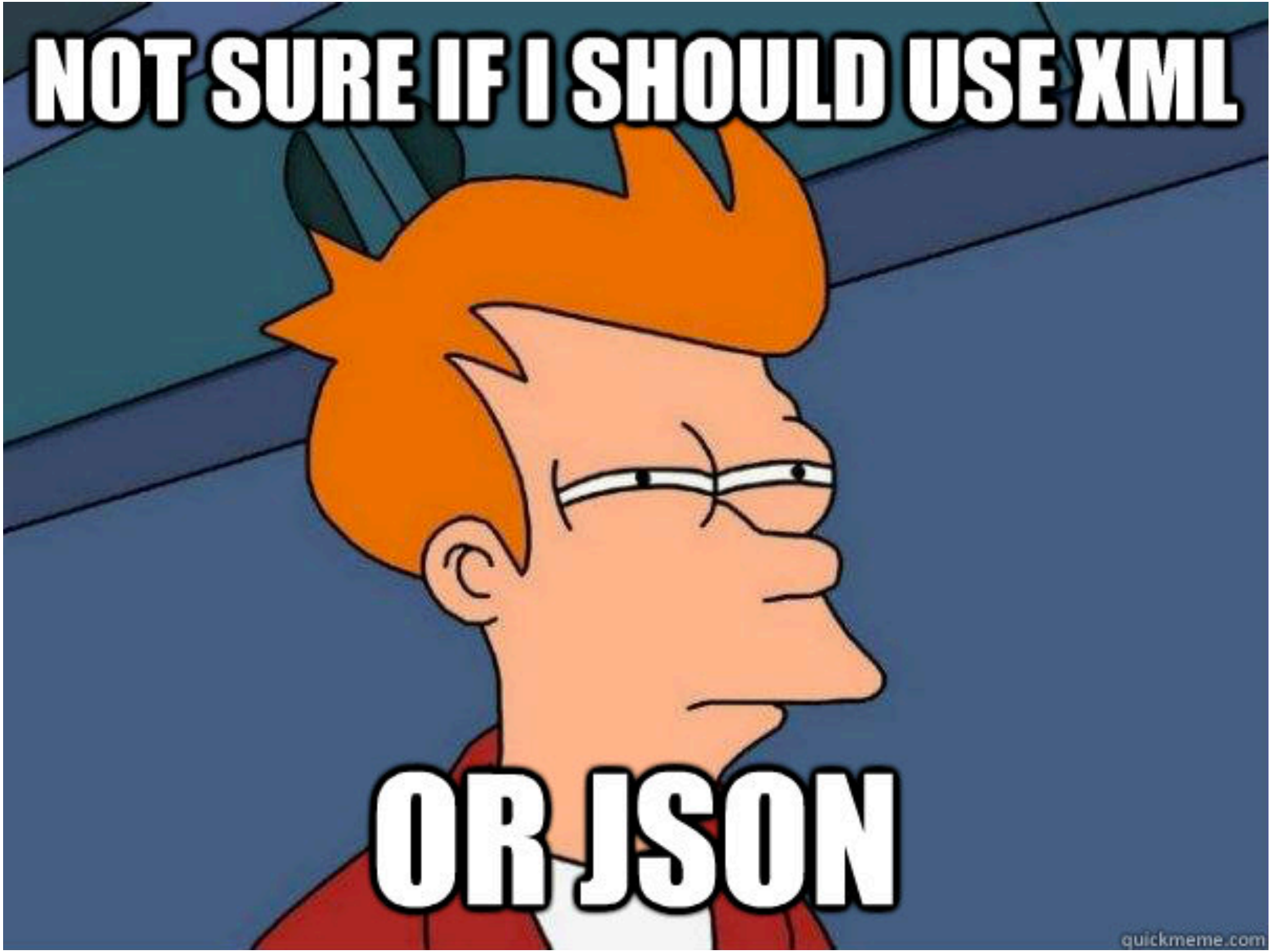
What is JSON?

JSON is a lightweight text based data exchange format

It is completely language independent

It is easy to understand, manipulate and generate

It is NOT a markup language


NOT SURE IF I SHOULD USE XML
OR JSON
quickmeme.com

JSON vs. XML

Just like XML:

- Plain text formats
- Self-describing (human readable)
- Hierarchical (values can contain lists of objects or single values)

JSON vs. XML

Differently from XML:

- Lighter and faster than XML
- JSON uses typed objects. Whilst, all XML values are type-less strings and must be parsed at runtime
- Less syntax, no semantics

Object syntax

- Unordered sets of key/value pairs

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true
}
```

Object syntax

- Unordered sets of key/value pairs
- Begins with { (left brace)
- Ends with } (right brace)

```
{
  "name": "Tony Stark",
  "codename": "Iron Man",
  "date_of_birth": "29/05/1970",
  "age": 45,
  "height": 185.42,
  "birthplace": "Long Island, New York",
  "address": null,
  "isAvenger": true
}
```

Object syntax

- Unordered sets of key/value pairs
- Begins with **{** (left brace)
- Ends with **}** (right brace)
- Each key is followed by **:** (colon)

```
{
   "name" : "Tony Stark",
   "codename" : "Iron Man",
   "date_of_birth" : "29/05/1970",
   "age" : 45,
   "height" : 185.42,
   "birthplace" : "Long Island, New York",
   "address" : null,
   "isAvenger" : true
}
```

Object syntax

- Unordered sets of key/value pairs
- Begins with **{** (left brace)
- Ends with **}** (right brace)
- Each key is followed by **:** (colon)
- Key/value pairs are separated by **,** (comma)

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true
}
```

WARNING: No comma for last name/value pair

Arrays syntax

- An ordered collection of values or objects

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008,2010,2012,2013,2015] ,
   "list_of_enemies": [
     {"name": "Iron Monger", "terminated": false},
     {"name": "Mandarin", "terminated": false}]
}
```

Values

Arrays syntax

- An ordered collection of values or objects

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008,2010,2012,2013,2015] ,
   "list_of_enemies": [
      {"name": "Iron Monger", "terminated": false},
      {"name": "Mandarin", "terminated": false}]
}
```

Values

Objects

Arrays syntax

- An ordered collection of values or objects
- Begins with **[** (left bracket)
- Ends with **]** (right bracket)

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008,2010,2012,2013,2015],
   "list_of_enemies": [
     {"name": "Iron Monger", "terminated": false},
     {"name": "Mandarin", "terminated": false}]
}
```

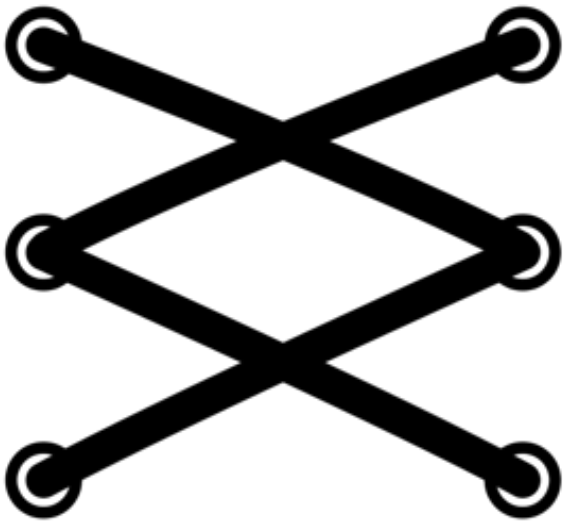
Arrays syntax

- An ordered collection of values or objects
- Begins with **[** (left bracket)
- Ends with **]** (right bracket)
- Values are comma separated

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008,2010,2012,2013,2015],
   "list_of_enemies": [
     {"name": "Iron Monger", "terminated": false},
     {"name": "Mandarin", "terminated": false}]
}
```

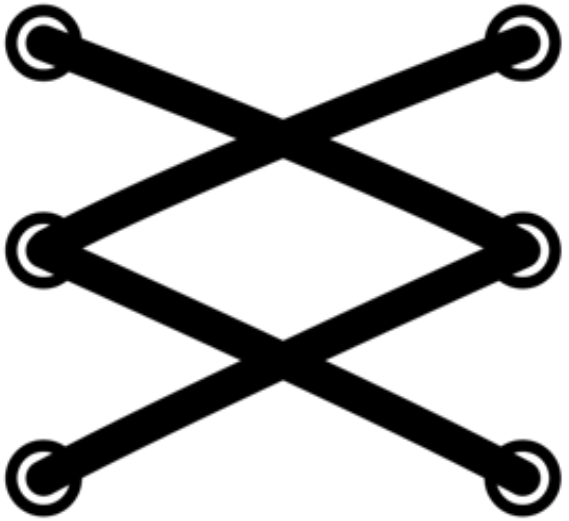
Data Types: Strings

- Sequence of one or more Unicode characters
- Wrapper in “double quotes”
- Backslash escapement



Data Types: Strings

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008,2010,2012,2013,2015],
   "list_of_enemies": [
     {"name": "Iron Monger", "terminated": false},
     {"name": "Mandarin", "terminated": false}]
}
```



Data Types: Booleans and Null

- Booleans: true or false
- Null: nothing or no value.

Data Types: Booleans and Null

```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008,2010,2012,2013,2015] ,
   "list_of_enemies": [
     {"name": "Iron Monger", "terminated": false},
     {"name": "Mandarin", "terminated": false}]
}
```

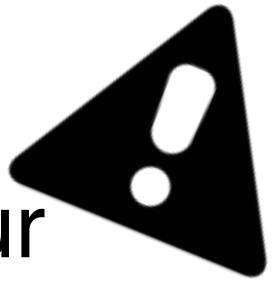
Data Types: Numbers

- Integer ✓
- Real ✓
- Scientific ✓
- No octal or hex ✗
- No NaN or Infinity ✗
  - (Use **null** instead)

Data Types: Numbers

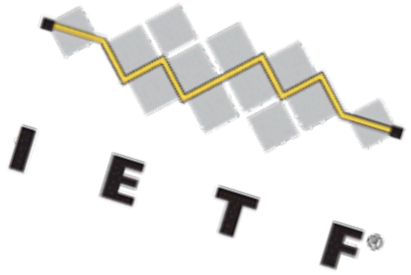
```
{
   "name": "Tony Stark",
   "codename": "Iron Man",
   "date_of_birth": "29/05/1970",
   "age": 45,
   "height": 185.42,
   "birthplace": "Long Island, New York",
   "address": null,
   "isAvenger": true,
   "list_of_allies": ["Thor","Hulk","Capt. America"],
   "movies_release": [2008, 2010, 2012, 2013, 2015] ,
   "list_of_enemies": [
     {"name": "Iron Monger", "terminated": false},
     {"name": "Mandarin", "terminated": false}]
}
```

WARNING



- No inherit validation (XML has DTD)
- Use http://jsonlint.com/ to validate your JSON

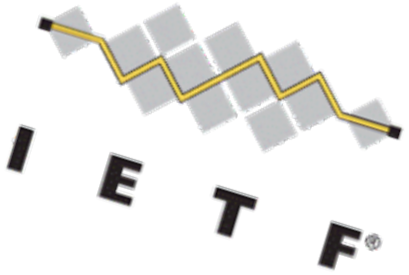
SenML Dataformat



SenML (Sensor Markup Language) is a media type for representing simple sensor measurements and device parameters.

Representations are available for **XML**, EXI and **JSON**. SenML is defined (in draft) by **Internet Engineering Task Force** - IETF

SenML Dataformat

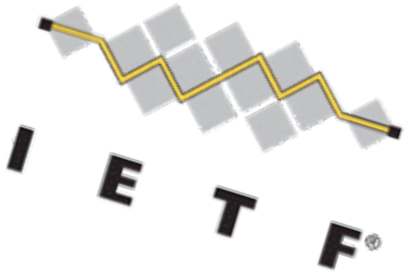


Every valid SenML document is a JSON object with

- an optional **base name (bn)** (e.g. device ID) to be used as a prefix for all resource names
- an array of **event** objects **(e)**

```
{
    "bn": "http://example.org/sensor1/",
    "e": [{ "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 } ]
}
```

SenML Dataformat



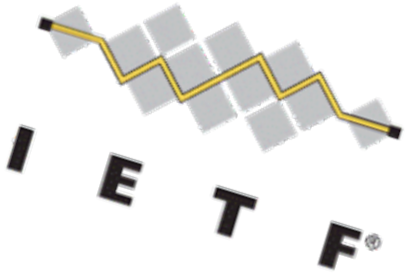
Every valid SenML document is a JSON object with

- an optional **base name (bn)** (e.g. device ID) to be used as a prefix for all resource names
- an array of **event** objects **(e)**

```
{
    "bn": "http://example.org/sensor1/",
    "e": [{ "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 } ]
}
```

Each event object contains a **resource name (n)**, a **value (v)**, a **timestamp (t)** in Unix Timestamp (seconds since 01/01/1970) and a **units (u)**.

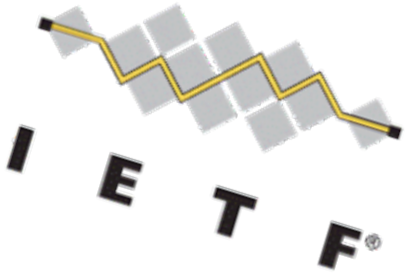
SenML Dataformat



Documents may contain single timestamped events…

```
{
      "bn": "http://example.org/sensor1/",
      "e": [
            { "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 } ]
}
```

SenML Dataformat



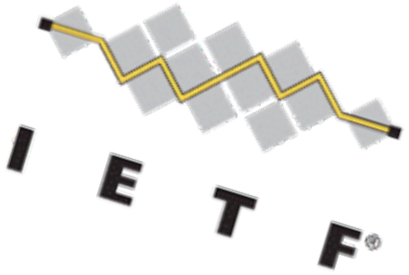
Documents may contain single timestamped events…

```
{
      "bn": "http://example.org/sensor1/",
      "e": [
            { "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 } ]
}
```

…or multiple timestamped events for the same resource…

```
{
      "bn": "http://example.org/sensor1/",
      "e": [
            { "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 },
            { "n": "temperature", "u": "Cel", "t": 1235, "v":22.8 },
            { "n": "temperature", "u": "Cel", "t": 1236, "v":22.2 } ]
}
```

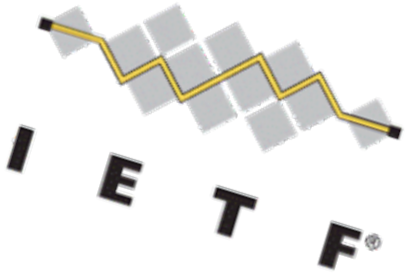
SenML Dataformat



…or, events for multiple resources

```
{
      "bn": "http://example.org/sensor1/",
      "e": [
            { "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 },
            { "n": "light", "u": "lm", "t": 1235, "v":135 },
            { "n": "acidity", "u": "pH", "t": 1236, "v":7 } ]
}
```

SenML Dataformat



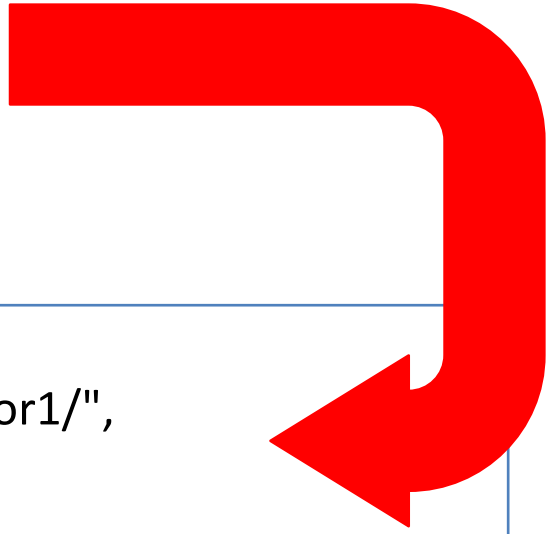
To shorten documents, an optional **base time (bt)** may be given. All times **(t)** are deltas from **bt**. The following two documents are equivalent.

```
{
      "bn": "http://example.org/sensor1/",
      "e": [
            { "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 },
            { "n": "light", "u": "lm", "t": 1235, "v":135 },
            { "n": "acidity", "u": "pH", "t": 1236, "v":7 } ]
}
```

SenML Dataformat

To shorten documents, an optional **base time (bt)** may be given. All times **(t)** are deltas from **bt**. The following two documents are equivalent.

```
{
      "bn": "http://example.org/sensor1/",
      "e": [
            { "n": "temperature", "u": "Cel", "t": 1234, "v":22.5 },
            { "n": "light", "u": "lm", "t": 1235, "v":135 },
            { "n": "acidity", "u": "pH", "t": 1236, "v":7 } ]
}
```



```
{
      "bn": "http://example.org/sensor1/",
      "bt": 1234,
      "e": [
            { "n": "temperature", "u": "Cel", "t": 0, "v":22.5 },
            { "n": "light", "u": "lm", "t": 1, "v":135 },
            { "n": "acidity", "u": "pH", "t": 2, "v":7 } ]
}
```

References

- http://www.w3schools.com/xml/default.asp
- http://www.w3schools.com/json/
- http://wiki.1248.io/doku.php?id=senml
- https://tools.ietf.org/html/draft-jennings-senml-10