The Annotated Rainbow DTD

Rainbow version 2.2

David Sklar
Electronic Book Technologies, Inc.
21 February 94

Note: readers familiar with previous versions of Rainbow should read section 3 to obtain a brief overview of the significant changes in this new version. Also note that the revision bars on the left side show all deltas between 2.1 and 2.2.

0. Legal notes

Copyright 1993,1994 Electronic Book Technologies, Inc.

Permission to use, copy, modify and distribute the Rainbow DTD and this "annotated DTD" documentation for any purpose and without fee is hereby granted, provided that this copyright notice appears in all copies. If you modify the Rainbow DTD, rename your modified version and make it clear that it is not the "official" Rainbow maintained by EBT.

EBT makes no representation about the suitability of the DTD for any purpose other than as the starting point for conversion to SGML. It is provided "as is" without expressed or implied warranty.

The Rainbow DTD is maintained by EBT, Inc. Please direct all questions, bug reports, or suggestions for changes to David Sklar, via email "dfs@ebt.com".

Please note that EBT provides an SGML declaration for this DTD, defining a variant concrete syntax that is required for all Rainbow documents.

Public Identifier: "-//EBT//DTD Rainbow 2.2//EN"

1. Goals

Rainbow is an SGML document type designed to represent documents that have been created via word processor (WP) technologies (e.g. MSWord, WordPerfect, Frame, Interleaf). Rainbow documents are not intended to be stored permanently or to be distributed; Rainbow is only intended to be the ***starting point*** for conversion efforts from WP formats to SGML applications.[1] Rainbow is an optimal starting point for conversion efforts because it provides a layer of "insulation" between the conversion software and the WP proprietary formats (which are typically under-documented, idiosyncratic, and highly subject to change).

The primary design goals:

1. WP-independent
2. Formatting-based: the mapping from WP to Rainbow should be as simple as possible
3. Unambiguous: there should be only one way (or very few ways) for a given document to be represented in Rainbow
4. Maintains all information that is essential for recognition of content
5. Simplicity

2. Limitations

Completeness in representation of WP data is not a goal. When one considers the maturity of today's WP technologies (many of which now include significant page-layout and drawing features) and the rate at which that industry continues to mature, one immediately realizes that any attempt towards completeness is doomed to failure.

Thus, we have decided *not* to set such goals as:

- support for recreation of the original source

---

[1] For complete information on the motivation behind Rainbow and the intended use for Rainbow documents, see the article by D. Sklar in <TAG>, January 94.

- native representation of equations
- support for "non-prosaic" documents (e.g. a WordPerfect representation of an employment-application form, with chains of underscores representing form blanks, etc.)
- retention or handling of WP "layout" features (e.g. multi-column)

Our goal is to handle a small, core set of features that are common to all WPs and that are the bread-and-butter of most prosaic documents. We acknowledge that there are conversion projects for which Rainbow is not suitable.

3. Overview of Changes

3.1 Deltas from Version 2.1 to Version 2.2

The HEAD element (used to store the title of a STRUCLVL) is now just a wrapper around one or more PARA elements.

The CLFTYPE element has been introduced, allowing representation of character-level formatting stencils. Thus, the CLF element now accepts a "clftype" attribute.

The NAMEDCLF element has been introduced, allowing representation of character sequences that have been given positive semantic identification via the use of character-level style stencils.

CLF nesting is now possible, but the use and depth of such recursion is limited.

ANCHOR, IDXTERM, and XREF content models repaired to match the original intended semantics.

The "rbowtab" SDATA entity has been added, intended for representation of tab characters found in WP textual content.

3.2 Deltas from Version 1 to Version 2.1

Each paragraph must have a paratype; thus:

- The STYINFO section is no longer optional.
- There must be a "default paratype" that is used for paragraphs that did not have a paratype in the original document.

Multi-segment paragraphs are handled quite differently:

- Paragraphs are no longer composed of multiple PARACONT elements; rather, each segment is an individual PARA element (with continuation segments so marked via an attribute).
- Tab characters are no longer considered to be segment delimiters.

A technique for attaching arbitrary WP-dependent attributes to any Rainbow object is now provided: SYSATTR.

Marking of original-source location has been made more generic: WPLOC has replaced PAGENUM.

The MARK element has been eliminated, because identification of marker objects is difficult to automate accurately and thus is better performed downstream (i.e. by Rainbow consumers).

The spec now provides information on the handling of empty paragraphs in the original document.

4. Glossary of Key Terms

paratype	Most WPs allow the author to specify that a paragraph is of a given type. In MSWord, each paragraph can be assigned a "style"; that is its paratype.

paralevel formatting attributes	Attributes that can only be assigned to a paragraph as a whole. Abbreviation: PLF Examples: space-before, left-indent, first-indent
charlevel formatting attributes	Attributes assigned to a contiguous range of text inside a paragraph, or assigned to a paragraph as a whole Abbreviation: CLF Examples: font-weight, font-slant
CLF-type	The charlevel equivalent of a paratype. Some WPs allow the author to create style definitions for sub-paralevel text.
override formatting	Applies only to PLF and CLF formatting for a paragraph that has a paratype. WPs that support paratypes allow the user to specify formatting for the paratype. This formatting definition is thus automatically applied to all paragraphs assigned that paratype. These WPs usually allow the user to "override" these paratype-level formatting attributes, on a per-paragraph basis.
non-soft page break	A page break that is due to an explicit request in the WP document. There are two types of non-soft page breaks: 1. Explicit page-break markers that are found in the WP document, lying between paralevel objects. 2. The point just before a paragraph whose formatting (override or paratype-based) specifies that it should start at the top of a new page.

5. The Annotated DTD

```
<!ELEMENT rainbow  - - (FILEINFO?,STYINFO,DOC)>
```

General information about the document.

```
<!ELEMENT fileinfo - O EMPTY>
<!ATTLIST fileinfo     origin   CDATA #IMPLIED
                       dtdver   CDATA #REQUIRED  -- Rev. number of Rainbow DTD
                                                    used for this instance -->
```

Third parties implementing "Rainbow Makers" should consult with EBT to register origin names; note that origin names should include the version of the WP (e.g. use "MIF-3" instead of just "MIF"). A multi-WP standard like RTF should be represented by a string that includes identification of the WP that generated it (e.g. "MacWord-RTF-*versionnum*" or "WinWord-RTF-*versionnum*").

The required "dtdver" describes the version/revision of the Rainbow DTD to which the document is intended to conform. For example, the DTD version described by this document is "2.2".

```
_____________________________________________
```

The formatting (PLF and CLF) attributes are present for one reason only: to allow recognition of content. The purpose of these attributes is ***not*** to facilitate recreation of the formatting or layout of the original WP document. Thus, the set of attributes is not complete.

Remember: a paralevel object can have both PLF and CLF attributes; but a subparalevel object can only have CLF attributes.

Here are the DTD specifications for the PLF and CLF attributes. Comments are used to specify the semantics.

```
<!ENTITY % boolean "NUMBER"  -- 1 means yes, 0 means no -->

<!ENTITY % plf-att -- Paragraph Level Formatting --
"left-indent CDATA #IMPLIED
      -- measured in points; absolute number; do not use += notation --
 right-indent CDATA #IMPLIED
      -- measured in points; absolute number; do not use += notation --
 first-indent CDATA #IMPLIED
      -- same as in DynaText: offset from the left-indentation --
 justification (LeftJust|CenterJust|RightJust|FullJust|InJust|OutJust) #IMPLIED
      -- FullJust means both left and right justification
         InJust/OutJust means justify towards binding/perimeter --
 line-spacing CDATA #IMPLIED
      -- same as in DT: distance between baselines --
 space-before CDATA #IMPLIED
 space-after CDATA #IMPLIED
      -- both of the above are as in DT: measured in pts --
 keep-with (Next|Prev) #IMPLIED
      -- keep with (next/prev) paragraph --
 tab-stops CDATA #IMPLIED
      -- colon-separated numbers; each number is a measurement (in points)
         of the tabstops distance from the left-indentation --
 keep-together %boolean; #IMPLIED
      -- does author demand that the paragraph not cross page boundaries? --
 pg-brk-before %boolean; #IMPLIED       -- page-break before --
 col-brk-before %boolean; #IMPLIED      -- column-break before --
 border NUMBER #IMPLIED
      -- value is sum of the appropriate members of this list:
                1 for top
                2 for right
                4 for bottom
                8 for left   --
">

<!ENTITY % clf-att -- Character Level Formatting --
"charset CDATA #IMPLIED
 font-family CDATA #IMPLIED
 font-size CDATA #IMPLIED
 font-weight (Medium|Bold) #IMPLIED
 font-slant (Roman|Ital)   #IMPLIED    -- Ital for both italic and oblique --
 score-location (Under|Over|Through) #IMPLIED
 score-type (Single|Double|Dotted) #IMPLIED
      -- score-type is ignored if score-location is not set --
 vertical-offset CDATA #IMPLIED
      -- as in DT: offset from the baseline, in points, - for sub, + for sup --
 foreground CDATA #IMPLIED
 background CDATA #IMPLIED
      -- If the generator can do so, it must specify color as a RGB
            hexadecimal numbers in this format:  #rrggbb
         If the generator is unable to do so, it can use an arbitrary
            representation (e.g. name, index number).
         Note that if the latter representation is used, it will not
            be possible for the Rainbow consumer to reproduce colors
            in its rendering facilities. --
      -- Note: if the text is not in a special color, don't set
               the foreground attribute. --
      -- Note: if the text is not on a special background, don't set
               the background attribute. --
      -- Example: use #888888 for background in the typical case of shaded
                  background for black-on-white printing --
 lowercase-display (SmallCaps|FullCaps)  #IMPLIED
      -- Note that this specifies how lowercase data is displayed;
         this does not describe the data itself. --
 outline %boolean; #IMPLIED
      -- Is each letter being displayed as a hollow outline? --
 change-bar (BarLeft|BarRight) #IMPLIED
 hidden %boolean; #IMPLIED
">

-------------------------------------------
```

The WP document's stylesheet must be stored in the STYINFO section. Rainbow supports both paragraph-level stencils (paratypes) and sub-paragraph level stencils (clftypes).

There must be at least one paratype in the STYINFO section, even if the original WP document has no stylesheet. This is because each PARA in a Rainbow document must have an assigned paratype, even if it was not assigned a paratype in the original WP document. A Rainbow generator is responsible for creating a default paratype named "#DEFAULT-PARATYPE" which should act as the paratype for paragraphs that were not assigned one in the original WP document.[2] The "#DEFAULT-PARATYPE" paratype should define "normal" PLF and CLF attributes; for example, it should set font-weight to Medium, font-slant to Roman, vertical offset to 0, etc.

However, CLFTYPEs are not required, because many old WPs (e.g. Word for Windows v2) did not support sub-paragraph-level style stencils.

Each PARATYPE must explicitly set ***each*** PLF and CLF attribute; this extreme requirement helps ensure that the Rainbow generator does not create PARAs that contain unnecessary override attributes.

However, a CLFTYPE should only set the CLF attributes that it explicitly controls. (Most CLFTYPES explicitly set only a very small number of attributes, e.g. font-weight or font-slant.)

```
<!ELEMENT styinfo  - - (PARATYPE+,CLFTYPE*)>

<!ELEMENT paratype - O EMPTY>
<!ATTLIST paratype     name     CDATA #REQUIRED
                       %plf-att; %clf-att;>

<!ELEMENT clftype - O EMPTY>
<!ATTLIST clftype     name     CDATA #REQUIRED
                      %clf-att;>
-------------------------------------------
```

The representation of the document's content.

```
<!ELEMENT doc      - - ( HEAD?,
                         (((PARA|ILLUS|TABLE|SYSOBJ)+,
                           STRUCLVL*)
                          |(STRUCLVL+ ) ) )

                       -- inclusion exceptions --
                       +(WPLOC|IDXTERM)          >
-------------------------------------------
```

Many word processors support representation of hierarchy through implication. For example, an MS Word document is essentially a flat list of paralevel objects, each of which can be assigned a paratype. Use of paratypes like "Heading 1", "Heading 2", etc. — applied to paralevel objects that represent section titles — allow this flat list to represent a hierarchy of arbitrary depth.

The implied hierarchy can be represented ***explicitly*** in the Rainbow via the STRUCLVL element. The process that generates Rainbow can watch for the title-bearer paratypes and use them to generate STRUCLVL elements. Each STRUCLVL element starts with a HEAD element, which is a wrapper around the Rainbow representation of the heading paragraph. (The model is "PARA+" because the Rainbow representation of a single WP paragraph can be multiple Rainbow paragraphs, as described later.)

If the original document does not contain title-bearer paragraphs, the generated Rainbow document will of course not contain any STRUCLVL elements. In such a case, it is up to the downstream process (the one that takes in Rainbow) to attempt to recognize title-bearers (by using the formatting information in the Rainbow).

```
<!ELEMENT struclvl - - (SYSATTRS?,HEAD,(PARA|ILLUS|TABLE|SYSOBJ)*,STRUCLVL*)>
<!ELEMENT head     - - (SYSATTRS?,PARA+)>
<!ATTLIST (struclvl|head)
             id CDATA #IMPLIED>

-------------------------------------------
```

The value of the paratype attribute must match the name attribute of exactly one PARATYPE element. As described previously, each PARA must be assigned a paratype. (See the previous discussion concerning the use of the special paratype "#DEFAULT-PARATYPE".)

---

2 Rainbow generators for WPs that require all paragraphs to have a paratype are not required to create this special paratype.

Paragraphs may not be nested. Embedded paralevel objects (like footnotes) are handled via the NOTE element described later.

Note that a PARA element will have PLF/CLF attributes *only* in cases of ***override***. Thus, in documents which were authored with heavy and consistent use of paratypes, the typical PARA element will have no formatting attributes. The authors of Rainbow generators are encouraged to be very careful to avoid unnecessary override attributes.

A ***single*** paragraph in the original WP document maps to ***one or more*** paragraphs in the Rainbow document. That is because any contiguous sequence of hard-return characters (explicit line breaks that are not intended to represent paragraph delimiters) in the WP paragraph ends the current PARA and begins a new "continuation PARA".[3] A continuation PARA must have a "continue" attribute of 1 and a non-zero "preced-hard-returns" attribute, as well as the "paratype" and formatting-override attributes. Note that a continuation PARA's set of formatting attributes (paratype and overrides) will always be identical to the set of formatting attributes that lies on its left sibling.[4]

```
<!ELEMENT para  - - (SYSATTRS?,PARACONT)>
<!ATTLIST para
                      paratype CDATA    #REQUIRED
                      %plf-att; %clf-att;
      -- If the para is a child of a head, the ID attribute should go
         on the head, not the para --
                      id       CDATA    #IMPLIED
      -- remaining attrs are set only if this element
         represents a "continuation" of a para in orig WP doc --
                      continue %boolean #IMPLIED
                      preced-hard-returns NUMBER #IMPLIED >

<!ENTITY % par_cont
         "#PCDATA|SYSOBJ|ANCHOR|XREF|AUTOGEN|GRAPHIC|EQN|NOTE">
<!ELEMENT paracont  - -
          (%par_cont;|NAMEDCLF|CLF|)+ >
```

Paragraphs that contain only whitespace data should be handled specially by the Rainbow generator, because many WP authors use blank paragraphs as a method for putting extra vertical space between two paralevel objects. A sequence of one or more "data-less" paragraphs should be recognized by the Rainbow generator and should not be transferred to the output Rainbow document; rather, the sequence should simply result in an override setting of the "space-before" attribute of the PARA immediately following it.

A sequence of data-less paragraphs that is immediately followed by a non-soft page break[5] should be ignored completely.

Note that Rainbow's support for non-Latin and multiple character sets is in the early design stage; thus, the Rainbow DTD is subject to change in the near future in order to address those issues more completely.

Any "symbol" or "upper-Latin" character must be represented in Rainbow documents via references to entities, using standard ISO public entity sets. This ensures portability of Latin-language Rainbow documents across Mac, PC, UNIX.

Discretionary hyphens in the original WP document (i.e. non-data "marks" that specify desired intra-word breaking points, thus providing hints to the WP's internal auto-hyphenation system) should be discarded by the Rainbow generator.

Each tab character in the original WP textual content should be transformed into "`&rbowtab;`" which is a reference to this SDATA entity defined in the Rainboow DTD:

---

[3] Thus, a single WP paragraph will always map to exactly one "first" PARA followed by zero or more "continuation" PARAs.

[4] Note that the use of the term "continuation paragraph" is problematic, for in some circles the term refers to the latter part of a paragraph that is "split in two" by an embedded table/graphic/etc.

[5] See the glossary at the beginning of this document.

```
<!ENTITY rbowtab SDATA "[rbowtab]">

-------------------------------------------
```

If an equation or graphic occurs at the paralevel (e.g. a direct child of a section container), it must be bundled as an ILLUS element. If it is embedded in a paralevel object, it is simply represented as a GRAPHIC or EQN. In the former case, it is possible that there is an associated caption for the illustration. In Rainbow, that caption would simply be a PARA that lies before or after the ILLUS.

Consider a graphic embedded in an MS Word document. In the RTF, the graphic's binary data is represented via ASCII characters, with leading attributes specifying the binary format type (e.g. "Macintosh PICT"). The Rainbow extractor should perform the necessary conversions to create a file containing the graphic and generate a GRAPHIC element with appropriate filename and format (e.g. "PICT") attributes.

Equations are simply another form of illustration object, but it is a bit more flexible in that the data can be stored either in a file or as CDATA in the EQN element itself. Consider an equation represented in MS Word's proprietary equation language. The Rainbow generator could place the equation (still in MS's proprietary equation-markup language) in a file and create a EQN element with format="msword-equ" and an appropriate setting for the filename attribute. Alternatively, the data could be placed in the EQNCONT element and the filename attribute removed. A third alternative is that the Rainbow generator convert the equation to the ISO-standard equation DTD and store that in the EQNCONT element and set format to "iso-dtd".

Note that the values for the format attribute have not been determined (the above examples used fictitious ones). Third parties implementing Rainbow generators should consult with EBT to "register" standard format names for both graphics and equations.

```
<!ELEMENT illus     - O (SYSATTRS?,(GRAPHIC|EQN))>
<!ATTLIST illus         %plf-att;
                        id       CDATA #IMPLIED>

<!ELEMENT graphic   - O (SYSATTRS?)>
<!ATTLIST graphic       format CDATA #REQUIRED
                        filename CDATA #REQUIRED>

<!ELEMENT eqn       - O (SYSATTRS?,EQNCONT?)>
<!ELEMENT eqcont    - - CDATA>
<!ATTLIST eqn           format CDATA #REQUIRED
                        filename CDATA #IMPLIED>

-------------------------------------------
```

CLF elements come in two forms:

1. A CLF element that sets its name attribute (referring to a CLFTYPE in the STYINFO section) is a "named CLF". It represents a contiguous sequence of characters (not crossing a paragraph boundary) that has been given a semanatic "type" by the author.
2. A CLF element that does not set its name attribute is an "unnamed CLF". It represents a sequence of characters that has been given override formatting without any indication of the semantics behind that special formatting.

A named CLF element may contain one or more nested unnamed CLFs. But a named CLF element may not contain another named CLF, and an unnamed CLF may not have any CLF children.

Thus, assuming a WP that does not support sub-para-level stencils, the sequence "Use ***Dyna*Text** for" would be represented this way:

```
Use <CLF font-weight=bold font-slant=ital>Dyna</><CLF font-weight=bold>Text</> for
```

```
<!ELEMENT namedclf  - -  (SYSATTRS?,(%par_cont;|CLF)+) >
<!ELEMENT clf       - -  (SYSATTRS?,(%par_cont;)+ )>

<!ATTLIST namedclf      clftype CDATA #REQUIRED
                        %clf-att;
                        ID CDATA #IMPLIED >
<!ATTLIST clf           %clf-att;
                        ID CDATA #IMPLIED >

-------------------------------------------
```

NOTE elements store the content of objects like footnotes, annotations and sidebars; anything that would be represented in an on-line document as a "revelatory". In some WPs, the contents of notes are not stored directly at the point where the

note is "referenced" in the text; but in Rainbow, the NOTE element must occur at the reference point. The Rainbow generator is responsible for performing any necessary "data movement".

```
<!ELEMENT note - - (SYSATTRS?,(PARA|ILLUS|TABLE|SYSOBJ)*)>

-------------------------------------------
```

Some word processors can auto-create "back-of-the-book" indexes. The author of the document is responsible for marking words and phrases that should appear in the index. Rainbow generators should mark these index entries using the IDXTERM element.

If part of the document prose is marked as an index term, the IDXTERM element is used with no attribute:

```
A <IDXTERM>manatee</> is a...
```

If the index term (e.g "mammals, interesting") is not actually represented in the prose, the IDXTERM element is used with its "term" attributes:

```
A manatee <IDXTERM term1=mammals term2=interesting> is a...
```

Note that three term attributes are available, allowing up to three levels of hierarchy in the index. Note also that if any term attribute is set on a particular IDXTERM element, the PCDATA content of that element is ignored.

Some WPs allow the author to specify that an index entry should reference a different entry (*Amphibian: see Mammal*); this is represented in the Rainbow via the "see" attribute. Still others support similar "see also" semantics, represented in the Rainbow via the "see-also" attribute. It is always legal to specify "see" and/or "see-also", regardless of whether the term is coming from the IDXTERM's PCDATA content or from the IDXTERM's "term..." attributes.

```
<!ELEMENT idxterm    - - (#PCDATA|CLF)+>
<!ATTLIST idxterm        term1 CDATA #IMPLIED
                         term2 CDATA #IMPLIED
                         term3 CDATA #IMPLIED
                         see CDATA #IMPLIED
                         see-also CDATA #IMPLIED>

-------------------------------------------
```

Some word processors support the specification of cross-references. I.e. the author can attach "bookmarks" (IDs) to certain points in the document, and reference those bookmarks arbitrarily throughout the document.

A Rainbow generator encountering a bookmark should do one of these actions:

- If the bookmark is intended to give an ID to an entire STRUCLVL, PARA, CLF, HEAD, ILLUS, or TABLE, the bookmark's ID should simply become the ID of the STRUCLVL, PARA, CLF, HEAD, ILLUS, or TABLE in the Rainbow document.
- If the bookmark is intended to give an ID to a chunk of text that is embedded in a paralevel object (e.g. a word or phrase inside a paragraph), that chunk of text should be contained within an ANCHOR element in the Rainbow document. (Note that if the chunk of text that is being marked crosses one or more paralevel boundaries, this use of ANCHOR is not possible. See the next bullet.)
- If the bookmark is intended to mark a specific point in the document, an ANCHOR element with no content should be inserted in the Rainbow document at that point.

There are three types of word-processor-based "cross-references" which refer to bookmarks:

- The author inserts a reference to a bookmark in order to request that the word-processor auto-generate a page number. For example:
  ```
  For more information, see the note on page [PAGEGEN-XREF TO BOOKMARK XX].
  ```
  The Rainbow generator should generate a no-content XREF element with type set to "xref-pagenum".
- The author inserts a reference to a bookmark ***that contains textual content*** in order to request that the word-processor copy that textual content into the document at the current point. For example:
  ```
  For more information, see the section called [TEXTCOPY-XREF TO BOOKMARK XX].
  ```
  The Rainbow generator should generate a no-content XREF element with type set to "xref-textcopy".
- The author inserts a link, because she is using a word processor that is modern enough to support hypertext navigation. The Rainbow generator should generate an XREF element with type set to "xref-link".

```
<!ELEMENT anchor     - - (#PCDATA|CLF)+>
<!ATTLIST anchor         ID CDATA #REQUIRED>
```

```
<!ELEMENT xref      - - (#PCDATA|CLF)+>
<!ATTLIST xref          REFID CDATA #REQUIRED
                        TYPE (xref-textcopy|xref-pagenum|xref-link) #REQUIRED>

-------------------------------------------
```

Most WPs support autonumbering; in the WP file, a special object is placed at the point where the auto-generation is to occur. Rainbow generators are responsible for performing any work necessary to compute the ***resultant text*** that would actually be generated by the WP. That resultant text should be placed inside an AUTOGEN element to record the fact that the text is not source material. The generator can optionally use system-dependent attributes to store information about the auto-generation algorithm.

```
<!ELEMENT autogen      - - (SYSATTRS?,#PCDATA)>

-------------------------------------------
```

Programs that eat Rainbow data (e.g. Rainbow analyzers, viewers, or up-convertors) may need to report problems found in the Rainbow data. The most efficient interface for reporting such diagnostics is to open the original WP document (using the original WP itself) and scroll the document to the place where the problem occurs. This is possible because many WP programs support inter-process communication protocols allowing an external application to perform "remote control" -- i.e. to request that the WP scroll the document to a specific point.

Of course, remote control can be implemented only if the Rainbow document contains "location markers" that allow a given point in the Rainbow document to be "mapped" to a location in the original WP document. The WPLOC element (which is allowed anywhere) is designed to store location markers.

The WPLOC element is declared EMPTY and contains two required attributes. The "human-addr" attribute is optional; it should store the location's "human-readable" address. The "wp-addr" attribute is required; it must store the location address that would be sent to the WP in order to request that the WP scroll to the corresponding location in the original document. There is a very good reason for the distinction between human and WP location addresses: page numbers might be displayed using Roman numerals (e.g. "xxii") but most WPs accept only data numerals (e.g. 22) for the purposes of remote-application control.

Some WP formats contain markers representing each page break (both "hard" explicit ones and "soft" computed ones). Rainbow generators for such formats should use the WPLOC to represent each page break.

Other WP formats do not contain sufficient information for determining the locations of all page breaks. Rainbow generators for such formats should either generate no WPLOCs at all, or should produce WPLOCs with alternative information (e.g. paragraph number). Note, however, that WPLOCs should not be generated at all if they do not store location addresses that are accepted by the WP for the purposes of remote control.

```
<!ELEMENT wploc       - O EMPTY>
<!ATTLIST wploc       wp-addr    CDATA #REQUIRED
                      human-addr  CDATA #IMPLIED>

-------------------------------------------
```

The SYSOBJ is used to represent WP-specific objects not covered by any standard Rainbow elements.

Third-parties building Rainbow generators should consult with EBT regarding each intended use of SYSOBJ.

```
<!ELEMENT sysobj    - O (datafld|txtfld)+>
<!ATTLIST sysobj      type CDATA #REQUIRED
                      id       CDATA #IMPLIED>
<!ELEMENT datafld   - - CDATA>
<!ELEMENT textfld   - - (#PCDATA|CLF)+>
<!ATTLIST (datafld|textfld)   type CDATA #REQUIRED>

-------------------------------------------
```

The SYSATTR allows storage of WP-specific attributes on many Rainbow objects, including (but not limited to) STRUCLVL, PARA, PARACONT, and CLF. Each SYSATTR element stores its name in an attribute and its value in its CDATA content.

Third-parties building Rainbow generators should consult with EBT regarding each intended use of SYSATTR.

```
<!ELEMENT sysattrs    - O (sysattr*)>
<!ELEMENT sysattr     - - CDATA>
```

```
<!ATTLIST sysattr        attrname CDATA #REQUIRED>

-------------------------------------------
```

The CALS DTD is the model we followed to create this simple DTD representation:

```
<!ELEMENT table    - O (tgroup+)>
<!ATTLIST table       %plf-att;
                      id CDATA #IMPLIED>
<!ELEMENT tgroup   - - (colspec*,spanspec*,thead?,tbody)>
<!ATTLIST tgroup      cols NUMBER #REQUIRED>
<!ELEMENT colspec  - O EMPTY>
<!ATTLIST colspec     colnum NUMBER #IMPLIED
                      colname NMTOKEN #IMPLIED
                      align (left|right|center) #IMPLIED
                      colwidth CDATA #IMPLIED>
<!ELEMENT spanspec - O EMPTY>
<!ATTLIST spanspec    namest NMTOKEN #REQUIRED
                      nameend NMTOKEN #REQUIRED
                      spanname NMTOKEN #REQUIRED
                      align (left|right|center) #IMPLIED>
<!ELEMENT thead    - - (row+)>
<!ELEMENT tbody    - - (row+)>
<!ELEMENT row      - - (entry+)>
<!ELEMENT entry    - - (PARA|ILLUS|SYSOBJ)*>
<!ATTLIST entry       colname NMTOKEN #IMPLIED
                      spanname NMTOKEN #IMPLIED>

-------------------------------------------
```

6. Acknowledgements

The primary author of the Rainbow DTD is David Sklar, who is grateful for the contributions made by Ludo van Vooren, Bobby Koritala, Brian Adler, Steve deRose, Lou Reynolds, and the many reviewers and supporters.

7. How to keep in touch with Rainbow

EBT is making the Rainbow DTD and various Rainbow-Maker software freely available to the public via an anonymous FTP server directory: `ftp.ebt.com:/pub/nv/DTD/rainbow`

If you wish to be notified by email when information in the FTP directory is modified and when significant Rainbow-related events occur, subscribe to the Rainbow information service by emailing to: `rainbow@ebt.com`