

NDBI040: Big Data Management and NoSQL Databases

<http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-1-NDBI040/>

Lecture 5

Data Formats

Martin Svoboda

svoboda@ksi.mff.cuni.cz

1. 11. 2016

Charles University in Prague, Faculty of Mathematics and Physics

Czech Technical University in Prague, Faculty of Electrical Engineering

Lecture Outline

Data formats

- XML – Extensible Markup Language
- JSON – JavaScript Object Notation
- BSON – Binary JSON
- RDF – Resource Description Framework
- CSV – Comma-Separated Values
- Protocol Buffers

XML

Extensible Markup Language

Introduction

XML = *Extensible Markup Language*

- **Representation and interchange of semi-structured data**
 - + a family of related technologies, languages, specifications, ...
- Derived from **SGML**, developed by **W3C**, started in 1996
- Design goals
 - Simplicity, generality and usability across the Internet
- File extension: ***.xml**, content type: `text/xml`
- Versions: 1.0 and 1.1
- W3C recommendation
 - <http://www.w3.org/TR/xml11/>
- XML formats = particular languages
 - XSD, XSLT, XHTML, DocBook, ePub, SVG, RSS, SOAP, ...

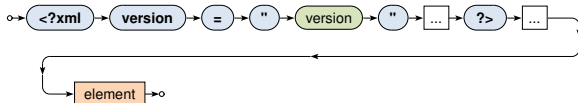
Example

```
<?xml version="1.1" encoding="UTF-8"?>
<movie year="2007">
  <title>Medvídek</title>
  <actors>
    <actor>
      <firstname>Jiří</firstname>
      <lastname>Macháček</lastname>
    </actor>
    <actor>
      <firstname>Ivan</firstname>
      <lastname>Trojan</lastname>
    </actor>
  </actors>
  <director>
    <firstname>Jan</firstname>
    <lastname>Hřebejk</lastname>
  </director>
</movie>
```

Document Structure

Document

- **Prolog**: XML version + some other stuff
- Exactly one **root element**
 - Contains other nested elements and/or other content



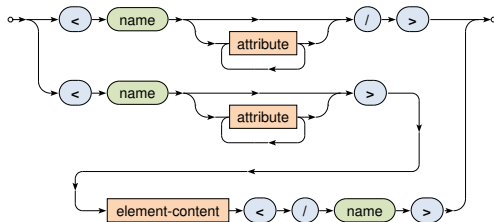
Example

```
<?xml version="1.1" encoding="UTF-8"?>
<movie>
  ...
</movie>
```

Constructs

Element

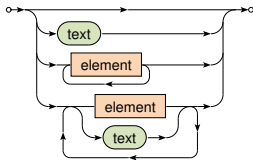
- Marked using **opening and closing tags**
 - ... or an abbreviated tag in case of empty elements
- Each element can be associated with a **set of attributes**
- **Well-formedness** is required



Constructs

Types of element content

- **Empty** content
- **Text** content
- **Element** content
 - Sequence of nested elements
- **Mixed** content
 - Elements arbitrarily interleaved with text



Constructs

Attribute

- Name-value pair



Escaping sequences (predefined entities)

- Used within values of attributes or text content of elements
- E.g.: `<` for `<`, `>` for `>`, `"` for `"`, ...

All available XML constructs

- Basic: `element`, `attribute`, `text`
- Other: `comment`, `processing instruction`, ...

JSON

JavaScript Object Notation

Introduction

JSON = *JavaScript Object Notation*

- **Open standard for data interchange**
- Design goals
 - **Simplicity:** text-based, easy to read and write
 - **Universality:** **object** and **array** data structures
 - Supported by majority of modern programming languages
- Derived from JavaScript (but language independent)
- Started in 2002
- File extension: ***.json**
- Content type: **application/json**
- <http://www.json.org/>

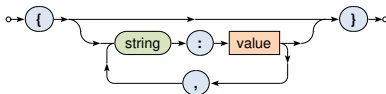
Example

```
{
  "title" : "Medvídek",
  "year" : 2007,
  "actors" : [
    {
      "firstname" : "Jiří",
      "lastname" : "Macháček"
    },
    {
      "firstname" : "Ivan",
      "lastname" : "Trojan"
    }
  ],
  "director" : {
    "firstname" : "Jan",
    "lastname" : "Hřebejk"
  }
}
```

Data Structure

Object

- **Unordered collection of name-value pairs (properties)**
 - Correspond to structures such as objects, records, structs, dictionaries, hash tables, keyed lists, associative arrays, ...



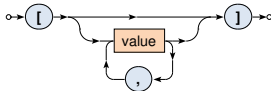
Examples

- { "name" : "Ivan Trojan", "year" : 1964 }
- { }

Data Structure

Array

- **Ordered collection of values**
 - Correspond to structures such as arrays, vectors, lists, sequences, ...
- Values can be of different types, duplicate values are allowed



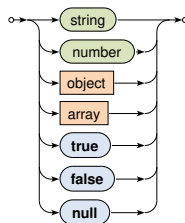
Examples

- [2, 7, 7, 5]
- ["Ivan Trojan", 1964, -5.6]
- []

Data Structure

Value

- Unicode **string**
 - Enclosed with double quotes
 - Backslash escaping sequences
 - Example: `"a \n b \" c \\ d"`
- **Number**
 - Decimal integers or floats
 - Examples: `1`, `-0.5`, `1.5e3`
- **Nested object**
- **Nested array**
- **Boolean** value: `true`, `false`
- Missing information: `null`



BSON

Binary JSON

Introduction

BSON = *Binary JSON*

- **Binary-encoded serialization of JSON documents**
 - Extends the set of basic data types of values offered by JSON (such as a string, ...) with a few new specific ones
- Design characteristics: **lightweight, traversable, efficient**
- Used by **MongoDB**
 - Document NoSQL database for JSON documents
 - Data storage and network transfer format
- File extension: ***.bson**
- <http://bsonspec.org/>

Example

JSON

```
{  
  "title" : "Medvídek",  
  "year" : 2007  
}
```

BSON

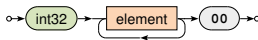
```
24 00 00 00  
02 74 69 74 6C 65 00 0A 00 00 00 4D 65 64 76 C3 AD 64 65 6B 00  
10 79 65 61 72 00 D7 07 00 00  
00
```

Document Structure

Document = serialization of **one JSON object or array**

- JSON object is serialized directly
- JSON array is first transformed to a JSON object
 - Property names correspond to position numbers
 - E.g.:

```
[ "Trojan", "Svěrák" ] →  
{ "0" : "Trojan", "1" : "Svěrák" }
```
- Structure
 - **Document size** (total number of bytes)
 - Sequence of **elements**
 - Terminating hexadecimal 00 byte



Document Structure

Element = serialization of **one JSON property**

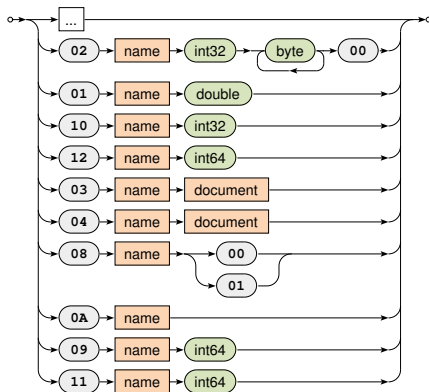
- Structure
 - **Type selector**
 - 02 (**string**)
 - 01 (double), 10 (32-bit **integer**), 12 (64-bit integer)
 - 03 (**object**), 04 (**array**)
 - 08 (**boolean**)
 - 0A (**null**)
 - 09 (datetime), 11 (timestamp)
 - ...
 - **Property name**
 - Unicode string terminated by 00



- **Property value**

Document Structure

Element = serialization of **one JSON property**



RDF

Resource Description Framework

Introduction

RDF = *Resource Description Framework*

- Language for **representing information about resources** in the World Wide Web
 - + a family of related technologies, languages, specifications, ...
 - Used in the context of the **Semantic Web, Linked Data, ...**
- Developed by **W3C**
- Started in 1997
- Versions: 1.0 and 1.1
- W3C recommendations
 - <https://www.w3.org/TR/rdf11-concepts/>
 - Concepts and Abstract Syntax
 - <https://www.w3.org/TR/rdf11-mt/>
 - Semantics

Statements

Resource

- Any real-world entity
 - **Referents** = **resources identified by IRI**
 - E.g. physical things, documents, abstract concepts, ...
 - **Values** = **resources for literals**
 - E.g. numbers, strings, ...

Statement about resources = one RDF **triple**

- Three components: **subject**, **predicate**, and **object**

Examples

```
<http://db.cz/movies/medvidek>  
<http://db.cz/terms#actor>  
<http://db.cz/actors/trojan> .
```

```
<http://db.cz/movies/medvidek>  
<http://db.cz/terms#year>  
"2007" .
```

Statements

Triple components

- **Subject**
 - Describes the resource the given statement is about
 - IRI or blank node identifier
- **Predicate**
 - Describes the property or characteristic of the subject
 - IRI
- **Object**
 - Describes the value of that property
 - IRI or blank node identifier or literal

Although triples are inspired by natural languages, they have nothing to do with processing of natural languages

Identifiers and Literals

IRI = *Internationalized Resource Identifier*

- Absolute (not relative) IRIs with optional fragment identifiers
- RFC 3987
- Unicode characters
- Examples
 - `http://db.cz/movies/medvidek`
 - `http://db.cz/terms#actor`
 - `mailto:svoboda@ksi.mff.cuni.cz`
 - `urn:issn:0167-6423`
- **URLs are often used** in practice → information about given resources can then be published / retrieved via HTTP

Identifiers and Literals

Blank node identifiers

- **Blank nodes** (anonymous resources)
 - Allow to express statements about resources without explicitly naming (identifying) them
- Their identifiers have only **local scope of validity** (e.g. within a given file),
syntax depends on a particular serialization format
- E.g.: `_:node18`

Identifiers and Literals

Literals

- Plain values
 - E.g.: `"Medvídek"`, `"2007"`
- Typed values
 - E.g.: `"Medvídek"^^xs:string`, `"2007"^^xs:integer`
- Strings with language tags
 - E.g.: `"Medvídek"@cs`
- *Types and language tags cannot be mutually combined*

Data Model

Directed labeled multigraph

- Vertices
 - **One vertex for each IRI or literal value**
- Edges
 - **One edge for each individual triple**
 - Edges are directed $subject \xrightarrow{predicate} object$
 - Property names (predicate IRIs) are used as edge labels

Example

```
<http://db.cz/movies/medvidek>
  <http://db.cz/terms#actor> <http://db.cz/actors/machacek> .

<http://db.cz/movies/medvidek>
  <http://db.cz/terms#actor> <http://db.cz/actors/trojan> .

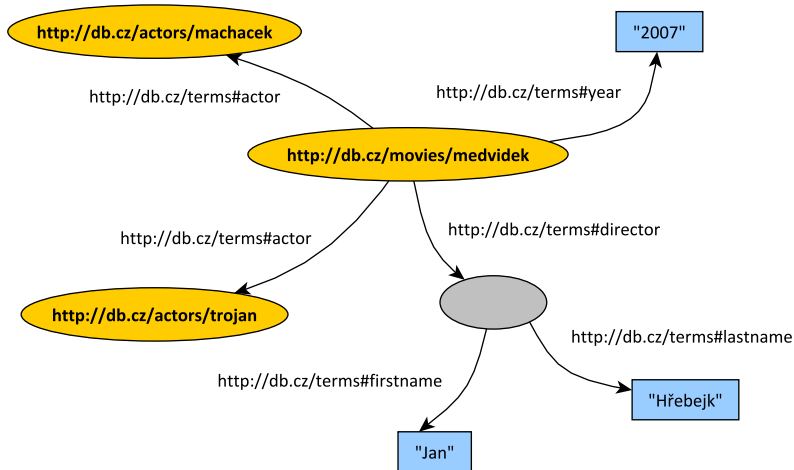
<http://db.cz/movies/medvidek>
  <http://db.cz/terms#year> "2007" .

<http://db.cz/movies/medvidek>
  <http://db.cz/terms#director> _:n18 .

_:n18
  <http://db.cz/terms#firstname> "Jan" .

_:n18
  <http://db.cz/terms#lastname> "Hřebejk" .
```

Example



Serialization

Available approaches

- **N-Triples** notation
 - <https://www.w3.org/TR/n-triples/>
- **Turtle** notation (*Terse RDF Triple Language*)
 - <https://www.w3.org/TR/turtle/>
- **RDF/XML** notation
 - XML syntax for RDF
 - <https://www.w3.org/TR/rdf-syntax-grammar/>
- **JSON-LD** notation
 - JSON-based serialization for Linked Data
 - <https://www.w3.org/TR/json-ld/>
- ...

N-Triples Notation

RDF **N-Triples** notation = *A line-based syntax for an RDF graph*

- Simple, line-based, plain text format
- File extension: ***.rdf**
- <https://www.w3.org/TR/n-triples/>

Example

- *Already presented...*

Document

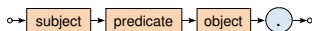
- Statements are terminated by dots, delimited by EOL



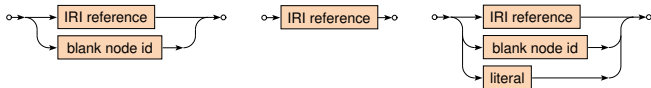
N-Triples Notation

Statement

- Individual triple components are delimited by spaces



Triple components: subject, predicate, object



N-Triples Notation

IRI reference

- IRIs are enclosed in angle brackets

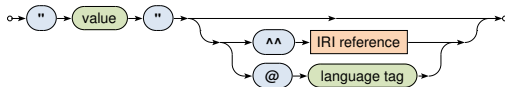


Blank node identifier



Literal

- Literals are enclosed in double quotes



Turtle Notation

Turtle = *Terse RDF Triple Language*

- Compact text format, various abbreviations for common usage patterns
- File extension: ***.ttl**
- Content type: text/turtle
- <https://www.w3.org/TR/turtle/>

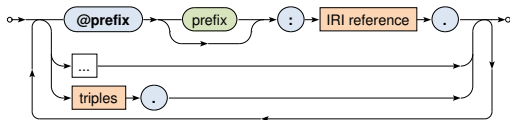
Example

```
@prefix i: <http://db.cz/terms#> .
@prefix m: <http://db.cz/movies/> .
@prefix a: <http://db.cz/actors/> .
m:medvidek
  i:actor a:machacek , a:trojan ;
  i:year "2007" ;
  i:director [ i:firstname "Jan" ; i:lastname "Hřebejk" ] .
```

Turtle Notation

Document

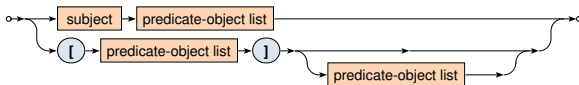
- Contains a **sequence of triples and/or declarations**
- Prefix declarations
 - **Prefixed names can be used instead for full IRI references**
- Groups of triples
 - **Triples sharing the same subject and object or at least the same subject can be *grouped* together**
- Individual groups of triples are terminated by dots



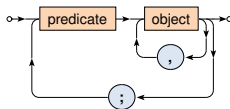
Turtle Notation

Statement

- Blank nodes can be defined using brackets

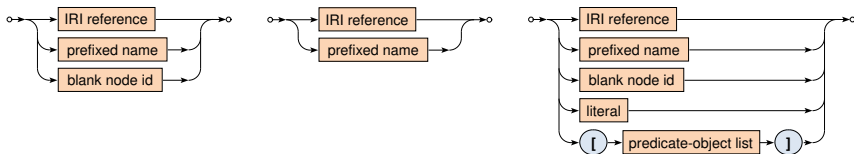


Predicate-object list / object list

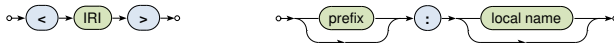


Turtle Notation

Triple components: subject, predicate, object



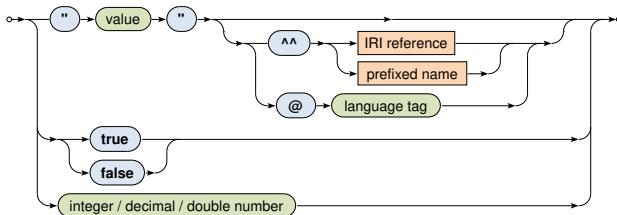
IRI reference / prefixed name



Turtle Notation

Literal

- Traditional literals
+ new abbreviated forms of numeric and boolean literals



Example

Example revisited

```
@prefix i: <http://db.cz/terms#> .  
@prefix m: <http://db.cz/movies/> .  
@prefix a: <http://db.cz/actors/> .  
m:medvidek  
  i:actor a:machacek , a:trojan ;  
  i:year "2007" ;  
  i:director [ i:firstname "Jan" ; i:lastname "Hřebejk" ] .
```

CSV

Comma-Separated Values

Introduction

CSV = *Comma-Separated Values*

- Unfortunately **not fully standardized**
 - Different field separators (commas, semicolons)
 - Different escaping sequences
 - No encoding information
- RFC 4180, RFC 7111
- File extension: ***.csv**
- Content type: **text/csv**

Example

```
firstname,lastname,year
```

```
Ivan,Trojan,1964
```

```
Jiří,Macháček,1966
```

```
Jitka,Schneiderová,1973
```

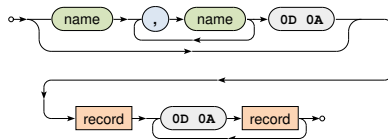
```
Zdeněk,Svěrák,1936
```

```
Anna,Geislerová,1976
```

Document Structure

Document

- Optional **header** + list of **records**



Record

- Comma separated list of **fields**



Protocol Buffers

Introduction

Protocol Buffers

- **Extensible mechanism for serializing structured data**
 - Used in communication protocols, data storage, ...
- Design goals
 - Language-neutral, platform-neutral
 - **Small, fast, simple**
- Developed (and widely used) by **Google**
- Started in 2008 internally and 2011 publicly
- Versions: **proto2**, **proto3**
- File extension: ***.proto**
- <https://developers.google.com/protocol-buffers/>

Introduction

Intended usage

- Schema creation → automatic source code generation → sending messages between applications

Components

- **Interface description language**
- **Source code generator** (protoc compiler)
 - Supported languages
 - Official: C++, C#, Java, Python, Ruby ...
 - 3rd party: Perl, PHP, Scala, ...
- **Binary serialization format**
 - Compact, not self-describing

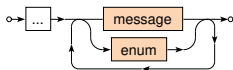
Example

```
syntax = "proto3";  
message Actor {  
    string firstname = 1;  
    string lastname = 2;  
}  
message Movie {  
    string title = 1;  
    int32 year = 16;  
    repeated Actor actors = 17;  
    enum Genre {  
        UNKNOWN = 0;  
        COMEDY = 1;  
        FAMILY = 2;  
    }  
    repeated Genre genres = 2048;  
}
```

Schema Structure

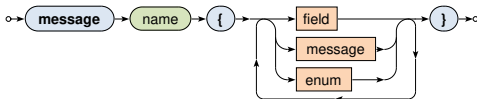
Schema

- One schema may contain multiple **message descriptions**
- Other constructs are allowed as well, e.g. **enumerations**



Message

- Represents a small logical record of information
- Defines a **set of uniquely numbered fields**
- Nested messages or enumerations are allowed too



Schema Structure

Field

- Describes one data value



- **Rule** – allowed number of value occurrences
 - Default = 0 or 1 value
 - repeated = 0 or more values (i.e. an arbitrary number)
 - The order of individual values is preserved
- **Name** – name of a given field
- ...

Schema Structure

Field

- **Type**
 - Atomic: `int32`, `int64`, `double`, `string`, `bool`, `bytes`, ...
 - Mappings to data types of particular programming languages as well as default values are introduced
 - Composed: messages, enumerations, ...
- **Tag** – internal integer identifier
 - Used to identify fields within a message in a binary format
 - Frequently used fields should be assigned lower tags
 - Since lower number of bytes will then be needed

Schema Structure

Enumeration

- Description of a **predefined list of values**
- The first item is considered to be default and its value must be equal to 0



A few other constructs are available too (e.g. maps)

Lecture Conclusion

Data formats

- Tree: **XML, JSON**
- Graph: **RDF**
- Relational: **CSV**
- Binary: **BSON, Protocol Buffers**