NPRG039

Advanced Aspects and New Trends in XML (and Related) Technologies

RNDr. Irena Holubová, Ph.D.

holubova@ksi.mff.cuni.cz

Lecture 3. XML Alternatives

http://www.ksi.mff.cuni.cz/~holubova/NPRG039/

XML Alternatives

- YAML
- OGDL
- SDL
- DL
- Boulder
- ONX
- JSON
- SMEL
- Property Lists
- ATerms
- SOX
- MicroXML
- LMNL
- JITTs
- ConsiceXML
- SML
- TexMecs
- Waterken Doc
- UBF
- Xqueeze
- …

separate lecture

MicroXML

- Motivation: XML is difficult to understand and process
  - Various historical reasons: namespaces, complex structures of XPath, XQuery, XSLT which are often not exploited, …
  - HTML5: better combined with JSON (simplicity)
- MicroXML = simplification of XML compatible with earlier versions
  - Emerged from discussions of issues of XML
    - XML-DEV mailing list
      - Open, publicly archived, unmoderated list supporting XML implementation and development
      - XML-DEV archives are publicly accessible
  - Under W3C
  - Start of specification: December 2010
  - First specification draft: October 2012

compare W3C specifications

http://www.xml.org/xml-dev

http://www.w3.org/community/microxml/

Note: What is HTML5?

- Status: W3C Recommendation
- News:
  - Support for the latest multimedia
    - `<video>`, `<audio>`, `<canvas>`
  - Integration of SVG and MathML
    - Replaces generic `<object>`
  - New elements/attributes to enrich the semantic content of documents
    - `<section>`, `<article>`, `<header>`
  - Some elements, such as `<a>`, `<cite>` and `<menu>` have been changed, redefined or standardized
  - Scripting application programming interfaces
    - Element canvas for 2D drawing, drag-and-drop, document editing, web storage, …
    - Used with JavaScript

HTML 5

http://www.w3.org/TR/html5/

MicroXML Goals

- Key goals of the community group:
  - The syntax of MicroXML is a subset of XML 1.0.
  - MicroXML specifies a data model and a mapping from the syntax to the data model, which is substantially consistent with XML 1.0.
  - MicroXML is dramatically simpler than XML regarding its specification, syntax, and data model.
  - MicroXML is designed to complement rather than replace XML, JSON, and HTML.
  - MicroXML supports the needs of documents, in particular mixed content.
  - MicroXML supports Unicode.
  - MicroXML supports the use of text editors for authoring.
  - MicroXML is able to straightforwardly represent HTML.
  - The specification of MicroXML is as self-contained as is practical.

MicroXML

Well-formedness

- XML: parsers are required to halt immediately upon encountering the first error
  - User-unfriendly for users used to HTML
- MicroXML: does not insist on any approach to handling errors
  - Parser should signal error, but can halt, recover, continue, …

```
<para>Hello, I claim to be <strong>MicroXML</para>
```

  - e.g., parser can add `</strong>` to correct the input, but it cannot claim that it is a MicroXML input

MicroXML

Basic Constructs

- Supports only one encoding: UTF-8
- Document contains markup and character data
  - Elements, attributes, character data
- Namespaces are not supported
  - Colons (`':'`) are forbidden in element and attribute names
  - `xmlns` attribute is forbidden
- Whitespaces in attribute values are not normalized

```
<para>Hi. I'm some form of
  <abbr ref="Extensible Markup Language">XML</abbr></para>
```

```
<para>Hi. I'm some form of
  <abbr ref="Extensible Markup
Language">XML</abbr></para>
```

Two same XML documents, but different MicroXML documents

MicroXML
PIs, Comments, Declarations

- PIs are prohibited in MicroXML
- Comments are allowed, but they are not a part of the data model
  - Ignored by applications
  - Idea: “comments are for people, not programs”
- XML declarations are not supported
- Entities: only hexadecimal-encoded character

Simply Speaking:
- Elements = structure
- Attributes = metadata
- Content = content

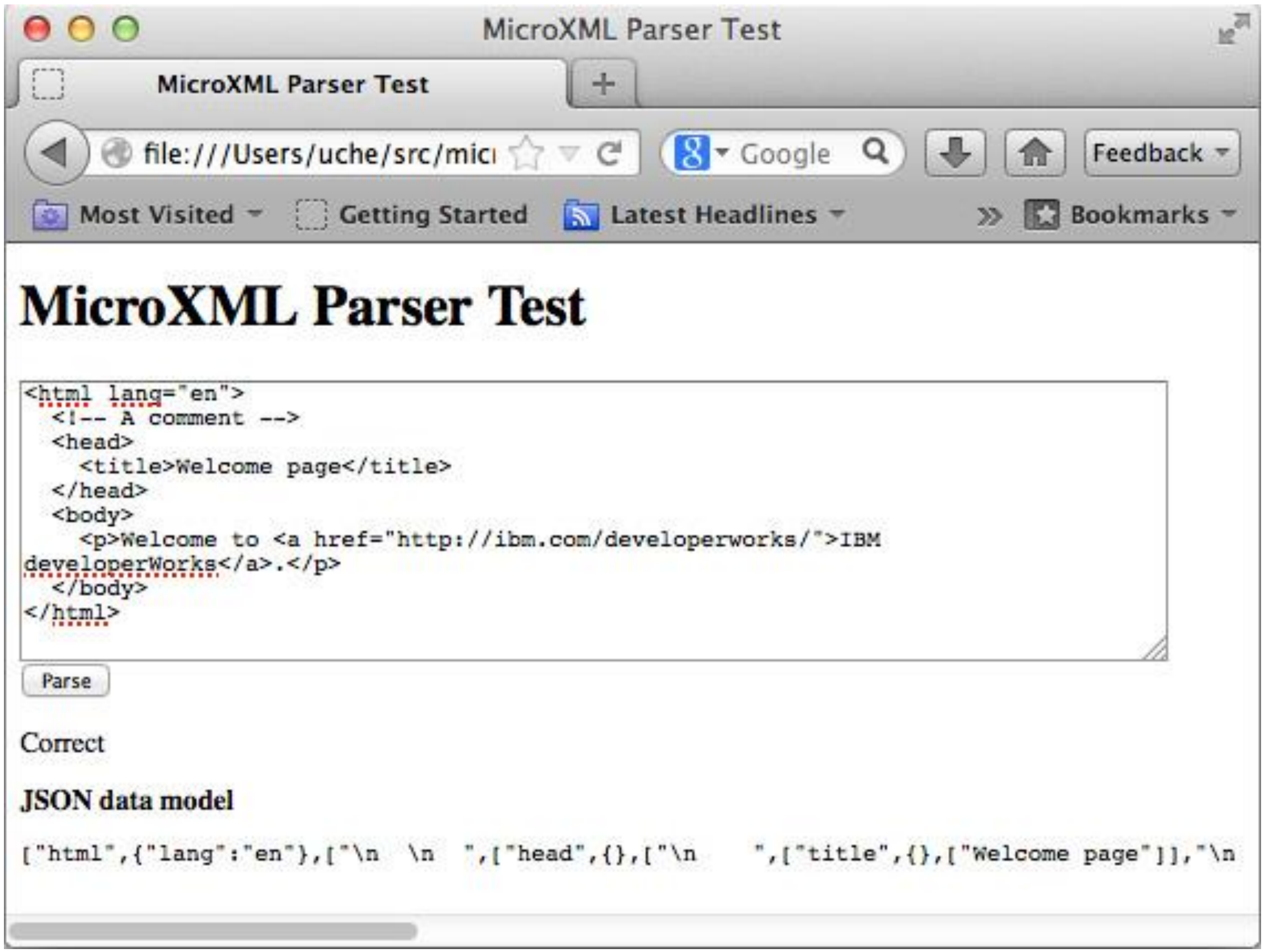
MicroXML Grammar

```
# Documents
  document ::= comments (doctype comments)? element comments
  comments ::= (comment | s)*
  doctype ::= "<!DOCTYPE" s+ name s* ">"
# Elements
  element ::= startTag content endTag
            | emptyElementTag
  content ::= (element | comment | dataChar | charRef)*
  startTag ::= '<' name (s+ attribute)* s* '>'
  emptyElementTag ::= '<' name (s+ attribute)* s* '/>'
  endTag ::= '</' name s* '>'
# Attributes
  attribute ::= attributeName s* '=' s* attributeValue
  attributeValue ::= '"' ((attributeValueChar - '"') | charRef)* '"'
                   | "'" ((attributeValueChar - "'") | charRef)* "'"
  attributeValueChar ::= char - ('<'|'&')
  attributeName ::= "xml:"? name
# Data characters
  dataChar ::= char - ('<'|'&'|'>')
# Character references
  charRef ::= decCharRef | hexCharRef | namedCharRef
  decCharRef ::= '&#' [0-9]+ ';'
  hexCharRef ::= '&#x' [0-9a-fA-F]+ ';'
  namedCharRef ::= '&' charName ';'
  charName ::= 'amp' | 'lt' | 'gt' | 'quot' | 'apos'
```

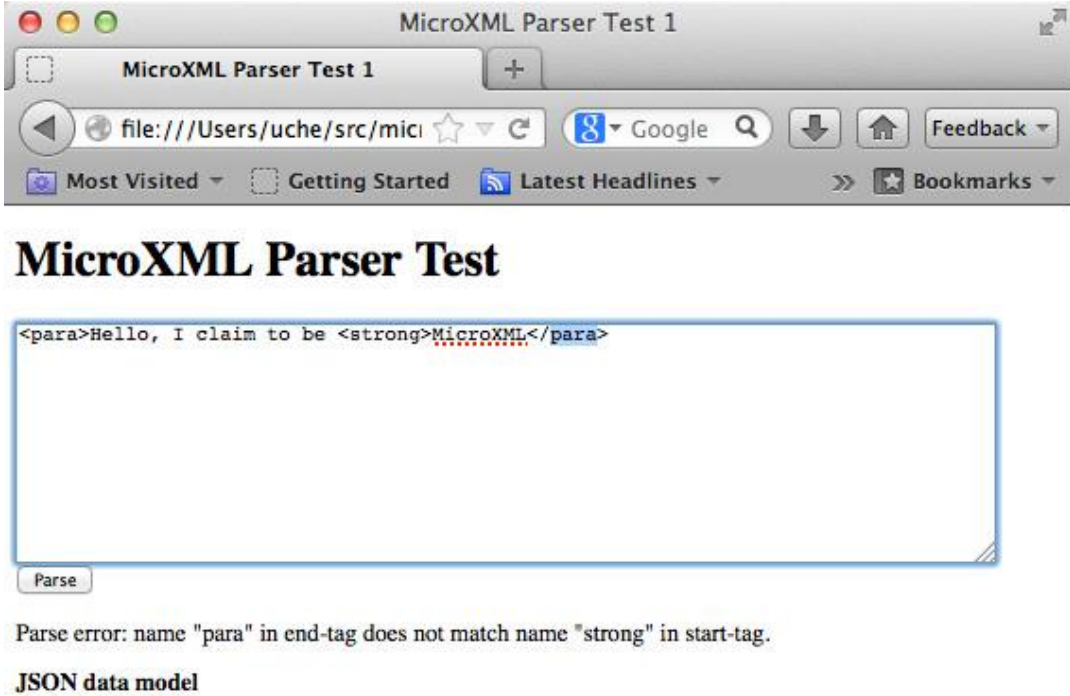
MicroXML Grammar

```
# Comments
  comment ::= '<!--' (commentContentStart commentContentContinue*)? '-->'
# Enforce the HTML5 restriction that comments cannot start with '-' or '->'
  commentContentStart ::= (char - ('-'|'>')) | ('-' (char - ('-'|'>')))
# As in XML 1.0
  commentContentContinue ::= (char - '-') | ('-' (char - '-'))
# Names
  name ::= nameStartChar nameChar*
  nameStartChar ::= [A-Z] | [a-z] | "_" | [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x2FF]
                  | [#x370-#x37D] | [#x37F-#x1FFF] | [#x200C-#x200D]
                  | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF]
                  | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
  nameChar ::= nameStartChar | [0-9] | "-" | "." | #xB7 | [#x0300-#x036F]
             | [#x203F-#x2040]
# White space
  s ::= #x9 | #xA | #xD | #x20
# Characters
  char ::= s | ([#x21-#x10FFFF] - forbiddenChar)
  forbiddenChar ::= surrogateChar | #FFFE | #FFFF
  surrogateChar ::= [#xD800-#xDFFF]
```

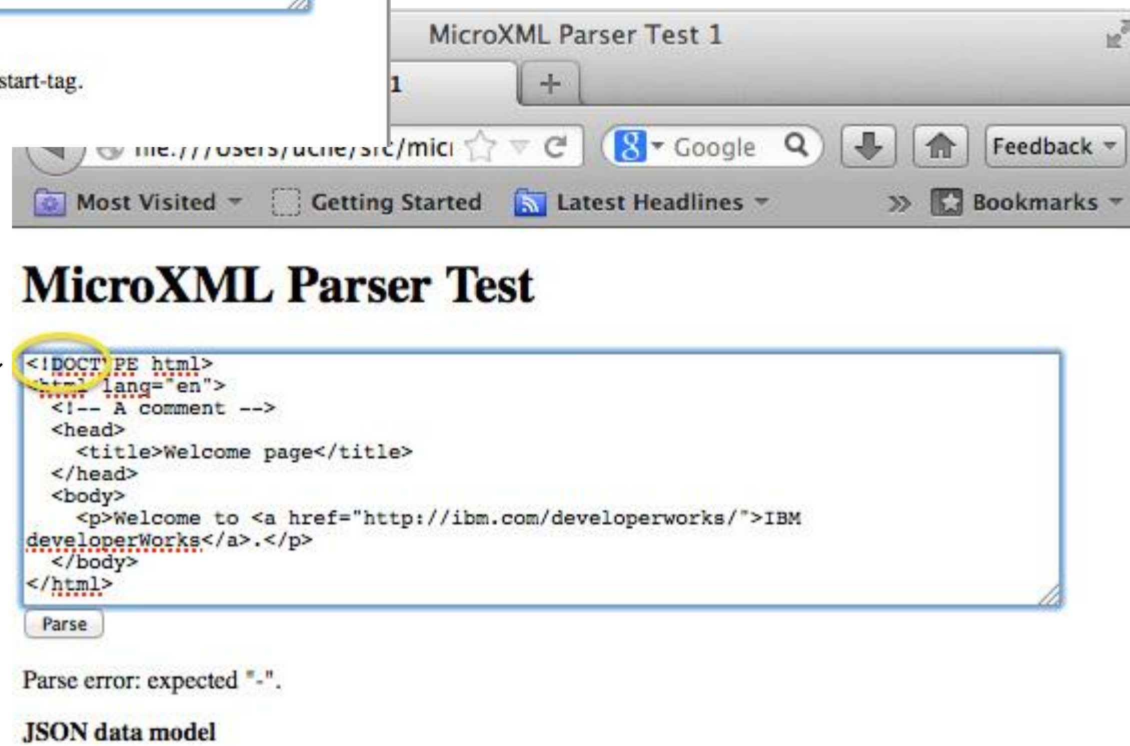
MicroXML Example 1



James Clark's JavaScript parser (microxml-js)



This parser does not support DTD declarations



MicroXML

Future Work

- Many follow-up discussions
  - Error recovery
  - Micro schemata
  - Micro transforms
- More advanced implementations
- Support in various tools

```
<comment lang="en" date="2012-09-11">
I <em>love</em> &#xB5;<!-- MICRO SIGN -->XML!<br/>
It's so clean &amp; simple.</comment>
```

Simple Outline XML (SOX)

- An alternative syntax for XML
- For reading and creating XML content in a text editor
  - To be then easily transformed into correct XML
- Uses indenting to represent the structure of an XML document
  - Eliminates the need for closing tags
- Supports elements, attributes and text
  - Comments, PIs, … are not supported
- Java SAX parser and a SAX serialiser is provided

http://www.langdale.com.au/SOX/

SOX Grammar

- Each line represents a(n) element/attribute/text node

SOX	XML
element>	<element ... > ... </element>

element>
    attribute=value
    ...
	<element
	   attribute="value"
	 ... >
	 ...
	</element>

element>
    ...
    text node
    ...
	<element ...>
	    ... text node ...
	</element>

SOX Grammar

- Indentation represents element-subelement relationship

A>	<A>
B>	<B>
C>	<C/>
D>	</B>
	<D/>
	</A>

SOX Grammar

- Multiline text is quoted with triple quote marks

pre>
    """Text spanning several
    lines forming a single XML
        'so-called' text node"""
	<pre>Text spanning several lines forming a single XML 'so-called' text node</pre>

SOX and White Spaces

- Whitespaces = spaces and tabs
- Whitespace is treated as follows:
  - Lines consisting only of whitespace are ignored.
  - Indentation is represented by a whitespace at the beginning of a line
    - Tabs = 8 spaces
  - In unquoted text:
    - Leading and trailing whitespace (other than the indent) is ignored
    - Internal span of whitespace is treated as a single space
  - A single space is unconditionally appended to the unquoted text forming an XML text node.
    - Can be prevented by quoting
  - All other whitespace is ignored

SOX Examples

```
stylesheet>
    xmlns=http://www.w3.org/1999/XSL/Transform
    version=1.0
    template>
        match=node()
        copy>
            apply-templates>
                select=node()
```

XSLT script

```
html>
    head>
        title> My Home Page
    body>
        h1> Contact Details
        p>  I can be contacted at
            a>  href=mailto:me@myplace.net
                this address
            except when on vacation.
```

XHTML document

YAML (Ain't Markup Language)

- Originally: Yet Another Markup Language
- Human-readable data serialization format
- Concepts from programming languages
  - C, Perl, and Python
  - Aim: easy mapping of data types
- Ideas from XML and data format of electronic mail (RFC0822)
  - Hierarchical data representation
- First proposal: 2001
- Sample use-cases: configuration files, debugging dumps, document headers (similar to, e.g., e-mails), …

http://yaml.org/

YAML

Design Goals

- YAML is easily readable by humans.
- YAML data is portable between programming languages.
- YAML matches the native data structures of agile languages.
  - Python, Ruby, PHP, …
  - Simplicity, automated unit testing, quickness and lightness of development, …
- YAML has a consistent model to support generic tools.
- YAML supports one-pass processing.
- YAML is expressive and extensible.
- YAML is easy to implement and use.

YAML

Basics

- Unicode encoding
- Basic primitives:
  - mappings (hashes/dictionaries)
  - sequences (arrays/lists)
  - scalars (strings/numbers)
- Indentation-based scoping
  - Similar to Python
  - For easy inspection of the data's structure
  - No support for tabs (must be replaced with spaces)
- Content can be nested

YAML

Collections

- Collections
  - Use indentation for scope
  - Begin each entry on its own line
- Entries:
  - In sequences: begin with "- "
  - In mappings: use ": "
- Comments begin with "#"

```
- Mark McGwire
- Sammy Sosa
- Ken Griffey
```

Sequence of scalars

comment

```
hr:  65    # Home runs
avg: 0.278 # Batting average
rbi: 147   # Runs Batted In
```

Mapping scalars to scalars

```
american:
  - Boston Red Sox
  - Detroit Tigers
  - New York Yankees
national:
  - New York Mets
  - Chicago Cubs
  - Atlanta Braves
```

Mapping scalars to sequences

```
-
  name: Mark McGwire
  hr:   65
  avg:  0.278
-
  name: Sammy Sosa
  hr:   63
  avg:  0.288
```

Sequence of mappings

YAML

Simplifications

- In case of small, simple data
  - Sequence: comma-separated list within square brackets `[]`
  - Mapping: comma separated list within curly braces `{}`

```
- [name , hr, avg]
- [Mark McGwire, 65, 0.278]
- [Sammy Sosa , 63, 0.288]
```

Sequence of sequences

```
Mark McGwire: {hr: 65, avg: 0.278}
Sammy Sosa: {
    hr: 63,
    avg: 0.288
  }
```

Mapping of mappings

```
# Products purchased
- item    : Super Hoop
  quantity: 1
- item    : Basketball
  quantity: 4
- item    : Big Shoes
  quantity: 1
```

Compact nested mapping

- Within a collection, `key: value` pairs can start immediately following the "`-`", "`:`", or "`?`" (see later)

YAML

Structures

a logical part of data

- “---” indicate start of a document
- “...” indicate end of a document
  - Without starting a new one, closing a stream connection etc.

```
# Ranking of 1998 home runs
---
- Mark McGwire
- Sammy Sosa
- Ken Griffey

# Team ranking
---
- Chicago Cubs
- St Louis Cardinals
```

Two documents in a stream
(each with a leading comment)

```
---
time: 20:03:20
player: Sammy Sosa
action: strike (miss)
...
---
time: 20:03:47
player: Sammy Sosa
action: grand slam
...
```

Play by play feed from a game

YAML

Anchors and Aliases

- Repeated nodes (objects) are first identified by an anchor
  - Marked with "&"
- Then they can be aliased
  - Referenced with "*"

DTD: ID, IDREF(S)

XML Schema: key, keyref

```
---
hr:
  - Mark McGwire
  # Following node labeled SS
  - &SS Sammy Sosa
rbi:
  - *SS # Subsequent occurrence
  - Ken Griffey
```

Node for "Sammy Sosa" appears twice in this document

YAML

Complex Keys

- "? " indicates a complex mapping key

```
? - Detroit Tigers
  - Chicago cubs
:
  - 2001-07-23

? [ New York Yankees,
    Atlanta Braves ]
: [ 2001-07-02, 2001-08-12,
    2001-08-14 ]
```

keys

values

Mapping between sequences

YAML

Strings

- Scalar string content:
  - Literal style (indicated by "|") where all line breaks are significant
  - Folded style (indicated by ">"): each line break is folded to a space
    - Unless it ends an empty or a more-indented line

```
# ASCII Art
--- |
  \//||\/||
  // ||  ||__
```

ASCII art, new lines are preserved

```
--- >
  Mark McGwire's
  year was crippled
  by a knee injury.
```

In the folded scalars,
newlines become spaces

YAML

Strings

```
>
 Sammy Sosa completed another
 fine season with great stats.

   63 Home Runs
   0.288 Batting Average

 What a year!
```

Folded newlines are preserved
for "more indented" and empty lines

```
name: Mark McGwire
accomplishment: >
  Mark set a major league
  home run record in 1998.
stats: |
  65 Home Runs
  0.278 Batting Average
```

Indentation determines scope of
">" and "|"

YAML
Quotation

- YAML's quotation:
  - Plain style (most examples so far)
  - Quoted styles
    - Double-quoted style – provides escape sequences
      - For arbitrary strings
    - Single-quoted style – when escaping is not needed
      - Only the quote can be escaped when needed
- All can span multiple lines
  - Line breaks are always folded

e.g., when a key involves ":"

```
unicode: "Sosa did fine.\u263A"
control: "\b1998\t1999\t2000\n"
hex esc: "\x0d\x0a is \r\n"

single: '"Howdy!" he cried.'
quoted: ' # Not a ''comment''.'
tie-fighter: '|\-*-/|'
```

Quotation

```
plain:
  This unquoted scalar
  spans many lines.

quoted: "So does this
  quoted scalar.\n"
```

Multi-line scalar

YAML

Data Types

- Untagged nodes are given a type depending on the application
  - seq, map, str, int, float, null, binary, omap (ordered map), set, …

```
canonical: 12345
decimal: +12345
octal: 0o14
hexadecimal: 0xC
```

Integers

```
canonical: 1.23015e+3
exponential: 12.3015e+02
fixed: 1230.15
negative infinity: -.inf
not a number: .NaN
```

Floating point

```
null:
booleans: [ true, false ]
string: '012345'
```

Miscellaneous

```
canonical: 2001-12-15T02:59:43.1Z
iso8601: 2001-12-14t21:59:43.10-05:00
spaced: 2001-12-14 21:59:43.10 -5
date: 2002-12-14
```

Timestamps

YAML

Explicit Typing

- Denoted with a tag
  - Identifier starting with "!"
- Global tags = URIs (i.e., unique across all applications)
  - May be specified in a tag shorthand notation using a handle
- Application-specific local tags may also be used

```
---
not-date: !!str 2002-04-28

picture: !!binary |
 R0lGODlhDAAMAIQAAP//9/X
 17unp5WZmZgAAAOfn515eXv
 Pz7Y6OjuDg4J+fn5OTk6enp
 56enmleECcgggoBADs=

application specific tag: !something |
 The semantics of the tag
 above may be different for
 different documents.
```

Explicit typing

```
%TAG ! tag:clarkevans.com,2002:
--- !shape
  # Use the ! handle for presenting
  # tag:clarkevans.com,2002:circle
- !circle
  center: &ORIGIN {x: 73, y: 129}
  radius: 7
- !line
  start: *ORIGIN
  finish: { x: 89, y: 102 }
- !label
  start: *ORIGIN
  color: 0xFFEEBB
  text: Pretty vector drawing.
```

Global tags

YAML

Explicit Typing

```
# Unordered sets are represented as a
# mapping where each key is associated
# with a null value
--- !!set
? Mark McGwire
? Sammy Sosa
? Ken Griff
```

Unordered set

```
# Ordered maps are represented as
# a sequence of mappings, with
# each mapping having one key
--- !!omap
- Mark McGwire: 65
- Sammy Sosa: 63
- Ken Griffy: 58
```

Ordered mapping

Bigger Example 1

An Invoice

```
--- !<tag:clarkevans.com,2002:invoice>
invoice: 34843
date   : 2001-01-23
bill-to: &id001
    given  : Chris
    family : Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city    : Royal Oak
        state   : MI
        postal  : 48046
ship-to: *id001
product:
    - sku         : BL394D
      quantity    : 4
      description : Basketball
      price       : 450.00
    - sku         : BL4438H
      quantity    : 1
      description : Super Hoop
      price       : 2392.00
tax  : 251.42
total: 4443.52
comments:
    Late afternoon is best.
    Backup contact is Nancy
    Billsmer @ 338-4338.
```

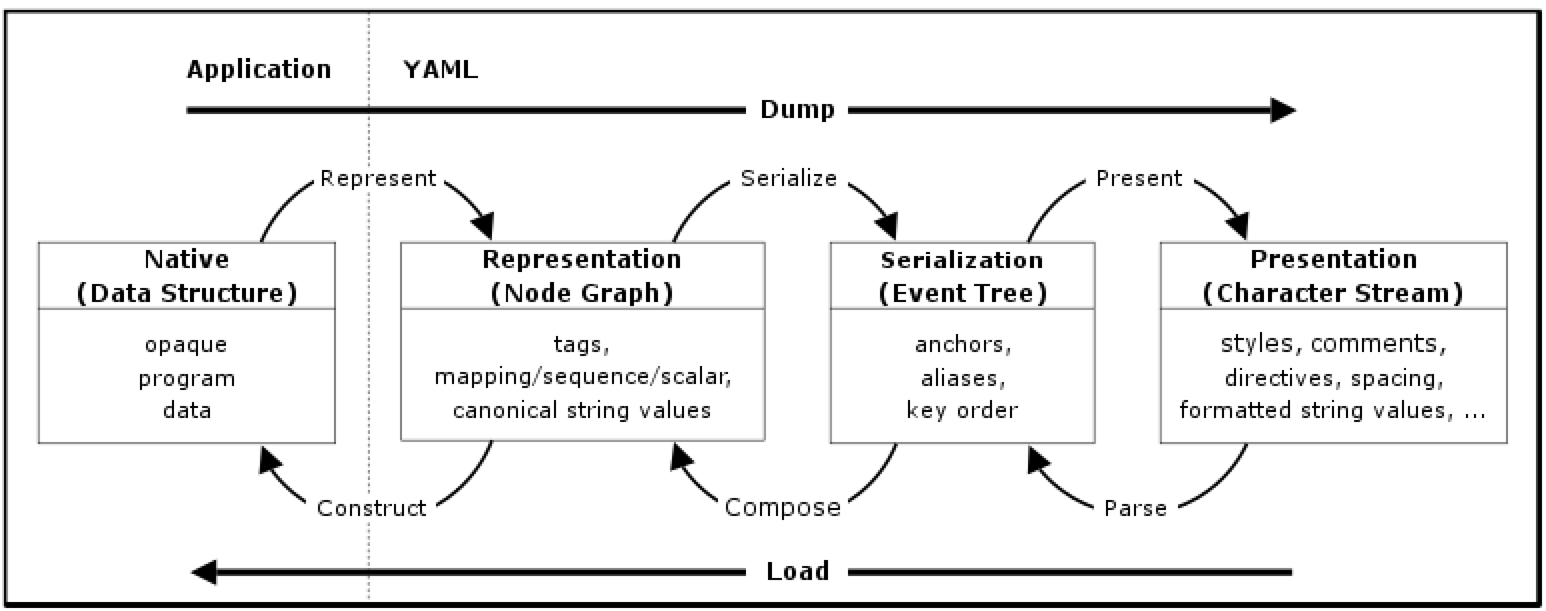
Bigger Example 2

Log File

```
---
Time: 2001-11-23 15:01:42 -5
User: ed
Warning:
  This is an error message
  for the log file
---
Time: 2001-11-23 15:02:31 -5
User: ed
Warning:
  A slightly different error
  message.
---
Date: 2001-11-23 15:03:17 -5
User: ed
Fatal:
  Unknown variable "bar"
Stack:
  - file: TopClass.py
    line: 23
    code: |
      x = MoreObject("345\n")
  - file: MoreClass.py
    line: 58
    code: |-
      foo = bar
```

How YAML Processor Works

- Translating between native data structures and a character stream
  - Dump native data structures → character stream
  - Load native data structures ← character stream



How YAML Processor Works

Dump

- **Representing Native Data Structures**
  - Using sequences, mappings and scalars
  - Form a directed graph
- **Serializing the Representation Graph**
  - Representation is serialized to an ordered tree
  - Problem:
    - Maps are not ordered
      - An ordering is imposed
    - Nodes may be referenced more than once
      - Replaced by anchors and aliases
- **Presenting the Serialization Tree**
  - Presenting the YAML serializations as a character stream in a human-friendly manner
  - Requires presentation details: the amount of indentation, how to format scalar content, …

Particular strategy depends on the YAML processor

How YAML Processor Works

Load

- **Parsing the Presentation Stream**
  - Stream of characters → a series of events
  - Discards all the details introduced in the presentation process
    - Indentation, formatting, …
- **Composing the Representation Graph**
  - Takes a series of serialization events and produces a representation graph
- **Constructing Native Data Structures**
  - Based only on the information available in the representation
    - Not on comments, directives, mapping key order, node styles, scalar content format, indentation levels, …

YAML

Relation to JSON

- JSON:
  - Primary design goal: simplicity and universality
    - Trivial to generate and parse
    - At the cost of reduced human readability
  - Lowest common denominator information model
    - Can be easily processed by every modern programming environment
- YAML:
  - Primary design goal: human readability
  - Support for serializing arbitrary native data structures
  - Consequence: more difficult to parse/generate
- YAML can be viewed as a natural superset of JSON
  - Every JSON file is also a valid YAML file

YAML

Relation to XML

- No direct correlation
- Ongoing efforts to define standard XML/YAML mappings
  - Results in usage of subsets at both sides
- XML
  - Based on SGML → many structural constraints
  - A pioneer in many aspects
- YAML:
  - Primarily a data serialization language
  - Result of lessons learned from XML and other technologies

<?xml?>

YAML

Implementations and Bindings

- C++
- Ruby
- Python
- Java
- Pearl
- C#
- PHP
- JavaScript
- Haskell
- …

http://www.yaml.org/

Simple Declarative Language (SDL)

- An XML alternative
- “Easy way to describe lists, maps, and trees of typed data in a compact, easy-to-read and type-aware representation”
- Use-cases: property files, configuration files, logs, and simple serialization requirements, …

http://sdl4r.rubyforge.org/doc/

SDL

Data Types

- Type-aware:
  - Unicode string – examples: `"hello"` or `` `aloha` ``
  - character (32 bits signed) – example: `'/'`
  - long integer (64 bits signed)
  - float (32 bits signed)
  - double float (64 bits signed)
  - decimal (128+ bits signed)
  - boolean – examples: `true` or `false` or `on` or `off`
  - date `yyyy/mm/dd` – example `2005/12/05`
  - date time `yyyy/mm/dd hh:mm(:ss)(.xxx)(-ZONE)`
    example – `2005/12/05 05:21:23.532-JST`
  - time span
  - Base64
  - `null`

SDL
Comments

- Four comment types
  - `//` single line comments identical to Java, C, etc.
    - Can occur anywhere in a line
    - All text after // up to the new line will be ignored.
  - `#` property style comments
    - Work the same way as //
  - `--` separator comments useful for visually dividing content
    - Work the same way as //
  - Slash star (`/*`) style multiline comments
    - Everything in between is ignored

SDL

Documents

- Made up of tags = data structure with a list of values, a map of attributes, and (if it has a body) child tags
- Tag contains:
  - a name
    - If not present, the name “content” is used
  - a namespace (optional)
  - 0 or more values (optional)
  - 0 or more attributes (optional)
  - 0 or more children (optional)

```
# name value pairs
first_name "Akiko"
last_name "Johnson"
height 68
```

```
# a tag having only a name
my_tag
```

```
# a tag with a value list
person "Akiko" "Johnson" 68
```

SDL
Documents

```
# a tag with attributes
person first_name="Akiko" last_name="Johnson" height=68
```

```
# a tag with values and attributes
person "Akiko" "Johnson" height=60
```

```
# a tag with attributes using namespaces
person name:first-name="Akiko" name:last-name="Johnson"
```

```
# a tag with values, attributes, namespaces, and children
my_namespace:person "Akiko" "Johnson" dimensions:height=68 {
    son "Nouhiro" "Johnson"
    daughter "Sabrina" "Johnson" location="Italy" {
        hobbies "swimming" "surfing"
        languages "English" "Italian"
        smoker false
    }
}
```

```
# anonymous tag examples
files {
    "/folder1/file.txt"
    "/file2.txt"
}
```

SDL

String Literals

- Within double quotes (“”)
  - Double quotes, backslash characters (`\`), and new lines (`\n`) must be escaped
- Within backquotes (` `)
  - Not necessary (or possible) to escape any type of character within a backquote string literal

```
file "C:\\folder\\file.txt"
say "I said \"something\""
```

```
line "this is a \
    long string of text"
```

```
file `C:\folder\file.txt`
say `I said "something"`
regex `\w+\.suite\(\)`
```

```
long_line `This is
     a long line
     fee fi fo fum`
```

SDL

Binary Literals

- Base64 characters enclosed in square brackets [ ]

```
key [sdf789GSfsb2+3324sf2] name="my key"
image [
    R3df789GSfsb2edfSFSDF
    uikuikk2349GSfsb2edfS
    vFSDFR3df789GSfsb2edf
  ]
upload from="ikayzo.com" data=[
    R3df789GSfsb2edfSFSDF
    uikuikk2349GSfsb2edfS
    vFSDFR3df789GSfsb2edf
  ]
```

SDL

DateTime Literals

- Date, time span, and date/time literals
- If a timezone is not specified, the locale timezone is used

```
date 2005/12/05
hours 03:00:00
minutes 00:12:00
seconds 00:00:42
short_time 00:12:32.423 # 12 minutes, 32 seconds, 423 milliseconds
long_time 30d:15:23:04.023 # 30 days, 15 hours, 23 mins, 4 secs, 23 millis
before -00:02:30 # 2 hours and 30 minutes ago
in_japan 2005/12/05 14:12:23.345-JST
```

SDL and Ruby

- SDL4R = SDL parser for Ruby

```
size 4
smoker false
```

```
root = Tag.new("root").read(Pathname.new("values.sdl"))
size = root.child("size").value
smoker = root.child("smoker").value
```

```
require 'fileutils'
require 'sdl4r'

root = SDL4R::Tag.new("root") do
    new_child("server") do
        set_attribute("port", 1234)
    end
end
File.open("my_directory/my_config.sdl", "w") { |io|
    io.write(root.children_to_string)
}
```

```
server port=1234
```

http://sdl4r.rubyforge.org/

Base64

- Binary-to-text encoding
  - Represent binary data in an ASCII string format
    - e.g., for data transfer
  - To ensure that the data remains intact
- First task: choice of 64 encoding characters
  - A subset common to most encodings
  - Printable
- e.g., MIME's Base64 implementation uses A–Z, a–z, and 0–9 for the first 62 values
  - Other versions differ in the last two characters

Base64

Example

Man is distinguished, not only by his reason, but by this
singular passion fromother animals, which is a lust of the
mind, that by a perseverance of delightin the continued and
indefatigable generation of knowledge, exceeds the
shortvehemence of any carnal pleasure.

TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24s
IGJ1dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmlt
YWxzLCB3aGljaCBpcyBhIGx1c3Qgb2YgdGhlIG1pbmQsIHRoYXQgYnkgYSBw
ZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGludWVkIGFuZCBp
bmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRz
IHRoZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=

- Approx. 33% longer

Base64

Example

- In ASCII M, a, n are stored as 77, 97, 110
- 8-bit binary values: 01001101, 01100001, 01101110
- Joined together: 010011010110000101101110
- Groups of 6 bits are converted into individual numbers from left to right
  - 2^6 = 64 different binary values
- The input is extended with 0s if necessary

<table>
<tr><th>Text content</th><td colspan="8">M</td><td colspan="8">a</td><td colspan="8">n</td></tr>
<tr><th>ASCII</th><td colspan="8">77 (0x4d)</td><td colspan="8">97 (0x61)</td><td colspan="8">110 (0x6e)</td></tr>
<tr><th>Bit pattern</th><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><th>Index</th><td colspan="6">19</td><td colspan="6">22</td><td colspan="6">5</td><td colspan="6">46</td></tr>
<tr><th>Base64-encoded</th><td colspan="6">T</td><td colspan="6">W</td><td colspan="6">F</td><td colspan="6">u</td></tr>
</table>

Base64 Index Table

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

References

- MicroXML: http://www.w3.org/community/microxml/
- Introducing MicroXML: http://archive.xmlprague.cz/2013/presentations/Introducing_MicroXML.pdf
- SOX: http://www.langdale.com.au/SOX/
- YAML: http://yaml.org/
- YAML specification: http://www.yaml.org/spec/1.2/spec.html
- Simple Declarative Language: http://sdl4r.rubyforge.org/

  http://sdl4r.rubyforge.org/doc/