Do we really want to *see* markup?

James David Mason

Why do we care about seeing markup?

- Not a question we'd have asked 40 years ago
- Raised by the current interest in "Invisible XML"
- Takes me back to 1983 (and before)
- Thus, an introduction to "Invisible SGML"

How I got here

- Scientific editor and writer, Oak Ridge National Laboratory, 1977
- Self-taught compositor, 1978
- Developer, trainer for typesetting and editing staff, 1979
- Problem solver for data acquisition, 1980
- Standards developer, 1981

In the beginning

- "Text-formatting compilers" (batch formatters often created for printing computer documentation)
- Simple text editors (usually programmers' editors)
- Adapted output devices
  - —Line printers
  - —Typewriter-like printers
  - —Graphics devices (plotters, raster devices)
- Early electronic phototypesetters
- No choice but to look at markup

Oak Ridge publishing in transition, 1979–80

- UNIX, DEC PDP-11/70
- Custom editing terminals
- Troff, tbl, eqn
- Bell Laboratories “Programmer’s Workbench Memorandum Macros”
- Autologic APS-μ5 phototypesetter
- Graphics proof device

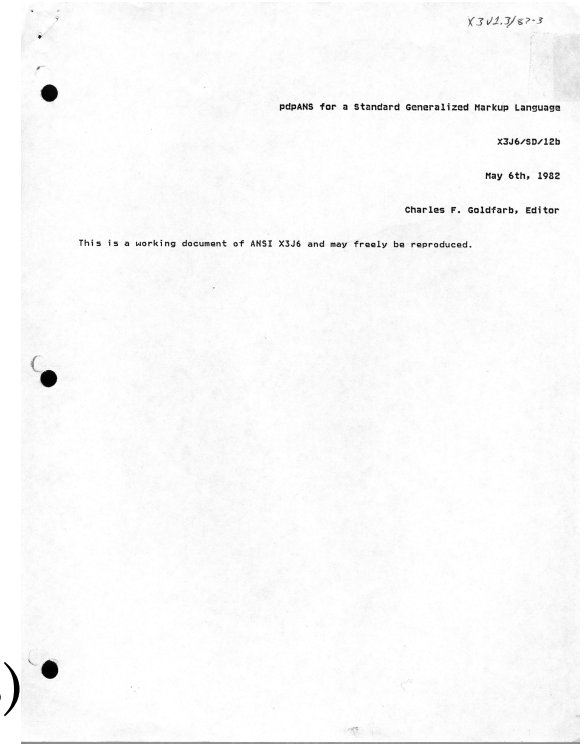
```
.H 1 "Generic markup before SGML"
.BL
.LI
Programmable typesetting languages
.DL
.LI
Runoff (c. 1964, Saltzer, Morris, and McIlroy, MIT)
.LI
Script (c. 1968, Stuart Madnick, IBM)
.LI
Troff (c. 1973, Joe Ossanna, Bell Laboratories)
.LI
@roman {T sub E X}@ (1978, Donald Knuth, Stanford)
.LI
Scribe (1980, Brian Reid, CMU)
.LE
.LI
Macro packages for programmable typesetting languages
.LE
```

(*Note:* this is “troff –mm”, c. 1980.)

Generic markup before SGML

- Programmable typesetting languages
  - —Runoff (c. 1964, Saltzer, Morris, and McIlroy, MIT)
  - —Script (c. 1968, Stuart Madnick, IBM)
  - —Troff (c. 1973, Joe Ossanna, Bell Laboratories)
  - —TEX (1978, Donald Knuth, Stanford)
  - —Scribe (1980, Brian Reid, CMU)
- Macro packages for programmable typesetting languages

SGML, c. 1982


X3V1.3/82-3

PdpANS for a Standard Generalized Markup Language

X3J6/SD/12b

May 6th, 1982

Charles F. Goldfarb, Editor

This is a working document of ANSI X3J6 and may freely be reproduced.

- Documents look much like XML, c. 2019:
  - —"<" and ">" delimiters
  - —attributes on start tags
  - —"/" in end tags
  - —"&" for symbols
- No "SGML Commands" (markup declarations)
  - —Elements
  - —Entities
- Recognition of need for "Document Type Description Syntax"

GCA Standard 101-1983, *GenCode*™ (aka SGML 1983)

- Beginnings of structure definition
  - —STRUCT declaration
  - —Multiple elements defined in one declaration
  - —Whitespace-delimited table
- Early attempts to define markup minimization
  - —Driven by expectation of manual input
  - —More complex than minimization in final standard

Minimization: making markup invisible

Tag omission

- XML

```
<!ELEMENT li     (%text;)>

<list type="bullet">
<li>text</li>
<li>graphics</li>
</list>
```

- SGML

```
<!ELEMENT li - o (%text;)>

<list type=bullet>
<li>text
<li>graphics
</list>
```

Tag omission (and why HTML “<p>” is crazy)

```
<!ELEMENT section - - (title, p+) >
<!ELEMENT title   - - (#PCDATA)  >
<!ELEMENT p       o o (%text;) -p >

<section>
<title>A section title</title>
The first paragraph
<p>
A second paragraph
<p>
A third paragraph
</section>
```

(This technique is why Tim Berners-Lee thought “<p>” was just a separator.)

More minimization

Short tag

- Empty start tag (<>) when only one element is allowed in context
- Omitted generic identifier on end tag (</>)
- Omitted delimiter when followed by tag (`</li</list>`, `<p<quote>`)
- Null end tag (`<li/first item/`)

Saving keystrokes, not hiding markup

Why minimization?

- No language-aware editors until c. 1986
- Earliest editors had only batch validation (i.e., after the input errors were made)
- Users accept doing coding, but SGML is more verbose than earlier markup languages:
  - Troff –mm: `.LI`
  - SGML: `<li>` `</li>`
- Users are lazy (or at least stingy with keyboard effort)!
- Overall drive to reduce clutter
- Outside influences

No philosophical underpinnings, just considering user input

Outside disruption: ODA

- ISO/IEC 8613, *Office Document Architecture* (later *Open Document Architecture*)
- Based on Wolfgang Horak's dissertation (Munich, c. 1980)
- In theory, compound documents (text and graphics)
- Two concurrent structures, "Logical" and "Layout"
- Early attempt at WYSIWYG (i.e., coding hidden from user)
- ODIF: binary interchange format (with both inline and stand-off codes)
- Placed in same committee as SGML and related work (ISO/IEC JTC1/SC18, 1985)
- Finished off by visible markup in SGML
- Influenced SGML thinking

ODA: Killed by Visible Markup

- ODA
  - —An idea, a model; *not* a product
  - —Designed to be invisible, required commercial R&D
  - —Only concrete manifestation was interchange format produced only by machines and consumed only by machines
  - —Vaporware
    - Scope creep, thousands of options never implemented, even in a laboratory
    - Users wouldn't know it even existed
- SGML
  - —Designed to be created by humans; an idea *and* a product
  - —Designed and built by amateurs for their own use
  - —Human-readable editable form *was* interchange format
  - —User-designed early implementations: AAP, DoD, etc.
  - —HTML was a tipping point (SC18 in Dublin, 1995)

ODA AND SGML

- Electro-political rivalry (SC18/WG3 vs. SC18/WG8 after 1985)
- ODA developers
  - —Couldn't stop SGML, a finished standard in 1986
  - —Tried to stop DSSSL and the rest of the SGML-related projects (and without DSSSL, there would have been no XPath, XSL, or XQuery)
- Charles Goldfarb couldn't stand the thought that ODA could do some things SGML couldn't:
  - —SGML representation of ODIF
    - Architectural forms (because ODA didn't have a schema mechanism)
    - CONCUR (for logical and layout structures)
  - —Further inclination to make markup invisible
    - How to implement limited WYSIWYG
    - How to import data

More invisible markup: text as tags

- SGML in late 1982
  - —No DTD yet
  - —Hints of regular expressions for future content model definition
- My fault!
  Suggestion in a presentation to use models with literals

```
memo: to, from, subject, date, body
  to: "To: ", #PCDATA
from: "From: ", #PCDATA
```

- Goldfarb:
  - —That's wrong, no literals to be in models
  - —Let's implement it

Datatag and Shortref

- Datatag: declaring literal strings to indicate element end tags as element transitions (data is passed through)

```
<!ELEMENT table - - (row+)>
<!ELEMENT row   – o ([cell, ", ", " "], cell)>
              <!-- cell, comma and space, arbitrary whitespace -->
```

- Shortref: mapping text events to entities and entities to tags (data is replaced)

```
<!USEMAP   tablemap table>
<!SHORTREF tablemap "&#RS;" rowtag> <!-- "&#RS;" is SGML "record start"-->
<!ENTITY   rowtag STARTTAG row> or <!ENTITY   rowtag "<row>" >
```

Turns two-column comma-separated entries into table body*

*Developed examples in ISO 8879; Bryan, *SGML, An Author's Guide*

Building a memo using Shortref

```
<-- newline plus text generates entities -->
<!SHORTREF memomap "&#RS;To: "   to
                   "&#RS;From: " from
⋮
>
<-- entities map to start tags -->
<!ENTITY to "<to>">
<!ENTITY from "<from>">
⋮
<!USEMAP memomap memo>
<!ELEMENT memo    - - (to, from, subject, date, body)>
<!ELEMENT to      o o (%text;)>
<!ELEMENT from    o o (%text;)>
⋮
<!ELEMENT body    o o (%text;)>
```

Tagging a memo with Shortref

Memo start triggers `memomap`

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
To: B. T. Usdin
From: J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

Tagging a memo with Shortref

Text "`newline, To: `" triggers map entry

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
To: B. T. Usdin
From: J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

Tagging a memo with Shortref

Map entry "to" points to entity "&to;"

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
To: B. T. Usdin
From: J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

Tagging a memo with Shortref

Entity "&to;" generates virtual start tag "<to>", replacing trigger

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
<to>B. T. Usdin
From: J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

Tagging a memo with Shortref

Process repeats for text "`newline, From: `"

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
<to>B. T. Usdin
From: J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

Tagging a memo with Shortref

Process repeats for text "`newline From: `"

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
<to>B. T. Usdin
From: J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

Tagging a memo with Shortref

Process repeats for text "`newline From: `"

```
<!SHORTREF memomap "&#RS;To: " to
                   "&#RS;From: " from>
<!ENTITY to "<to>">
<!ENTITY from "<from>">
<!USEMAP memomap memo>
<!ELEMENT memo - - (to, from,
                subject, date, body)>
<!ELEMENT to   o o (%text;)>
<!ELEMENT from o o (%text;)>
```

```
<memo>
<to>B. T. Usdin
<from>J. D. Mason
Subject: Balisage paper
Date: 1 July 2019

This is a demonstration of Shortref to
generate markup.

</memo>
```

With enough map entries and entities, the final parser output is a fully tagged memo

```
<memo>
<to>B. T. Usdin</to>
<from>J. D. Mason</from>
<subject>Balisage paper</subject>
<date>1 July 2019</date>
<body>
This is a demonstration of Shortref to generate markup.
</body>
</memo>
```

“Invisible SGML” evolved with the development of the standard

- Side effect of minimization (to save keystrokes, storage, data transmission)
- Encouraged by competition with ODA
- Grew through markup tricks (Datatag, Shortref)
- Gradually became methodology for capturing outside data
- Killed by smart editors and XML

XML ended “Invisible SGML”

- No minimization
- No features or declarations for Datatag and Shortref
- No built-in entities for text events (`&#RS; &#RE; &#TAB;`)
- No SGML Declaration to
  - —Enable minimization
  - —Enable features

Interpretation of invisible markup keeps coming back

- In the early days of XML, Simon St. Laurent kept suggesting ways of recreating SGML techniques, including these.
- Steven Pemberton has stirred up interest in “Invisible XML”.
- Michael Sperberg-McQueen is about to talk about “Aparecium”, which brings visibility to invisible tags.

“Invisible SGML” and “Invisible XML”

- Invisible SGML
  - —Assumed regular, consistent, predictable data
  - —Everything dependent on the primary parser (there wasn’t anything else to use)
  - —Depended on features not every parser implemented
  - —Evolved from other requirements to permit simplified data capture
- Invisible XML
  - —“Possible to describe the data using a context-free grammar”*
  - —Implemented in secondary tools (with libraries)
  - —Intended from the outset for data capture

*Michael Sperberg-McQueen, “Aparecium: An XQuery / XSLT library for invisible XML”, Presented at *Balisage: The Markup Conference 2019*.

Why *see* markup?

- The emphasis is not on “invisible” but on markup.
- Visibility was once an necessity but became a political tool.
- Accepting “invisible” markup has long been a trick to get to something that could be resolved as visible markup.
- Markup, whether visible or invisible, is a means to an end.

I’m not sure we all start out wanting to *see* markup, but we seem to feel we need it to be visible to *know what it’s doing*.