Elliotte Rusty Harold
elharo@ibiblio.org
August 2018

From XML to Flat Buffers: Markup in the Twenty-teens

Google Cloud

Warning!

The Contenders

- XML
- JSON
- YAML
- EXI
- Protobufs
- Flat Protobufs

What Uses What

From technology, tools, and systems I use frequently. There are many others.

	XML	JSON	YAML	EXI	Protobuf	Flat Buffers
App Engine Standard Java	X		X			
App Engine Flex			X			
Kubernetes		X	X			
Eclipse	X					
Maven	X					
Ant	X					
Google “APIs”	X	X	X		X	X
Publishing	X					

Google Cloud

XML

XML

- Very well defined standard
- By far the most general format:
    - Mixed content
    - Attributes and elements
- By far the best tool support. Nothing else is close:
    - XSLT
    - XPath
    - Many schema languages:
        - W3C XSD
        - RELAX NG

More Reasons to Choose XML

- Most composable for mixing and matching markup; e.g. MathML+SVG in HTML
- Does not require a schema.
- Streaming support: very large documents
- Better for interchange amongst unrelated parties
- The deeper your needs the more likely you’ll end up here.

Why Not XML?

- Relatively complex for simple tasks
- Limited to no support for non-string programming types:
    - Numbers, booleans, dates, money, etc.
    - Lists, maps, sets
    - You can encode all these but APIs don’t necessarily recognize or support them.
- Lots of sharp edges to surprise the non-expert:
    - 9/10 are namespace related
    - Attribute value normalization
    - White space
- Some security issues if you’re not careful (Billion laughs)

JSON

- Simple for object serialization and program data. If your data is a few basic types (int, string, boolean, float) and data structures (list, map) this works well.
- More or less standard (7-8 of them in fact)
- Consumption libraries for essentially all significant languages

Why Not JSON?

- It is surprising how fast needs grow past a few basic types and data structures.
- No comments!
- Some security issues early on, though these are mostly resolved.
- Implicit schemas.
- Usually the entire document is parsed up front before any data is available to the consumer. Not good for streaming and very large documents.

Limited Tool Support

- No good, cross language solutions for:
  - Schemas
  - Query language
  - Transform language
  - E.g. unit tests for libraries.json

Standards and tools are incomplete and inconsistent.

- ~7 different purported standards
- Areas of difference
  - Text encoding
  - Comments
  - Trailing commas in lists
  - How big can an int or a float be? What happens if one is too big?
  - NaN and Inf
  - Duplicate keys
- Works better in smaller, more homogenous environments; e.g. libraries.json

YAML

- Much loved by Python programmers
    - Not surprisingly for something that comes out of Python, indentation matters a great deal; for good or ill
- Technically a superset of JSON
- **Most human legible of the formats discussed**
- Supports references
- Streams well, unlike JSON and perhaps more easily than XML

Why Not YAML?

- Specification is very weak compared to XML and even JSON.
- It’s surprisingly hard to write a fully conformant YAML parser that correctly handles all the variations and edge cases.
- Consequently not all parsers are fully conformant. You **MUST** test your parser library to make sure it can handle the data you expect to throw at it.
- Arguably dangerous for consuming untrusted user input, especially with data binding. Parser bugs abound.
- Tool support is weak compared to JSON and extremely weak compared to XML.

EXI

- Efficient XML Interchange
- Binary XML
- Limited uptake, limited tool support
- For size, limited advantage compared to gzipped XML. *Maybe* some small speedups.
- Bottom Line:
    - If you need XML, use XML.
    - If you need something smaller and faster, go to next slide.

Protobufs

- Wicked Fast
- Very Compact
- Similar data structures to JSON: basic programming types like int and boolean, along with lists and structs.
- Used by gRPC. Sweet spot for protobufs: communication within a distributed program that is nonetheless a unified whole that just happens to run on multiple computers.
- Used internally at Google
- Major languages (C++, Java, Go, Python) are well supported. Others less so.

Why Not Protobufs?

- Opaque binary format: relatively hard to inspect and debug.
- Size is limited to what you can comfortably deserialize into a single object.
- Schema is absolutely required. Must be present on both ends. You can’t process a protobuf without knowing the schema.
- Even more tightly coupled to classes and generated code than JSON. E.g. structs rather than maps.
- Versioning and updates are tricky.
  - Can’t remove anything
  - Watch out for required fields
- Works better behind the firewall than across the Internet

Flat Buffers

- Like protobufs but you don’t have to load the whole thing.
- Even faster than protobufs
- Limited support for now

Sweet Spots: XML

- Narrative content: words in a row meant for people to read. Books, articles, email, etc.
- Streaming data
- Complicated information hierarchies that don’t map easily to standard programming data structures
- Unknown schema, extensible formats
- Communication between different parties without pre-existing agreements
- Human editable

Sweet Spots: JSON

- Object and database serialization
- Program output that will be consumed by other programs. E.g. service APIs

Sweet Spots: YAML

- Config files
- Human editable

Sweet Spots: EXI

- I can’t think of any

Sweet Spots: Protobufs

- Execution Speed (or size, but usually speed) is the most important consideration; and you’re willing to invest a lot more money, time, and staff to save milliseconds.
- Object and database serialization
- Program output that will be consumed by the same or closely related programs. E.g. service APIs that assume the use of a vendor supplied client library to access.

Sweet Spots: Flat Buffers

- Mobile
- Games

Use the Right Tool for the Job

Questions? Disagreements? Rotten Tomatoes?