Internet Draft (SHAVE)
Expires April 1994
Filename draft-adie-shave-00.ps

C. Adie
Edinburgh University
12 October, 1993

SGML-based Hierarchical Attribute/Value Encoding (SHAVE)

STATUS OF THIS DOCUMENT

This document is an Internet Draft. Internet Drafts are working documents of the Internet Engineering Task Force (IETF), its Areas and its Working Groups. (Note that other groups may also distribute working documents as Internet Drafts.)

Internet Drafts are draft documents valid for a maximum of six months. Internet Drafts may be updated, replaced, or obsoleted by other documents at any time. It is not appropriate to use Internet Drafts as reference material or to cite them other than as a "working draft" or as "work in progress". To learn the current status of any Internet Draft, please check the `1id-abstracts.txt` listing contained in the Internet Drafts Shadow Directories on `ds.internic.net`, `nic.nordu.net`, `ftp.nisc.sri.com`, or `munnari.oz.au`.

SUMMARY

The usefulness of attribute/value pairs for conveying information is well established. There is a need for a standard text-based method of representing attribute/value data, which is capable of being easily written and read by humans, and also easily processed by a computer program. Often, such data is required to be transferred in electronic mail messages. This document describes how SGML (Standard Generalized Markup Language) can be used as the basis for such a representation.

Table of Contents

1. Introduction

The use of attribute/value pairs for conveying information is well established. Although ASN.1 is often used to transmit attribute/value information (eg in X.500), it is not a human readable representation.

There is a need for a standard text-based method of representing attribute/value data, which is capable of being easily written and read by humans, and also easily processed by a computer program. Often, such data is required to be transfered in electronic mail messages.

Typical applications for such a representation are:

- Exchange of personal contact information (such as might be shown on a business card). A recipient might process this information using a database program.
- Exchange of meta-information concerning a resource (eg a file or a service) on the Internet.
- Dissemination of information regarding an event (a meeting, lecture, conference etc.). A recipient might process this information using a personal time manager or simple diary program.
- As a format for an electronic mail "form", for form-filling applications.

This document describes how SGML (Standard Generalized Markup Language) can be used as the basis for such a representation. This document draws heavily on the work done by Dave Crocker on STIF [Crocker 93a] and PCI [Crocker 93b].

1.1 Requirements

The requirements for an attribute/value text format are as follows, roughly in order of importance.

1. Must be easy to read by people who are not computer experts.
2. Must be easy to write by non-experts, using very simple written instructions.
3. Must be easy to write a parser for.
4. Must be capable of handling multi-valued attributes, where there may be significance in the ordering of values.
5. Must be capable of handling nested attribute/value pairs - ie attributes whose values are attribute/value pairs.
6. Must be capable of handling non-text attribute values by reference.
7. Must be able to handle attribute values which are not in US ASCII.

8. It must be easy for a computer program to identify that part of a text file which contains attribute/value information relevant to it.

1.2 SGML

Why use SGML to achieve the above aims? There are several reasons:

- It is the obvious tool for the job, being designed explicitly for structuring text and offering a great deal of flexibility.
- It is supported by a range of existing commercial and public domain tools.
- It is easy for non-expert users to read and write SGML.
- It lends itself to hierarchically-structured data.
- It has very few "special characters" which require escaping in normal text.

To learn about SGML, there's a good little booklet called *"The SGML Primer"*, written by SoftQuad [SoftQuad 91], which offers a rapid walk through the main features of the language. The remainder of this document assumes you have read this booklet (or have an equivalent level of SGML knowlege).

The SGML "bible" is *The SGML Handbook* [Goldfarb 90], which contains the complete text of the ISO standard, as well as a great deal of excellent explanation.

1.3 The SHAVE Approach

The main problem with representing attribute/value information using SGML is one of nomenclature - the word "attribute" has a specific meaning in SGML, which differs from the meaning we've given it so far. To resolve the conflict, in the rest of this document we will use the term "attribute" in its SGML meaning, and speak about "parameter/value" data instead of about "attribute/value" data.

It is envisaged that such SGML-encoded parameter/value data will be parsed by two kinds of programs:

- Programs which are based on generic SGML parsers, which base their parsing on an externally-stored SGML Document Type Definition (DTD).
- Programs which do not understand SGML or DTDs *per se*, and are specific to a particular application. Such programs might be written in awk or Perl, or any other programming language.

In order to make it easier to write programs of the second kind ("naïve parsers"), we will introduce restrictions which limit the way in which an SGML document and DTD may be written. This document therefore describes some general rules (called "the SHAVE rules") for representing parameter/value data in SGML. It does not define a DTD, but describes restrictions which:

- restrict how an application should define its own DTD.

- restrict how a document instance should be written, over and above the rules imposed by the DTD and by SGML.

The remainder of this document is structured as follows. Chapter 2 describes assumptions about the general SGML environment. Chapter 3 lists rules which restrict how DTDs may be written. Chapter 4 describes restrictions on document instances. Chapter 5 (which is not formally part of the specification) gives examples of how the restrictions work.

For ease of identification and reference,

0. SHAVE rules are numbered, indented and printed in italics, like this.

Following each rule is an explanation and/or justification of the rule, if required.

2. General SGML Environment

1. *The SGML Reference Concrete Syntax shall be used, with the modification that the syntax-reference character set shall be ANSI X3.4 instead of ISO646 IRV.*

This rule implies that the SGML Reference Delimeter set is used (see Goldfarb p477). Among other things, it also implies that the alphabetic case of element names is not significant, but the case of entity names is significant.

2. *The SGML Reference Quantity Set shall apply.*

The Reference Quantity Set defines various limits which are usually left to the discretion of an implementor - see Goldfarb p470. This rule ensures that a SHAVE document may be parsed by any generic SGML parser without having to worry about the parser's implementation limits. The most important restriction is on the maximum length of SGML element and attribute names, which must not exceed 8 characters.

3. DTD Restrictions

A "SHAVE application" is the definition of a DTD and associated semantics, together with program(s) which parse conforming document instances. This Chapter defines restrictions with which a SHAVE application's DTD must comply.

3. *A parameter shall be represented as an SGML element.*

This is the key rule for understanding how SGML represents parameter/value information. The name of the parameter is the element name, and the value of the parameter is the element content.

4. *A parameter which takes a single text value shall have a content model of* `(#PCDATA)`.

5. *A parameter whose value is a set or sequence of other parameter/value pairs will have a content model which defines the parameters which may occur within it.*

These rules simply state the obvious modelling of hierarchical parameter/value data by nested SGML elements.

6. *A parameter which takes a list of text values shall have a content model similar to* `(item+)`, *where the* `item` *element is defined as:*

```
<!ELEMENT item O O (#PCDATA)>
```

The declaration of `item` (or a similar element) must be included in the DTD. The following Chapter describes restrictions on document instances relating to the use of this form of element.

7. *If the value of an element contains both (a) a text portion, and (b) subidiary parameter/value information, then the content model of the corresponding element shall specify that the* `#PCDATA` *representing (a) preceeds the other elements which represent (b).*

DTD designers should avoid specifying such content models - elements preferably should contain either other elements or `#PCDATA`, not both. However, there may be cases where both are needed in a single element, and in such cases the `#PCDATA` should come first in order to aid legibility.

8. *The following SGML entity declarations shall be included in an application DTD:*

```
<!ENTITY amp "&">
<!ENTITY lt "<">
```

Any octets (subject to the constraints of the character set in effect) are permitted in the `#PCDATA` content of an element, except for octets with decimal values 38 (the ampersand) and 60 (the less-than sign). These must be represented by entity references of the form `&` and `<` respectively.

9. *Applications which define parameters taking values which are stored externally to the document instance, shall do so using elements with a content notation defined in the application's DTD.*

An application program which processes the document instance may then choose to resolve the reference by retrieving the referenced value. This approach has been chosen rather than the "external entity" feature of SGML in order to meet criteria 1 and 2 of section 1.1 above.

10. *By default, `#PCDATA` within elements comprises characters from ANSI X3.4. Applications which wish to allow the use of alternative character sets shall provide an optional attribute `cset` for each element, which takes a value which uniquely identifies the character set used in `#PCDATA` within that element and all included elements which do not specify their own `cset` attribute.*

The default character set for `#PCDATA` is US ASCII. All markup is done in that character set. Elements inherit `cset` values from their enclosing elements - unfortunately SGML does not provide a keyword to express this.

We need to specify a list of character set names which can be used as values of `cset` attributes.

11. *Where an SGML attribute is specified in the DTD as taking one of a fixed number of values, the values shall be distinct from values of all other attributes of the same element type.*

SGML attributes may be used to qualify the content of an element. This rule allows the attribute name to be omitted in document instances for certain types of attribute, as described in the following Chapter. SGML attributes which take CDATA values do not fall into this category.

12. *Application designers who wish to allow the use of experimental parameters shall define the following element and attribute in their DTD:*

```
<!ELEMENT X - - RCDATA>
<!ATTLIST X type CDATA #REQUIRED>
```

Individuals may then use the `X` element with a `type` of their choice to contain experimental values. Note two points in particular:

- The `</X>` end tag is required.
- SGML requires that no other end tag may occur within an `X` element.

13. *If a MIME registration is required for a SHAVE format, the registered MIME subtype shall be used in the DTD as the document type name.*

The top-level MIME content type will probably be either "text" (if the data can usefully be displayed in text form) or "application" (otherwise). Applying for registration is the responsibility of the application designer(s). Note that SHAVE does not itself have or require a MIME registration.

4. Document Instance Restrictions

The rules in this Chapter restrict how markup occurs. The purpose of these restrictions is twofold - to make it easier for a human to read the document, and to make it easier to write a parser which is not fully SGML-aware.

14. *Comments in a document instance shall start with the four-character sequence `<!--` and end with the three-character sequence `-->`.*

This restriction prohibits spaces from occurring between the `--` and the `>` at the end of the comment (this is normally allowed in SGML). The `<!>` empty comment and comments within tags are excluded by this rule. Comments are discarded by the parser.

15. *Start tags shall not be omitted, and tags shall not be shortened, except where otherwise specified in these rules.*

16. *When a tag occurs in a document, its opening delimeter (less-than sign) must occur as the first non-blank character on the line.*

These rules makes things much easier for non-SGML-aware programs. Note that the element name immediately follows the opening delimiter, but the closing delimeter (greater-than sign) may be preceeded by white space.

17. *Within an element which contains `#PCDATA`, white space occurring before the first non-white character and white space occurring between the last non-white character and the opening delimeter of the tag which closes the element, shall be discarded by the parser.*

This rule ensures that leading spaces, newlines etc, introduced for legibility at the beginning and end of an element are ignored. However, white space within the text of an element is not ignored by the parser and may be treated as significant by the application.

18. *Where an SGML attribute is specified in the DTD as taking one of a fixed number of values which are distinct from values of all other attributes of the same element type, the attribute name shall be omitted in document instances.*

This rule refers to the "Attribute Minimisation" feature of SGML, as described in the ISO standard Annex C.1.1.3 (Goldfarb p70), which permits the omission of the attribute name. The above SHAVE rule makes this omission mandatory.

19. *Where an element has a content model of the form `(item+)` to represent a list of values, and each `item` within the element contains only `#PCDATA`, the following minimisations shall occur within the enclosing element: (a) All `item` end tags shall be omitted. (b) The first `item` start tag after the start tag of the enclosing element shall be omitted. (c) Other occurrences of the `item` start tag shall be shortened to the empty tag `<>`.*

This rule allows the pair of characters <> to be used as separators in a list of values in some common circumstances (see Goldfarb p75).

20. The Reference Close delimeter (;) shall not be omitted.

This delimeter terminates an entity reference, and must always be present. (The use of Record End as a Reference Close delimeter is thus not permitted.)

5. Examples

This chapter gives some examples of how some of the SHAVE rules might apply to a particular imaginard SHAVE application. This chapter is not definitive. For further examples, see [Adie 93], which is a complete specification of a SHAVE-conformant format.

Rules 6 and 19

If an application requires a `computer` parameter which takes a list of values, it might define a DTD containing:

```
<!ELEMENT opsys
<!ELEMENT item     O O (#PCDATA, opsys?)>
<!ELEMENT computer - O (item+)>
```

and a document instance might contain the following list containing three items:

```
<computer> Apple Macintosh
        <> IBM PC
        <> Sun Sparcstation
</computer>
```

Note that if an opsys element is present, then rule 19 does not apply and every item start tag must be present:

```
<computer>
  <item>     Apple Macintosh
  <item>     IBM PC
    <opsys>  DOS
  <item>     Sun Sparcstation
</computer>
```

These rules are complex, but have the advantage of maintaining the legibility of the document instance.

Rule 7

This rule is obeyed by the following element declaration:

```
<!ELEMENT foo - O ((#PCDATA), (bar1 & bar2))>
```

It is NOT obeyed by the following element declaration, which permits the `bar1` and `bar2` elements to occur before the `#PCDATA`:

```
<!ELEMENT foo - O ((#PCDATA) & bar1 & bar2)>
```

Rule 9

To define a parameter the value of which is accessed through a URL, an application would define a notation and an element thus:

```
<!NOTATION url SYSTEM >
<!ELEMENT foo - O (#PCDATA)>
<!ATTLIST foo type NOTATION (url) url>
```

The `foo` element in the document instance would then contain the URL:

```
<foo>  ftp:://ftp.ed.ac.uk/pub/mmaccess/mmaccess.ps.Z
</foo>
```

Rule 10

Suppose a `fargle` element is defined as follows:

```
<!ELEMENT fargle - O (#PCDATA)>
<!ATTLIST fargle cset CDATA #IMPLIED>
```

Then a document containing a `fargle` element containing the string Ä&ò in a character set called Code Page 437 might contain the following:

```
<fargle cset="Code Page 437">*</fargle>
```

where `*` stands for the following sequence of octets:

```
8/14  2/6  6/1  6/13  7/0  3/11  9/5
```

Note that the occurrence of octet `2/6` in the selected character set is replaced by the octet sequence `2/6 6/1 6/13 7/0 3/11` (ie `&` in the default character set). The parser would replace this octet sequence with the single octet `2/6` before emitting it to the application. Note that `2/6` need not correspond to the ampersand in the selected character set (although in this case it does). The point is that the SGML parser cares about octet `2/6`, not the octet(s) which happen to represent the ampersand in the selected character set.

Octet `3/12` (the less-than sign in the default set) would be treated similarly.

Rule 17

This rule means that in the following document fragment:

```
<name>            John Smith
<title>           Leader of the Opposition
</title>
```

the data may be lined up vertically without the leading spaces or tabs or the trailing newlines being treated as significant data. However, in the fragment:

```
<motto>     Weave a circle round him thrice
            And close your eyes with holy dread
</motto>
```

the newline and intervening spaces between the two phrases are preserved by the parser.

Rule 18

This rule implies that given the DTD fragment:

```
<!ELEMENT email - O #PCDATA>
<!ATTLIST email mailtype (rfc822|X.400) #REQUIRED>
```

a document should be encoded:

```
<email rfc822> foo@bar.com
```

and NOT as:

```
<email mailtype="rfc822"> foo@bar.com
```

6. References

[Adie 93] SGML-based Personal Contact Information (SPCI), C.Adie, September 1993. Internet Draft (work in progress).

[Crocker 93a] *Structured Text Interchange Format (STIF)*, D. Crocker, June 1993. Internet Draft (work in progress).

[Crocker 93b] *Encoding for Personal Contact Information (PCI)*, D. Crocker, June 1993. Internet Draft (work in progress).

[Goldfarb 90] *The SGML Handbook*, C. Goldfarb, Oxford University Press 1990 (ISBN 0-19-853737-9).

[SoftQuad 91] *The SGML Primer*, SoftQuad Inc 1991. Available for $10 from:

```
<spci>
 <person key="SoftQuad">
  <org>           SoftQuad Inc
  <work>
   <phone>        +1 416 239 4801
   <street>       56 Aberfoyle Crescent, Suite 810
   <city>         Toronto
   <country>      Canada
   <postcode>     M8X 2W4
</spci>
```

7. Security Considerations

There are no security implications of this specification.

8. Acknowlegements

This work was inspired by Dave Crocker's work on STIF [Crocker 93a] and PCI [Crocker 93b]. Lou Burnard provided helpful comments.

If you provide constructive comments, you could find your name appearing here.

9. Contact

```
<spci>
 <person key="Chris Adie">
  <name>          Chris Adie
  <dept>          Computing Service
  <org>           Edinburgh University
  <email>         C.J.Adie@edinburgh.ac.uk
  <work>
   <phone>        +44 31 650 3363
   <fax>          +44 31 662 4809
   <building>     University Library Building
   <street>       George Square
   <city>         Edinburgh
   <postcode>     EH8 9LJ
   <country>      Great Britain
</spci>
```