RUB

RUHR-UNIVERSITÄT BOCHUM

Bridging the Gap: Secure and lossless conversion of XML data structures to the JSON format

Bachelor thesis ■ March 30, 2017 – June 29, 2017

July 4, 2017

Advisors: Dennis Felsch & Paul Rösler

Jan Holthuis

hgi Lehrstuhl für Netz- und Datensicherheit

Overview

1 Introduction

2 Approach

3 Evaluation
- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

Usage of JSON and XML

- Web APIs are booming since *Web 2.0* and *IoT* hype
  - most of them use XML, JSON or both as data format
- Some “normal” websites are now based on these formats (e.g. *AngularJS*)
- Lots of file formats are XML-based (e.g. RSF/ASF, MathML, SVG, XHTML, ODT, OOXML, . . .)
- There are even JSON-based databases like *CouchDB* and *MongoDB*
- Countless industry standards in all sectors use XML

Why convert between XML and JSON?

- Parsing JSON is usually faster and less resource heavy than XML
- XML has more features and is widely used by the industry
- … but the complexity makes it harder for humans to read and adds more overhead
- Support by programming languages, frameworks and libraries is inconsistent

⇒ **plenty of reasons for converting between XML and JSON!**

Overview

1 Introduction

2 Approach

3 Evaluation

- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

Scope of thesis

- Find a way to convert arbitrary XML documents to JSON
- Be able to convert the JSON documents back to XML
- The conversion should . . .
  - result in **well-formed JSON/XML**,
  - **require no additional metadata** (type hints, etc.),
  - be **lossless** and
    - XML documents before and after XML → JSON → XML round-trip should be (logically) equivalent
  - be **secure**.
    - Not vulnerable to known attacks against parsers

Finding a lossless/secure method

- Define verifiable criteria for lossless conversion
- Check conversion tools that are already available
- If no sufficient converter exists:
  - Develop custom solution or extend existing one
  - Evaluate that custom tool, too!

Security

The converter should be secure against known XML attacks.

- Denial of Service (DoS)
    - Billion Laughs
    - Quadratic Blowup
    - Entity Recursion
- File System Access (FSA)
    - External DTD
    - XXE
    - XSLT
    - XInclude
- Server-Side Request Forgery (SSRF)
    - External DTD
    - XXE
    - `SchemaLocation/noNamespaceSchemaLocation`
    - XInclude

Losslessness

Definition (Lossless conversion)

Let $V := \{x \mid x \text{ is a valid JSON value}\}$ and
$W := \{x \mid x \text{ is a well-formed XML document}\}$.
A conversion method $K = (f_{enc}, f_{dec})$ is *lossless* if and only if

$$(f_{dec} \circ f_{enc})(x) \overset{\mathrm{c14n}}{=} x \tag{1}$$

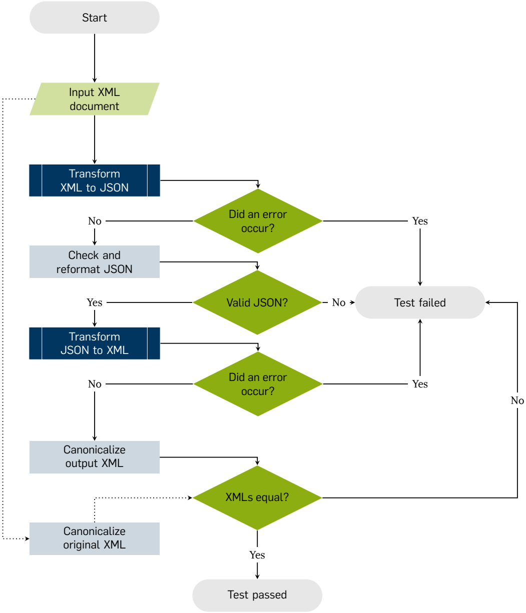
$$f_{enc} : W \mapsto V \tag{2}$$

$$f_{dec} : V \mapsto W \tag{3}$$

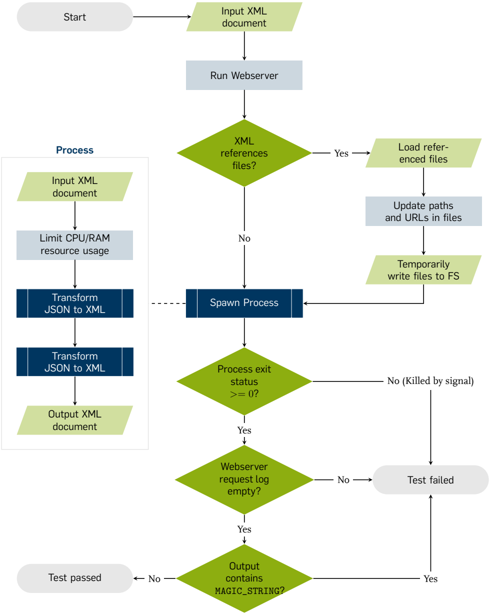
for all $x \in W$.

Testing method

- Checking every conceivable XML document is impossible
- Instead, devise a set of test cases that cover all parts of XML spec
- Also test conversion of complex real-world XML documents
- Use Testing framework to automate evaluation
  - Implemented using Python 3.5, defusedxml, demjson
  - Runs inside a Docker container
  - 2600 Lines of Code (LoC)


Start
Input XML document
Transform XML to JSON
Did an error occur?
No
Yes
Check and reformat JSON
Valid JSON?
Yes
No
Test failed
Transform JSON to XML
Did an error occur?
No
Yes
No
Canonicalize output XML
XMLs equal?
Canonicalize original XML
Yes
Test passed

hg NDS


Start
Input XML document
Run Webserver
XML references files?
Yes
Load referenced files
Update paths and URLs in files
Temporarily write files to FS
No
Process
Input XML document
Limit CPU/RAM resource usage
Transform JSON to XML
Transform JSON to XML
Output XML document
Spawn Process
Process exit status >= 0?
No (Killed by signal)
Yes
Webserver request log empty?
No
Test failed
Yes
Test passed
No
Output contains MAGIC_STRING?
Yes

hg NDS

Converters tested

- **Cobra vs Mongoose** Paul Battley, MIT, Ruby
- **GreenCape XML Converter** Niels Braczek, MIT, PHP
- **Json-lib** Andres Almiray, Apache 2.0, Java
- **JsonML** Stephen M. McKamey, MIT, JavaScript
- **JXON** Martin Raifer, Mozilla, GNU GPL 3.0, JavaScript
- **Json.NET** James Newton-King, MIT, C#
- **org.json.XML** Sean Leary / JSON.org, MIT, Java
- **Pesterfish** Jacob Smullya, MIT, Python
- **x2js** Abdulla G. Abdurakhmanov, Apache 2.0, JavaScript
- **x2js (Fork)** Sander Saares / Axinom, Apache 2.0, JavaScript
- **xmljson** S. Anand, MIT, Python

Overview

1 Introduction

2 Approach

3 Evaluation
- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

Test results

- Tested 11 converters
- Some support multiple modes
- Used 123 test documents

⇒ more than 2000 (!) tests

Overview

RUB

1 Introduction

2 Approach

3 Evaluation

- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

hg NDS

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
1	Attribute (multiple)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
2	Attribute	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
3	CDATA close in Text Node	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	Escaped CDATA section	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	CDATA section with Markup	✓	✗		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	CDATA support	✓	✗		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	Comments[2]	✗		✗															
8	Deep nesting	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	
9	Duplicate Child tag Names (Alternating Order)		✓		✓	✓				✓	✓								
10	Duplicate child tag names (different NS prefix)	✓	✓	✓	✓	✓	✓	✓				✓							
11	Duplicate child tag names	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓		✓	✓	✓
12	Element Order		✓		✓	✓				✓	✓								
13	Empty Elements	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓
14	Mixed Content	✗			✓	✓				✓	✓		✗						
15	Processing Instructions (Attibute Data)	✗		✗		✓		✓											
16	Processing Instructions (Markup)	✗		✗		✓		✓											
17	Processing Instructions outside root	✗				✓		✗											
18	Processing Instructions (Arbitrary Data)	✗		✗		✓		✓											
19	Processing Instructions (Whitespace)	✗		✗		✓													
20	Processing Instructions (Basic)	✗		✗		✓		✓											
21	Root Element Attribute	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		
22	Root Element Tag Name	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	
23	Simple Element List	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	
24	Type Inference with floats (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						✓
25	Type Inference with floats	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						
26	Type Inference with doubles (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						✓
27	Type Inference with doubles	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						
28	Type Inference with Boolean values (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
29	Type Inference with Boolean values	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	
30	Type Inference with Big Integers (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
31	Type Inference with Big Integers	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	
32	Whitespace (Indentation)				✓	✓				✓	✓		✗						
33	Whitespace (Mixed Content)	✗			✓	✓				✓	✓		✗						
34	Whitespace (Clean/Dirty)			✓	✓	✓				✓	✓	✓	✓					✓	
35	Namespace declaration (multiple)		✓	✓	✓	✓		✓				✓	✓						
36	Namespace declaration with prefix	✓	✓	✓	✓	✓	✓	✓				✓	✓						
37	Namespace declaration	✓	✓	✓	✓	✓	✓	✓				✓	✓						
	Total	**23**	**24**	**19**	**30**	**36**	**23**	**28**	**9**	**26**	**26**	**25**	**23**	**4**	**16**	**4**	**16**	**11**	**12**

[1] Extended JsonML version

[2] optional, support not required

General XML support

- Mixed Content support is sketchy
- Leading/Trailing Whitespace and indentation is often discarded
- Only few converters support Element Order
- Processing Instructions are not supported at all

XML Comments

- Retaining XML comments is not necessary
- All converters ignore comments except Json.NET
- Json.NET converts them to *JavaScript*-style block comments (`/* Comment */`)
- …but this violates the JSON specification (ECMA-404, RFC 7149)

Overview

1 Introduction

2 Approach

3 Evaluation

- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
38	C0 set (u000009, 00000A & 00000D)				✓	✓				✓	✓	✓	✓					✓	
39	Space Char (u000020)			✓	✓	✓				✓	✓	✓	✓					✓	
40	ASCII printable (u000021-00007E)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	
41	Discouraged (u00007F-000084)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
42	NEL Control Char (u000085)	✗	✓		✓	✓	✓			✓	✓	✓	✓					✓	
43	Discouraged (u000086-00009F)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
44	BMP plane I (u0000A0-000FFF)	✗	✓		✓	✓				✓	✓	✓	✓						
45	BMP plane II (u001000-001FFF)	✗	✓		✓	✓				✓	✓	✓	✓						
46	BMP plane III (u002000-002FFF)	✗	✓		✓	✓				✓	✓	✓	✓					✓	
47	BMP plane IV (u003000-003FFF)	✗	✓		✓	✓				✓	✓	✓	✓					✓	
48	BMP plane V (u004000-004FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
49	BMP plane VI (u005000-005FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
50	BMP plane VII (u006000-006FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
51	BMP plane VIII (u007000-007FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
52	BMP plane IX (u008000-008FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
53	BMP plane X (u009000-009FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
54	BMP plane XI (u00A000-00AFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓						
55	BMP plane XII (u00B000-00BFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
56	BMP plane XIII (u00C000-00CFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
57	BMP plane XIV (u00D000-00D7FF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
58	BMP plane XV (u00E000-00EFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
59	BMP plane XVI (u00F000-00FDCF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
60	BMP plane Discouraged (u00FDD0-00FDEF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
61	BMP plane XVII (u00FDF0-00FFFD)	✗	✓		✓	✓				✓	✓	✓	✓						
62	SMP plane (u010000-01FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
63	SMP plane Discouraged (u01FFFE-01FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
64	SIP plane (u020000-02FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
65	SIP plane Discouraged (u02FFFE-02FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
	Total	**1**	**26**	**2**	**28**	**28**	**21**	**1**	**0**	**28**	**28**	**28**	**28**	**0**	**19**	**0**	**19**	**24**	**0**

[1] Extended JsonML version

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
66	Unassigned plane 3 (u030000-03FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
67	Plane 3 Discouraged (u03FFFE-03FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
68	Unassigned plane 4 (u040000-04FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
69	Plane 4 Discouraged (u04FFFE-04FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
70	Unassigned plane 5 (u050000-05FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
71	Plane 5 Discouraged (u05FFFE-05FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
72	Unassigned plane 6 (u060000-06FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
73	Plane 6 Discouraged (u06FFFE-06FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
74	Unassigned plane 7 (u070000-07FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
75	Plane 7 Discouraged (u07FFFE-07FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
76	Unassigned plane 8 (u080000-08FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
77	Plane 8 Discouraged (u08FFFE-08FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
78	Unassigned plane 9 (u090000-09FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
79	Plane 9 Discouraged (u09FFFE-09FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
80	Unassigned plane 10 (u0A0000-0AFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
81	Plane 10 Discouraged (u0AFFFE-0AFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
82	Unassigned plane 11 (u0B0000-0BFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
83	Plane 11 Discouraged (u0BFFFE-0BFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
84	Unassigned plane 12 (u0C0000-0CFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
85	Plane 12 Discouraged (u0CFFFE-0CFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
86	Unassigned plane 13 (u0D0000-0DFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
87	Plane 13 Discouraged (u0DFFFE-0DFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
88	SSP plane (u0E0000-0EFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
89	SSP plane Discouraged (u0EFFFE-0EFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
90	SPUA-A plane (u0F0000-0FFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
91	SPUA-A plane Discouraged (u0FFFFE-0FFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
92	SPUA-B plane (u100000-10FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
93	SPUA-B plane Discouraged (u10FFFE-10FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓		✓		✓	✓	
	Total	0	28	0	28	28	28	0	0	28	28	28	28	0	28	0	28	28	0

[1] Extended JsonML version

Character Support

- CobraVsMongoose, Json-lib and org.json.XML only support ASCII characters
- The latter two convert all other characters converted to question marks (`?`)
- Number of question marks depends on the number of bytes used to encode the char in UTF-8
- Example:
  - `FOR ALL` (U+2200, "∀")

  → 3 bytes in UTF-8 (`0xE2 0x88 0x80`)

  → 3 question marks in JSON (`"???"`)

Character Support

Whitespace

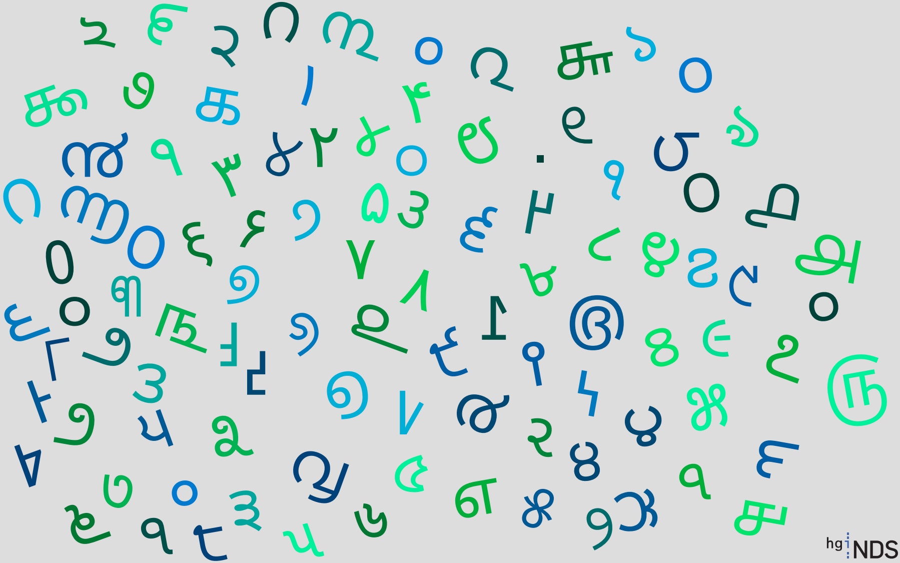
- Some converters discard leading/trailing Whitespace
- Can affect not only Tab, Carriage Return, Line Feed and Space…
- …but lots of other chars, too!

CHARACTER TABULATION	000009
LINE FEED (LF)	00000A
CARRIAGE RETURN (CR)	00000D
SPACE	000020
NEXT LINE (NEL)	U+0085
NO-BREAK SPACE	U+00A0
OGHAM SPACE MARK	U+1680
MONGOLIAN VOWEL SEPARATOR	U+180E
EN QUAD	U+2000
EM QUAD	U+2001
EN SPACE	U+2002
EM SPACE	U+2003
THREE-PER-EM SPACE	U+2004

FOUR-PER-EM SPACE	U+2005
SIX-PER-EM SPACE	U+2006
FIGURE SPACE	U+2007
PUNCTUATION SPACE	U+2008
THIN SPACE	U+2009
HAIR SPACE	U+200A
LINE SEPARATOR	U+2028
PARAGRAPH SEPARATOR	U+2029
NARROW NO-BREAK SPACE	U+202F
MEDIUM MATHEMATICAL SPACE	U+205F
IDEOGRAPHIC SPACE	U+3000
ZERO WIDTH NO-BREAK SPACE	U+FEFF

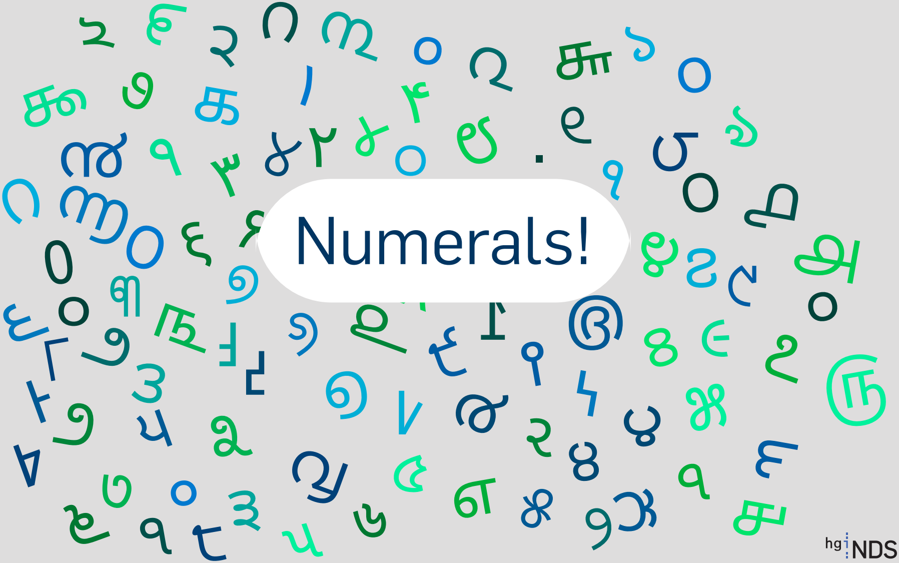
Character Support

What’s this?



Character Support

What's this?



Numerals!

Character Support

Numeral Transformation

- **xmljson** transforms Unicode numerals into their ASCII equivalents
- . . . but only those on the Unicode BMP plane (U+000000 to U+00FFFF)

ARABIC-INDIC DIGITS	U+0669	U+0660	KHMER DIGITS	U+17E9	U+17E0
EXTENDED ARABIC-INDIC DIGITS	U+06F9	U+06F0	MONGOLIAN DIGITS	U+1819	U+1810
NKO DIGITS	U+07C9	U+07C0	LIMBU DIGITS	U+194F	U+1946
DEVANAGARI DIGITS	U+096F	U+0966	NEW TAI LUE DIGITS	U+19D9	U+19D0
BENGALI DIGITS	U+09EF	U+09E6	TAI THAM HORA DIGITS	U+1A89	U+1A80
GURMUKHI DIGITS	U+0A6F	U+0A66	TAI THAM THAM DIGITS	U+1A99	U+1A90
GUJARATI DIGITS	U+0AEF	U+0AE6	BALINESE DIGITS	U+1B59	U+1B50
ORIYA DIGITS	U+0B6F	U+0B66	SUNDANESE DIGITS	U+1BB9	U+1BB0
TAMIL DIGITS	U+0BEF	U+0BE6	LEPCHA DIGITS	U+1C49	U+1C40
TELUGU DIGITS	U+0C6F	U+0C66	OL CHIKI DIGITS	U+1C59	U+1C50
KANNADA DIGITS	U+0CEF	U+0CE6	VAI DIGITS	U+A629	U+A620
MALAYALAM DIGITS	U+0D6F	U+0D66	SAURASHTRA DIGITS	U+A8D9	U+A8D0
SINHALA LITH DIGITS	U+0DEF	U+0DE6	KAYAH LI DIGITS	U+A909	U+A900
THAI DIGITS	U+0E59	U+0E50	JAVANESE DIGITS	U+A9D9	U+A9D0
LAO DIGITS	U+0ED9	U+0ED0	MYANMAR TAI LAING DIGITS	U+A9F9	U+A9F0
TIBETAN DIGITS	U+0F29	U+0F20	CHAM DIGITS	U+AA59	U+AA50
MYANMAR DIGITS	U+1049	U+1040	MEETEI MAYEK DIGITS	U+ABF9	U+ABF0
MYANMAR SHAN DIGITS	U+1099	U+1090	FULLWIDTH DIGITS	U+FF19	U+FF10

Overview

1 Introduction

2 Approach

3 Evaluation
- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
94	MSWord 2003 XML file	✗		✗		✓		✗											
95	MSWord XML file	✗		✗		✓		✗											
96	OpenDocument sample file	✗	✗	✗	✓	✓							✗						
97	XML 1.0 5th Edition (XHTML version)	✗		✗	✓	✓		✗					✗						
98	XML 1.0 5th Edition (XML version)	✗	✗	✗		✓		✗					✗				✗		✗
99	SVG Car Demo	✗		✗	✓	✓							✗						
100	DocBook V5.0 transition guide	✗	✗	✗	✓	✓		✗					✗						
101	MusicBrainz.org API Response	✗	✓	✗	✓	✓													
102	SVG Photos Demo		✗		✓	✓							✗						
103	RSS 0.91 sample document				✓	✓				✓	✓		✗						
104	RSS 0.92 sample document	✗			✓	✓				✓	✓		✗						
105	RSS 2.0 sample document				✓	✓				✓	✓		✗						
106	XML tree Example (Thesis)	✗				✓							✗						
	Total	0	1	0	9	13	0	0	0	3	3	0	0	0	0	0	0	0	0

[1] Extended JsonML version

Overview

1 Introduction

2 Approach

3 Evaluation
- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

Security

- Json-lib is vulnerable to more than 50% of the attacks tested (DoS, FSA, SSRF)
- Pesterfish, xmljson and are vulnerable to DoS attacks
  - Python converters (Pesterfish, xmljson) are insecure if `ElementTree` parser from the standard library is used
  - Alternative: Use hardened `defusedxml` libary
- Json.NET is vulnerable for SSRF attacks

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
107	Billion Laughs Attack (PE Version)	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
108	Billion Laughs Attack	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
109	Entity Recursion Attack (PE)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
110	Entity Recursion Attack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
111	Quadratic Blowup Attack	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
112	Doctype Parameter Entity FSA Attack	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
113	External PE DTD FSA Attack	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
114	XInclude FSA Attack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
115	XSLT FSA Attack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
116	XXE FSA Attack (PE)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
117	XXE FSA Attack	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
118	DOCTYPE URL Invocation	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
119	SchemaLocation NoNamespace URL Invocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
120	SchemaLocation URL Invocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
121	XInclude URL Invocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
122	XXE URL Invocation (PE)	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
123	XXE URL Invocation	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Total	17	17	8	17	17	17	14	17	15	17	17	17	17	17	17	17	17	17

[1] Extended JsonML version

Overview

RUB

1 Introduction

2 Approach

3 Evaluation

- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

hg NDS

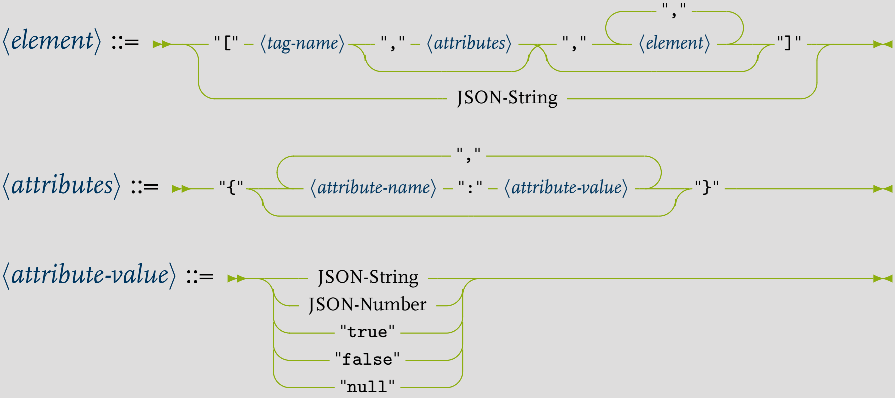
JsonML

JSON Markup Language

- *Leader of the pack* that passed 112 of 123 tests
- Originally written in JavaScript, but there are also other implementations (e.g. Java: `org.json.JSONML`)
- Fairly mature: Github repo dates back to November 2006
- Nicely documented grammar (Backus-Naur form)
- Downsides:
  - Resulting *JSON* is not really “friendly”
  - No support for XML Processing Instructions (no tested solution supports them)

JsonML

Syntax



⟨*element*⟩ ::=
"[" ⟨*tag-name*⟩ "," ⟨*attributes*⟩ "," "," ⟨*element*⟩ "]"
JSON-String

⟨*attributes*⟩ ::=
"{" "," ⟨*attribute-name*⟩ ":" ⟨*attribute-value*⟩ "}"

⟨*attribute-value*⟩ ::=
JSON-String
JSON-Number
"true"
"false"
"null"

JsonML

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="style.css"?>
<!-- Nice album! -->
<albums>
  <album catno="ARGO LP-628">
    <artist>Ahmad Jamal Trio</artist>
    <title>At The Pershing</title>
    <recording>Recorded <date>January 16,
     ↪    1958</date>.</recording>
  </album>
</albums>
```

```
[ "albums", "\n  ",
  [ "album", { "catno": "ARGO LP-628" }, "\n    ",
    [ "artist", "Ahmad Jamal Trio" ], "\n    ",
    [ "title", "At The Pershing" ], "\n    ",
    [ "recording", "Recorded ", [ "date", "January 16,
     ↪    1958" ], "." ], "\n  "
  ], "\n"
]
```

JsonML

Efficiency I

Size comparison of JsonML, Pesterfish and XML at the example of a Flat OpenDocument Text (*.odt):

	Size (in bytes)	Compared to XML (in %)	
		Size	Change
Canonical XML	5787196	100.0	0
JsonML[a]	5405329	93.4	−6.6
Pesterfish[a]	15061634	260.3	+160.3
Pesterfish[b]	14480612	250.2	+150.2

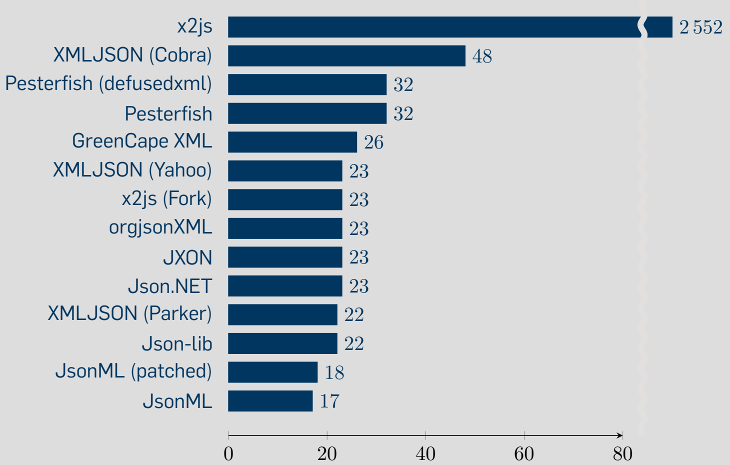
[a] JSON unchanged

[b] insignificant JSON whitespace removed

JsonML

Efficiency II

Nesting Depth comparison at the example of a Microsoft Word XML Document:



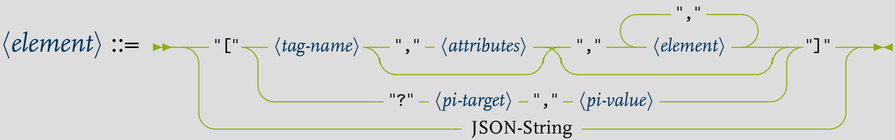
JsonML + Processing Instructions

Adding Support

- JsonML is missing support for Processing Instructions (PIs)
- Example PI: `<?my-target plus some data?>`
- Can be described via tuple $P := \langle \mathit{target}, \mathit{data} \rangle$
- JSON representation: `["?my-target", "plus some data"]`
- Tag names may not start with "?", so it can't be mistaken for an element containing text node
- If PIs appear outside root element as top level constructs – Use an empty string as parent tag name

JsonML + Processing Instructions

Syntax Extension



JsonML + Processing Instructions

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="style.css"?>
<!-- Nice album! -->
<albums>
  <album catno="ARGO LP-628">
    <artist>Ahmad Jamal Trio</artist>
    <title>At The Pershing</title>
    <recording>Recorded <date>January 16,
     ↪     1958</date>.</recording>
  </album>
</albums>
```

```
["", "\n",
  [ "?xml-stylesheet", "href=\"style.css\"" ],"\n","\n",

  [ "albums", "\n  ",
    [ "album", { "catno": "ARGO LP-628" }, "\n    ",
      [ "artist", "Ahmad Jamal Trio" ], "\n    ",
      [ "title", "At The Pershing" ], "\n    ",
      [ "recording", "Recorded ", [ "date", "January 16,
       ↪     1958" ], "." ], "\n  "
    ], "\n"
  ]
]
```

Overview

RUB

1 Introduction

2 Approach

3 Evaluation
- General XML support
- Character Support
- Complex XML Documents
- Security

4 Lossless and secure conversion

5 Conclusions

hg NDS

Conclusions

- Apparently, lossless conversion of XML to JSON is not trivial
- No converter could fully fullfill all requirements out of the box
- JsonML with added PI support can convert XML in a secure and lossless manner
- In that case, it's even possible to convert full-blown ODF/OOXML documents

Limitations

- Focus on XML→JSON→XML conversion
- Parsing DTDs, Entities and Schemas were out of scope
- Only bi-directional converters (i.e. those that implement inverse conversion, too) could be tested

Future Research

- Look at conversion that uses additional DTD/Schema information
  - Improve usage of native JSON types
  - Ability to collapse or replace insignificant whitespace
  - Use arrays/objects depending on Schema's `maxOccurs`
- Evaluate JSON→XML→JSON converters and their security
- Newer technologies like JSON Schema, JSON Include, etc. will probably increase attack surface of JSON parsers

  ⇒ closer look might be worthwhile

Thanks!

Questions?

Reach out via email:

- **Jan Holthuis**
  jan.holthuis@rub.de

Title image: Bob Mical, *"Data Center"* (flickr.com, CC BY-NC 2.0)