The hOCR Embedded OCR Workflow and Output Format

Thomas Breuel (editor)

- December 2007 - initial release
- March 2010 - bug fixes, clarifications

1 Rationale

The purpose of this document is to define an open standard for representing OCR results. The goal is to reuse as much existing technology as possible, and to arrive at a representation that makes it easy to reuse OCR results.

2 Getting Started

This document describes many tags and a lot of information that can be output. However, getting started with hOCR is easy: you only need to output the tags and information you actually want to. For example, just outputting **ocr_line** tags with bounding boxes is already very useful for many applications. Just start simple and add more output information as the need arises.

3 Terminology and Representation

This document describes a representation of various aspects of OCR output in an XML-like format. That is, we define as set of tags containing text and other tags, together with attributes of those tags. However, since the content we are representing is formatted text, However, we are not actually using a new XML for the representation; instead embed the representation in XHTML (or HTML) because XHTML and XHTML processing already define many aspects of OCR output representation that would otherwise need additional, separate and ad-hoc definitions. These aspects include:

- standard representations for common logical structuring elements, including section headings, citations, tables, emphasis, line breaks, quotations, citations, and preformatted text
- standard representations for fonts, embedded images, embedded vector graphics, tables, languages, writing direction, colors
- standard representations for geometric layout and positioning
- output files that are understood without any further modification by widely used

viewers (browsers), editors, conversion tools, and indexing tools

- libraries for parsing and generating the content
- support for document metadata

We are embedding this information inside HTML by encoding it within valid tags and attributes inside HTML; We are going to use the terms “elements” and “properties” for referring to embedded markup.
Elements are defined by the class= attribute on an arbitrary HTML tag. All elements in this format have a class name of the form “ocr…_…”.
Properties are defined by putting information into the “title=” attribute of an HTML tag. Properties in title attributes are of the form “name values…”, and multiple properties are separated by semicolons.
Here is an example:

```
<div class="ocr_page" id="page_1">
  <div class="ocr_carea" id="column_2" title="bbox 313 324 733 1922">
    <div class="ocr_par" id="par_7"> ... </div>
    <div class="ocr_par" id="par_19"> ... </div>
  </div>
</div>
```

The following properties can apply to most elements (where it makes sense):

- bbox x0 y0 x1 y1 – the bounding box of the element relative to the binarized document image
  - use x_bboxes below for character bounding boxes
  - do not use bbox unless the bounding box of the layout component is, in fact, rectangular
  - some non-rectangular layout components may have rectangular bounding boxes if the non-rectangularity is caused by floating elements around which text flows

The following properties can apply to most elements but should not be used unless there is no alternative:

- poly x0 y0 x1 y1 … – a closed polygon for elements with non-rectangular bounds
  - this property must not be used unless there is no other way of representing the layout of the page using rectangular bounding boxes, since most tools will simply not have the capability of dealing with non-rectangular layouts
  - note that the natural and correct representation of many non-rectangular layouts is in terms of rectangular content areas and rectangular floats
  - documents using polygonal borders anywhere must indicate this in the metadata
  - documents should attempt to provide a reasonable bbox equivalent as well
- order n – the reading order of the element (an integer)
  - this property must not be used unless there is no other way of representing the reading order of the page by element ordering within the page, since many tools will not be able to deal with content that is not in reading order
  - presence presence must be declared in the document meta data

The following property relates the flow between multiple ocr_carea elements, and between ocr_carea and ocr_linear elements.

- cflow s – the content flow on the page that this element is a part of
  - s must be a unique string for each content flow

    - must be present on ocr_carea and ocrx_block tags when reading order is attempted and multiple content flows are present
    - presence must be declared in the document meta data

4 Logical Structuring Elements

We recognize the following logical structuring elements:

```
  * ocr_document
    * ocr_linear
      * ocr_title
      * ocr_author
      * ocr_abstract
      * ocr_part [H1]
      * ocr_chapter [H1]
        * ocr_section [H2]
          * ocr_sub*section [H3,H4]
            * ocr_display
            * ocr_blockquote [BLOCKQUOTE]
            * ocr_par [P]
```

These logical tags have their standard meaning as used in the publishing industry and tools like LaTeX, MS Word, and others.

The standard HTML tags given in brackets specify the preferred HTML tags to use with those logical structuring elements, but it may not be possible or desirable to actually chose those tags (e.g., when adding hOCR information to an existing HTML output routine).

For all of these elements except "ocr_linear", there exists a natural linear ordering defined by reading order ("ocr_linear" indicates that the elements contained in it have a linear ordering). At the level of "ocr_linear", there may not be a single distinguished order. A common example of "ocr_linear" is a newspaper, in which a single newspaper may contain many linear, but there is no unique reading order for the different linear. OCR evaluation tools should therefore be sensitive to the order of all elements other than ocr_linear.

Tags must be nested as indicated by nesting above, but not all tags within the hierarchy need to be present.

Textual information like section numbers and bullets must be represented as text inside the containing element.

Documents whose logical structure does not map naturally onto these logical structuring elemetns must not use them for other purpose.

Image captions may be indicated using the "ocr_caption" element; such an element refers to the image(s) contained within the same float, or the immediately adjacent image if both the image and the "ocr_caption" element are in running text.

5 Typesetting Related Elements

The following typesetting related elements are based on a typesetting model as found in most typesetting systems, including XSL:FO, (La)TeX, OpenOffice, and Microsoft Word.

In those systems, each page is divided into a number of areas. Each area can either be a part of the body text (or multiple body texts, in the case of newspaper layouts). The content of the areas derives from a linear stream of textual content, which flows into the areas, filling them linewise in their preferred directions.

Overlayed onto the page is a set of floating elements; floating elements exist outside the normal reading order. Floating elements may be introduced by the textual content, or they may be related to the page itself (anchoring is a logical property). In typesetting systems, floating elements may be anchored to the page, to paragraphs, or to the content stream. Floating pelements can overlap content areas and render on top of or under content, or they can force content to flow around them. The default for floating elements in this spec is that their anchor is undefined (it is a logical property, not a typesetting property), and that text flows around them. Note that with rectangular content areas and rectangular floats, already a wide variety of non-rectangular text shapes can be realized.
[Issue: there is currently no way of indicating anchoring or flow-around properties for floating elements; properties need to be defined for this.]
The typesetting related elements therefore are:

- ocr_page
  - ocr_carea ("ocr content area" or "body area"; used to be called ocr_column)
    - ocr_line [SPAN]
  - (floats)
  - ocr_separator (any separator or similar element)
  - ocr_noise (any noise element that isn't part of typesetting)

The ocr_page element must be present in all hOCR documents. The following properties should be present:

- bbox
  - the bounding box of the page; for pages, the top left corner must be at (0,0), so a typical page bounding box will look like "bbox 0 0 2300 3200"
- image imagefile
  - image file name used as input
  - syntactically, must be a UNIX-like pathname or http URL (no Windows pathnames)
  - may be relative
  - cannot be resolved to the actual file in general (e.g., if the hOCR file becomes separated from the image fiels)
  - if the hOCR file is present in a directory hierarchy or file archive, should resolve to the corresponding image file
- imagemd5 checksum
  - MD5 fingerprint of the image file that this page was derived from
  - allows re-associating pages with source images
- ppageno n
  - the physical page number
  - the front cover is page number 0
  - should be unique
  - must not be present unless the pages in the document have a physical ordering
  - must not be present unless it is well defined and unique
- lpageno string
  - the logical page number expressed on the page
  - may not be numerical (e.g., Roman numerals)
  - usually is unique
  - must not be present unless it has been recognized from the page and is unambiguous

The following properties may be present:

- scan_res x_res y_res
    - scanning resolution in DPI
- x_scanner string
    - a representation of the scanner
- x_source string
    - an implementation-dependent representation of the document source
    - could be a URL or a /gfs/ path
    - offsets within a multipage format (e.g., TIFF) may be represented using additional strings or using URL parameters or fragments
    - examples
        - x_source /gfs/cc/clean/012345678911 17
        - x_source http://pageserver/012345678911&page=17

The ocr_carea elements should appear reading order unless this is impossible because of some other structuring requirement If the document contains multiple ocr_linear streams, then each ocr_carea must indicate which stream it belongs to.
In typesetting systems, content areas are filled with “blocks”, but most of those blocks are not recoverable or semantically meaningful. However, one type of block is visible and very important for OCR engines: the line. Lines are typesetting blocks that only contain glyphs (“inlines” in XSL terminology).

They are represented by the ocr_line area. In addition to the standard properties, the ocr_line area supports the following additional properties:

- hardbreak n
    - a zero (default) indicates that the end of the line is not a hard (explicit) line break, but a break due to text flow
    - a one indicates that the line is a hard (explicit) line break

Any special characters representing the desired end-of-line processing must be present inside the ocr_line element. Examples of such special characters are a soft hyphen (”-”), a hard line break (”<br>”), or whitespace (” ”) for soft line breaks.
Note that for many documents, the actual ground truth careas are well-defined by the document style of the original document before printing and scanning. From a single page, the careas of the original document style cannot be recovered exactly. However, the partition of a document by ocr_carea for an individual page shall be considered correct relative to ground truth if (1) all the text contained in a ground truth carea is fully contained within a single ocr_carea, (2) no text outside a ground truth carea is contained within an ocr_carea, and (3) the ocr_careas appear in the same order as the text flow relationships between the ground truth careas.

The following floats are defined:

- ocr_float
    - ocr_separator
    - ocr_textfloat
    - ocr_textimage
    - ocr_image
        - ocr_linedrawing – something that could be represented well and naturally in a vector graphics format like SVG (even if it is actually represented as PNG)

        - ocr_photo – something that requires JPEG or PNG to be represented well
    - ocr_header
    - ocr_footer
    - ocr_pageno
    - ocr_table

Floats should not be nested.

6 Inline Representations

There is some content that should behave and flow like text

- ocr_glyph – an individual glyph represented as an image (e.g., an unrecognized character)
    - must contain a single IMG tag, or be present on one
- ocr_glyphs – multiple glyphs represented as an image (e.g., an unrecognized word)
    - must contain a single IMG tag, or be present on one
- ocr_dropcap – an individual glyph representing a dropcap
    - may contain text or an IMG tag; the ALT of the image tag should contain the corresponding text
- ocr_glyphs – a collection of glyphs represented as an image
    - must contain a single IMG tag, or be present on one
- ocr_chem – a chemical formula
    - must contain either a single IMG tag or ChemML markup, or be present on one
- ocr_math – a mathematical formula
    - must contain either a single IMG tag or MathML markup, or be present on one

Mathematical and chemical formulas that float must be put into an ocr_float section.
Mathematical and chemical formulas that are "display" mode should be put into an ocr_display section.
Non-breaking spaces must be represented using the HTML nbsp; entity.
Soft hyphens must be represented using the HTML shy; entity.
Different space widths should be indicated using HTML and ensp, emsp, thinsp, zwnj, zwj.
The HTML lrm and rlm entities (indicating writing direction) must not be used; all writing direction changes must be indicated with tags.
Other superscripts and subscripts must be represented using the HTML <SUP> and <SUB> tags, even if special Unicode characters are available.
Furigana and similar constructs must be represented using their correct Unicode encoding.

7 Character Information

Character-level information may be put on any element that contains only a single "line" of text; if no other layout element applies, the ocr_cinfo element may be used.

- cuts c1 c2 c3 …
    - character segmentation cuts (see below)

    - there must be a bbox property relative to which the cuts can be interpreted
- nlp c1 c2 c3 …
    - estimate of the negative log probabilities of each character by the recognizer

For left-to-write writing directions, cuts are sequences of deltas in the x and y direction; the first delta in each path is an offset in the x direction relative to the last x position of the previous path. The subsequent deltas alternate between up and right moves. Assume a bounding box of (0,0,300,100); then

```
cuts("10 11 7 19") =
[ [(10,0),(10,100)], [(21,0),(21,100)], [(28,0),(28,100)], [(47,0),(47,100)]
]
cuts("10,50,3 11,30,-3") =
    [ [(10,0),(10,50),(13,50),(13,100)], [(21,0),(21,30),(18,30),(18,100)] ]
```

Here is an example:

```
<span class="ocr_cinfo" title="bbox 0 0 300 100; nlp 1.7 2.3 3.9 2.7; cuts 9
11 7,8,-2 15 3">hello</span>
```

Cuts are between all codepoints contained within the element, including any whitespace and control characters. Simply use a delta of 0 (zero) for invisible codepoints.

Writing directions other than left-to-right specify cuts as if the bounding box for the element had been rotated by a multiple of 90 degrees such that the writing direction is left to right, then rotated back.

It is undefined what happens when cut paths intersect, with the exception that a delta of 0 always corresponds to an invisible codepoint.

8 OCR Engine-Specific Markup

A few abstractions are used as intermediate abstractions in OCR engines, although they do not have a meaning that can be defined either in terms of typesetting or logical function. Representing them may be useful to represent existing OCR output, say for workflow abstractions.

Common suggested engine-specific markup are:

- ocrx_block
    - any kind of "block" returned by an OCR system
    - engine-specific because the definition of a "block" depends on the engine
- ocrx_line
    - any kind of "line" returned by an OCR system that differs from the standard ocr_line above
    - might be some kind of "logical" line
- ocrx_word
    - any kind of "word" returned by an OCR system
    - engine specific because the definition of a "word" depends on the engine

The meaning of these tags is OCR engine specific. However, generators should attempt to ensure the following properties:

- an ocrx_block should not contain content from multiple ocr_careas
- the union of all ocrx_blocks should approximately cover all ocr_careas
- an ocrx_block should contain either a float or body text, but not both
- an ocrx_block should contain either an image or text, but not both
- an ocrx_line should correspond as closely as possible to an ocr_line
- ocrx_cinfo should nest inside ocrx_line
- ocrx_cinfo should contain only x_conf, x_bboxes, and cuts attributes

The following properties are defined:

- x_font s
  - OCR-engine specific font names
- x_fsize n
  - OCR-engine specific font size
- x_boxes b1x0 b1y0 b1x1 b1y1 b2x0 b2y0 b2x1 b2y1 …
  - OCR-engine specific boxes associated with each codepoint contained in the element
  - note that the `bbox` property is a property for the bounding box of a layout element, not of individual characters
  - in particular, use `<span class="ocr_cinfo" title="x_bboxes ....">`, not `<span class="ocr_cinfo" title="bbox ...">`
- x_confs c1 c2 c3 …
  - OCR-engine specific character confidences
- x_wconf n
  - OCR-engine specific confidence for the entire contained substring

9 Font, Text Color, Language, Direction

OCR-generated font and text color information is encoded using standard HTML and CSS attributes on elements with a class of ocr_... or ocrx_... Language and writing direction should be indicated using the HTML standard attributes lang= and dir=, or alternatively can be indicated as properties on elements.

OCR information and presentation information can be separated by putting the CSS info related to the CSS in an outer element with an ocr_ or ocrx_ class, and then overriding it for the presentation by nesting another SPAN with the actual presentation information inside that:

<span class="ocr_cinfo" style="*ocr style*"><span style="*presentation style*"> ... </span></span>

The CSS3 text layout attributes can be used when necessary. For example, CSS supports writing-mode, direction, glyph-orientation ISO15924-based script, text-indent, etc.

10 Alternative Segmentations / Readings

Alternative segmentations and readings are indicated by a SPAN with “class=alternatives”. It must contains INS and DEL elements. The first contained element should be INS and represent the most probably interpretation, the subsequent ones DEL. Each INS and DEL element should have “class=alt” and a property of either “nlp” or “x_cost”. These SPAN, INS, and DEL tags can nest arbitrarily.

Example:

```
<SPAN class="alternatives">
<INS class="alt" title="nlp 0.3">hello</INS>
<DEL class="alt" title="nlp 1.1">hallo</DEL>
</SPAN>
```

Whitespace within the SPAN but outside the contained INS/DEL elements is ignored and should be inserted to improve readability of the HTML when viewed in a browser.

11 Grouped Elements and Multiple Hierarchies

The different levels of layout information (logical, physical, engine-specific) each form hierarchies, but those hierarchies may not be mutually compatible; for example, a single ocr_page may contain information from multiple sections or chapters. To represent both hierarchies within a single document, elements may be grouped together. That is, two elements with the same class may be treated as one element by adding a "groupid *identifier*" property to them and using the same identifier.

Grouped elements should be logically consistent with the markup they represent; for example, it is probably not sensible to use grouped elements to interleave parts of two different chapters. Therefore, grouped elements should usually be adjacent in the markup.

Applications using hOCR may choose to manipulate grouped elements directly, but the simplest way of dealing with them is to transform a document with grouped elements into one without grouped elements prior to further processing by first removing tags that are not of interest for the subsequent processing step, and then collapsing grouped elements into single elements. For example, output that contains both logical and physical layout information, where the logical layout information uses grouped elements, can be transformed by removing all the physical layout information, and then collapsing all split ocr_chapter elements into single ocr_chapter elements based on the groupid. The result is a simple DOM tree. This transformation can be provided generically as a pre-processor or Javascript.

The presence of grouped elements does not need to be indicated in the header; when it affects their operations, hOCR processors should check for the presence of grouped elements in the output and fail with an error message if they cannot correctly process the hOCR information.

12 Capabilities

Any program generating files in this output format must indicate in the document metadata what kind of markup it is capable of generating. This includes listing the exact set of markup sections that the system could have generated, even if it did not actually generate them for the particular document.

The capability to generate specific properties is given by the prefix ocrp_…; the important properties are:

- ocrp_lang – capable of generating lang= attributes
- ocrp_dir – capable of generating dir= attributes
- ocrp_poly – capable of generating polygonal bounds
- ocrp_font – capable of generating font information (standard font information)
- ocrp_nlp – capable of generating nlp confidences

The capability to generate other specific embedded formats is given by the prefix ocr_embeddedformat_<formatname>.
If an OCR engine represents a particular tag but cannot determine reading order for that tag, it must must specify a capability of ocr_<tag>_unordered.
If a document lists a certain capabilities but no element or attribute is found that corresponds to that capability, users of the document may infer that the content is absent in the source document. If a capability is not listed, the corresponding element or attribute must not be present in the document.

13 Profiles

hOCR provides standard means of marking up information, but it does not mandate the presence or absence of particular kinds of information. For example, an hOCR file may contain only logical markup, only physical markup, or only engine-specific markup. As a result, merely knowing that OCR output is hOCR compliant doesn't tell us whether that file is actually useful for subsequent processing.

OCR systems can use hOCR in various different ways internally, but we will eventually define some common profiles that mandate what kinds of information needs to be present in particular kinds of output. Of particular importance are:

- **physical layout profile**: OCR output in XHTML format with a defined set of common physical layout markup capabilities (page, carea, floats, line). Logical layout may be present as well, but the document tree structure must represent the physical layout structure, with logical layout elements split and grouped as needed.
- **logical layout profile**: OCR output in XHTML format with a defined set of common logical layout markup capabilities (linear, chapter, section, subsection). Physical layout may be present as well, but the document tree structure must represent the logical layout structure, with logical layout elements split and grouped as needed.

Other possible profiles might be defined for specific engines or specific document classes:

- common commercial OCR output (e.g., Abbyy)
    - ocr_page
    - ocrx_block, ocrx_line, ocrx_word
    - ocrp_lang
    - ocrp_font
- book target
    - all logical structuring elements (as applicable), except ocr_linear
    - ocr_page
- newspaper target

    - all logical structuring elements (as applicable)
    - articles map on ocr_linear
    - ocr_page

14 Required Meta Information

The OCR system is required to indicate the following using meta tags in the header:

- name=ocr-system content="name version"
- name=ocr-capabilities content=capabilities
    - see the capabilities defined above

The OCR system should indicate the following information

- name=ocr-number-of-pages content=number-of-pages
- name=ocr-langs content=languages-considered-by-ocr
    - use ISO 639-1 codes
    - value may be "unknown"
- name=ocr-scripts content=scripts-considered-by-ocr
    - use ISO 15924 letter codes
    - value may be "unknown"

15 HTML Markup

The HTML-based markup is orthogonal to the hOCR-based markup; that is, both can be chosen independent of one another. The only thing that needs to be consistent between the two markups is the text contained within the tags. hOCR and other embedded format tags can be put on HTML tags, or they can be put on their own DIV/SPAN tags.
There are many different choices possible and reasonable for the HTML markup, depending on the use and further processing of the document. Each such choice must be indicated in the meta data for the document.
Many mappings derived from existing tools are quite similar, and most follow the restrictions and recommendations below already without further modifications.
Depending on the particular HTML markup used in the document, the document is suitable for different kinds of processing and use. The formats have the following intents:

- html_none: straightforward equivalent of Goodoc or XDOC
- html_ocr: straightforward recording of commercial OCR system output
- html_absolute: target format for services like Google's View as HTML
- html_xytable: target format for layout-preserving on-screen document viewing
- html_simple: target format for convenient on-line viewing and intermediate format for indexing

As long as a format contains the hOCR information, it can be reprocessed by layout analysis software and converted into one of the other formats. In particular, we envision layout analysis tools for converting any hOCR document into html_absolute, html_xytable, and html_simple. Furthermore, internally, a layout analysis system might use html_xytable as an intermediate format for converting hOCR into html_simple.

15.1 Restrictions on HTML Content

To avoid problems, any use of HTML markup must follow the following rules:

- HTML content must not use class names that conflict with any of those defined in this document (“ocr_*”)
- HTML content must not use the title= attribute on any element with an ocr_* class for any purposes other than encoding OCR-related properties as described in this document

15.2 Recommendations for Mappings

When possible, any mapping of logical structure onto HTML should try to follow the following rules:

- the mapping should be “natural”–similar to what an author of the document might have entered into a WYSIWYG content creation tool
- text should be in reading order
- all tags should be used for the intended purpose (and only for the intended purpose) as defined in the HTML 4 spec
- floats are contained in DIV elements with a style that includes a float attribute
- repeating floating page elements (header/footer) should be repeated and occur in their natural location in reading order (e.g., between pages)
- embedded images and SVG should be contained in files in the same directory (no ”/” in the URL) and embedded with IMG and EMBED tags, respectively

Specifically

- <EM> and <STRONG> should represent emphasis, and are preferred to <B>, <I>, and <U>
- <B>, <I>, and <U> should represent a change in the corresponding attribute for the current font (but an OCR font specification must still be given)
- <P> should represent paragraph breaks
- <BR> should represent explicit linebreaks (not linebreak that happen because of text flow)
- <H1>, …, <H6> should represent the logical nesting structure (if any) of the document
- <A> should represent hyperlinks and references within the document
- <BLOCKQUOTE> should represent indented quotations, but not other uses of indented text.
- <UL>, <OL>, <DL> should represent lists

- <TABLE> should represent tables, including correct use of the <TH> tag

If necessary, the markup may use the following non-standard tags:

- <NOBR> to indicate that line breaking is not permitted for the enclosed content
- <WBR> to indicate that line breaking is permitted at that location

15.2.1 html_none

The simplest HTML markup for hOCR formats contains no logical markup at all; it is simply a collection of DIV and SPAN elements with associated hOCR information. Note that such documents can still be rendered visually through the use of CSS.

15.2.2 html_simple

This is a format that follows the restrictions and recommendations above, and only uses the following tags:

- <H1> … <H6>
- <P>, <BR>
- <B>, <I>, and <U> for appearance changes (bold, italic, underline)
- <FONT> for any other appearance changes
- <A>
- <DIV> with a float style for floats
- <TABLE> for tables
- <IMG> for images
- all SVG must be externally embedded with the <EMBED> tag
- the use of other embedded formats is permitted
- all other uses of <DIV>, <SPAN>, <INS>, and <DEL> only for hOCR tags or other embedded formats (hCard, …)

15.2.3 html_ocr_<engine>

The HTML markup produced by default by the OCR engine for the given document. Examples of possible values are:

- html_ocr_finereader_8
- html_ocr_textbridge_11
- html_ocr_unknown – the HTML was generated by some OCR engine, but it's unknown which one

15.2.4 html_absolute_<element>

The HTML represents absolute positioning of elements on each page. The possible subformats are:

- html_absolute_cols – absolute positioning of cols
- html_absolute_pars – absolute positioning of paragraphs
- html_absolute_lines – absolute positioning of lines
- html_absolute_words – absolute positioning of words
- html_absolute_chars – absolute positioning of characters

Google Science Search and Google "View as HTML" both use html_absolute_lines; this is probably the most reasonable choice for approximating the appearance of the original document.

15.2.5 html_xytable_absolute

The HTML is a table that gives the XY-cut layout segmentation structure of the page in tabular form. Note that in this format, text order does not necessarily correspond to reading order.
The format must contain one TABLE of class ocr_xycut representing each page. The TABLE structure must represent the absolute size of the original page element. The markup of the content of the table itself is as in html_simple.

15.2.6 html_xytable_relative

The page representation is as in html_xytable_absolute, but table element sizes are expressed relative (percentages).

15.2.7 html_<processor>

The HTML represents markup that follows the mappings of the given document processor to HTML. Note that the document doesn't actually need to have been constructed in the processor and that the processor doesn't need to have been used to generate the HTML. For example, the html_latex2html tag merely indicates that, say, a scanned and ocr'ed article uses the same conventions for logical markup tags that an equivalent article actually written in LaTeX and actually converted to HTML would have used. Possible subformats are:

- html_latex2html
- html_msword – HTML mapping generated by "Save As HTML"
- html_ooffice – HTML mapping generated by "Save As HTML"
- html_docbook_xsl – HTML mapping generated by official XSL style sheets

16 Document Meta Information

For document meta information, use the Dublin Core Embedding into HTML. See also Citation Guidelines for Dublin Core

17 Sample Usage

The HTML format described here may seem fairly complicated and difficult to parse, but because there are lots of tools for manipulating HTML documents, they're actually pretty easy to manipulate. Here are some examples:

```
import libxml2,re,os,string

# convert the HTML to XHTML (if necessary)

os.system("tidy -q -asxhtml < page.html > page.xhtml 2> /dev/null")

# parse the XML

doc = libxml2.parseFile('page.xhtml')

# search all nodes having a class of ocr_line

lines = doc.xpathEval("//*[@class='ocr_line']")

# a function for extracting the text from a node

def get_text(node):
    textnodes = node.xpathEval(".//text()")
    s = string.join([node.getContent() for node in textnodes])
    return re.sub(r'\s+',' ',s)

# a function for extracting the bbox property from a node
# note that the title= attribute on a node with an ocr_ class must
# conform with the OCR spec

def get_bbox(node):
    data = node.prop('title')
    bboxre = re.compile(r'\bbbox\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)')
    return [int(x) for x in bboxre.search(data).groups()]

# this extracts all the bounding boxes and the text they contain
# it doesn't matter what other markup the line node may contain

for line in lines:
    print get_bbox(line),get_text(line)
```

Note that the OCR markup, basic HTML markup, and semantic markup can co-exist within the same HTML file without interfering with one another.