

HEML - Human Editable Markup Language (yet another)



AUTHOR	Sébastien Devaux
ABSTRACT	HEML is a lightweight markup language designed to be as versatile and generic than XML but also as easy to edit than markdown using only simple text file editor.
KEYWORDS	
CONTEXT	Component heml-1.0.10
PROCESSED	2022-01-02 12:25:42+01:00 / sdevaux@ummon

ISSUE / REVISION STATUS RECORD		
Edition	Date	Changes summary
3	2021-12-20	Document layout. Added meta commands (§ 3.3): script (§ 3.8), dictionary (dic & k, § 3.9).
2	2020-11-21	Added introduction to extract and exec meta commands (§ 3.5 § 3.6). Added markdown in the final discussion about the alternate file formats (§ 4.1).
1	2012-12-02	First issue

Summary

1	Presentation	4
1.1	Why HEML	4
1.2	Yet another markup language (that is not YAML)	4
1.3	Drawbacks	4
2	Java implementation	4
2.1	Links	4
2.2	Implementation status	4
2.3	Changes requests and bug tracking	5
2.4	Considered additional features	5
3	More features, XML mapping and examples	5
3.1	Elements and attributes	5
3.2	Paragraphs, indentations and bullets	5
3.3	Parser meta commands	5
3.4	Includes	6
3.5	Files Extracts	6
3.6	Insert a command output	7
3.7	Tables	8
3.8	Lua scripting	10
3.9	Dictionary and keyword substitution	10
3.10	Miscellaneous features	11
4	Similar formats discussion	11
4.1	Markdown	11
4.2	Wiki markups	11
4.3	YAML	11
4.4	txt2tags	12
4.5	JSON	12
4.6	XML	12

1 Presentation

1.1 Why HEML

XML files, while growing, can quickly become very difficult to edit using an average text editor or even a more advanced software development advanced editor. Graphical XML editors are very efficient to update a few attributes or elements but are not comfortable for long writing sessions.

Rich editors and modern word processing applications are very efficient to edit some well defined XML formats such as docbook and xhtml but are quite difficult to configure to any user format. Wiki formats (wikitext, confluence, and others) are by designed efficient for on line ASCII editing then easy to handle with any text editor. But those formats are not as generic than XML and are not very helpful for anything else than document editing.

HEML design is inspired from wikitext but targets to be fonctionnaly equivalent to XML and allow editing of any kind of data, not only documents for human readers.

More than just a document format, combination of the include and table feature will help to migrate legacy data format (for now limited to ASCII format) to XML. Then the HEML processor performs as a revers XLS processor able to support easy translation from ASCII more or less delimited files to XML.

Because readable, it is as easy to track change, merge, than programming language source code, and software development tools can support HEML management, in particular build tools (make) and SCM (subversion, git, ...).

1.2 Yet another markup language (that is not YAML)

Main features are:

- More concise, then less characters to type
- As generic as XML: can express rich structured (hierarchical) information in a same manner.
- Handles simple layouts (paragraph, indents and bullets) as wiki text.
- Big documents are still readable directly from ASCII tools. Consider using your preferred syntax highlighter to gain more readability (a vim syntax file example is include in source HEML distribution).

1.3 Drawbacks

While processing, some layout information may be lost. Then, after automatic processing, it is not always possible to restore a document that is still optimised for human edition.

In a same way, some layout and special features are lost when translating document to XML format and the exact reverse process can't be realized.

Those two drawbacks make HEML a human editable Only markup language.

2 Java implentation

2.1 Links

- [Download¹](#) .
- [Browse source code²](#) .

2.2 Implementation status

The main implemented features are:

- HEML to XML command line filter: simple executable to process files or standard output and output streams, including or not an XSL stylesheet to be chained to the HEML format processing.
- SAX like callback API: use HEML as your native data format.

¹<http://www.eduvax.net/dist/noarch/heml-1.0.10.jar>

²<https://github.com/seduvax/heml/>

2.3 Changes requests and bug tracking

HEML's change requests are managed and track using the ABS' charm feature. This mean you can get issues related to the current branch going into the project's sources root directory and invoke `make crls`. For more details about ABS, the tool used to build heml, see its [page at github](#)³.

2.4 Considered additional features

- Improve error handling and reporting.
- Special characters handling
- Improve HEML writer (try to get closer to some reversible HEML/XML translation)
- Android port including XSL processor to turn any android device to a rich notepad producing publishable structured data.
- Advanced tables: add fixed size field without delimiter and binary file support to the table feature. This will turn HEML to a generic legacy data format to XML converter.

3 More features, XML mapping and examples

3.1 Elements and attributes

HEML	XML
<pre> 1 {root 2 {elem %attr=value} 3 {elem %attr=value %attr2=value2 4 {subelem text} 5 } 6 }</pre>	<pre> 1 <root> 2 <elem attr="value"/> 3 <elem attr="value" attr2="value2"> 4 <subelem>text</subelem> 5 </elem> 6 </root></pre>

3.2 Paragraphs, indentations and bullets

HEML	XML
<pre> 1 {section %title=paragraph layout example 2 paragraph 3 paragraph 4 - bullet 5 - bullet 6 - sub bullet 7 }</pre>	<pre> 1 <section title="paragraph_layout_example"> 2 <p>paragraph</p> 3 <p>paragraph</p> 4 bullet 5 bullet 6 7 bullet 8 9 </section></pre>

3.3 Parser meta commands

Element's names starting with '?' character are handled as parser command. Currently supported commands are:

- `set` : changes parser settings, possible related attributes are:

³<https://github.com/seeduvax/AcrobatomaticBuildSystem>

- **encoding** : specify which character encoding to use to parse the file.
- **tab** : defines the character count for one tab character while text computing indentation levels (see paragraph layout examples above).
- **line** : when set to **wrap** , activated the line wrap mode where a paragraph can be split over many lines, and an empty line is needed to mark the end of a paragraph (same behavior than markdown).
- **include** : includes (and parse) another HEML file in place of the include command. File to include is specified using the
- **table** : includes or defines in-line tabular data (that is ASCII delimited data such as CSV format). More table examples are provided below.
- **exec** : Exec system command defined with the **cmd** attribute and insert what the command prints to its standard output.
- **script** : run enclosed script. At the time this sentence is written, only lua language is supported.
- **dic** : import a dictionary from a property file.
- **k** : keyword/value substitution from previously imported dictionary.

3.4 Includes

HEML	XML
FileA.heml: <pre> 1 {section %title=include example 2 paragraph 3 {?include %src=FileB.heml} 4 paragraph 5 }</pre> FileB.heml: <pre> 1 included paragraph 2 3 included paragraph</pre>	<pre> 1 <?xml version="1.0"?> 2 <section title="include example"> 3 <p>paragraph</p> 4 <p>included paragraph</p> 5 <p>included paragraph</p> 6 <p>paragraph</p> 7 </section></pre>

3.5 Files Extracts

HEML	XML
FileA.heml: <pre> 1 {section %title=include example 2 paragraph 3 {?extract %src=ParserCallback.java %from= ↵ interface %to=ROW_ELEM} 4 paragraph 5 }</pre> ParserCallback.java: <pre> 1 package net.eduvax.heml; 2 3 public interface ParserCallback { 4 void openElement(String name); 5 void closeElement(); 6 void addAttribute(String name,String ↵ value); 7 void endAttributes(); 8 void addText(String text); 9 void openPara(); 10 void closePara(); 11 void openEnum(); 12 void closeEnum(); 13 void openIndent(); 14 void closeIndent(); 15 void addComment(String comment); 16 void addCDATA(String cData); 17 void openDocument(); 18 void closeDocument(); 19 void stateChanged(Parser.State s); 20 void openTable(int rowStyle,String ↵ rowName,Iterable<String> fieldsName); 21 void addRow(Iterable<String> fieldsValue) ↵ ; 22 void closeTable(); 23 int ROW_ELEM=0; 24 int ROW_ATTR=1; 25 }</pre>	<pre> 1 <?xml version="1.0"?> 2 <section title="include_example"> 3 <p>paragraph</p> 4 <pre> void openElement(String name); 5 void closeElement(); 6 void addAttribute(String name,String ↵ value); 7 void endAttributes(); 8 void addText(String text); 9 void openPara(); 10 void closePara(); 11 void openEnum(); 12 void closeEnum(); 13 void openIndent(); 14 void closeIndent(); 15 void addComment(String comment); 16 void addCDATA(String cData); 17 void openDocument(); 18 void closeDocument(); 19 void stateChanged(Parser.State s); 20 void openTable(int rowStyle,String ↵ rowName,Iterable<String> fieldsName); 21 void addRow(Iterable<String> fieldsValue) ↵ ; 22 void closeTable();</pre> 23 <p>paragraph</p> 24 </section></pre>

3.6 Insert a command output

HEML	XML
FileA.heml:	
<pre> 1 {section %title=include example 2 paragraph 3 {?exec %cmd=ls -al} 4 paragraph 5 }</pre>	<pre> 1 <?xml version="1.0"?> 2 <section title="include_example"> 3 <p>paragraph</p> 4 <pre>total 20 5 drwxr-xr-x 3 sdevaux sdevaux 4096 30 mars ↵ 6 2021 . 7 drwxr-xr-x 9 sdevaux sdevaux 4096 2 janv. 12 ↵ 8 :25 .. 9 -rw-r--r-- 1 sdevaux sdevaux 2384 1 janv. 16 ↵ 10 :06 Makefile 11 -rw-r--r-- 1 sdevaux sdevaux 472 30 mars ↵ 12 2021 module.cfg 13 drwxr-xr-x 2 sdevaux sdevaux 4096 1 janv. 15 ↵ 14 :00 src 15 </pre> 16 <p>paragraph</p> 17 </section></pre>

3.7 Tables

The HEML table feature let concise tabular data writing and also import legacy CSV files. Some attributes are available to configure the parser behavior to handle the table content:

- **encoding** : name of character encoding to use when reading imported table. This parameter must be use in combination with **src** attribute.
- **fields** : name of the fields container elements or attributes. When only one name is provided, all fields have the same name unless **style** is set to **attr** . Default field name is **td** .
- **fieldSep** : list of characters to consider as field separator in the following table definition. Default separator character is "
- **record** : name of the record container elements. Default record name is **tr** .
- **recordSep** : list of characters to consider as record separator in the following table definition. Default separator character is "n" (carriage return).
- **src** : if set, its value is used as path or URL to specify file to import as table content. Default is undefined, then table content is expected.
- **style** : if value is **attr** , fields are expanded as record element's attribute, other wise as sub elements. Default behaviour is to create sub elements.
- **token** : when set to **true** , fields are extracted like StringTokenizer java class does. E.g. Adjacent separator characters are considered as a single one. Default value of this attribute is **false** .
- **trim** : when **true** , leading and ending blank characters (space, tabs, ...) are removed from extracted fields. Default value of this attribute is **true** .

HEML	XML
<pre> 1 {?table %record=tableRowName %fields=field1, ↵ field2,field3 2 un % un1 % un2 3 deux % deux1 % % deux3 4 trois % trois1 % trois2 % trois3 5 } </pre>	<pre> 1 <tableRowName> 2 <field1>un</field1> 3 <field2>un1</field2> 4 <field3>un2</field3> 5 </tableRowName> 6 <tableRowName> 7 <field1>deux</field1> 8 <field2>deux1</field2> 9 <field3/> 10 <f1>deux3</f1> 11 </tableRowName> 12 <tableRowName> 13 <field1>trois</field1> 14 <field2>trois1</field2> 15 <field3>trois2</field3> 16 <f1>trois3</f1> 17 </tableRowName> </pre>
<pre> 1 {?table %record=tableRowName %style=attr % ↵ fields=field1,field2,field3 2 un % un1 % un2 3 deux %% deux2 % deux3 4 trois % trois1 % trois2 % trois3 5 quatre % quatre1 6 } </pre>	<pre> 1 <tableRowName field1="un" field2="un1" field3 ↵ ="un2"/> 2 <tableRowName field1="deux" field2="" field3= ↵ "deux2">deux3</tableRowName> 3 <tableRowName field1="trois" field2="trois1" ↵ field3="trois2">trois3</tableRowName> 4 <tableRowName field1="quatre" field2="quatre1 ↵ "/> </pre>
<pre> 1 {?table %fieldSep=" \t" %token=true 2 a b c d e 3 1 2 3 4 4 } </pre>	<pre> 1 <tr> 2 <td>a</td> 3 <td>b</td> 4 <td>c</td> 5 <td>d</td> 6 <td>e</td> 7 </tr> 8 <tr> 9 <td>1</td> 10 <td>2</td> 11 <td>3</td> 12 <td>4</td> 13 </tr> </pre>

HEML	XML
From one HEML file:	
1 <code>{?table %src=test.csv %fieldSep=;}</code>	1 <code><tr></code>
	2 <code><td>1</td></code>
	3 <code><td>2</td></code>
	4 <code><td>3</td></code>
	5 <code><td>4</td></code>
	6 <code><td>5</td></code>
	7 <code></tr></code>
Included csv file:	8 <code><tr/></code>
1 <code>1;2;3;4;5;</code>	9 <code><tr></code>
2 <code>;;;;;</code>	10 <code><td/></code>
3 <code>;III;;V;</code>	11 <code><td/></code>
4 <code>a;b;c;d;e;f</code>	12 <code><td>III</td></code>
	13 <code><td/></code>
	14 <code><td>V</td></code>
	15 <code></tr></code>
	16 <code><tr></code>
	17 <code><td>a</td></code>
	18 <code><td>b</td></code>
	19 <code><td>c</td></code>
	20 <code><td>d</td></code>
	21 <code><td>e</td></code>
	22 <code><td>f</td></code>
	23 <code></tr></code>

3.8 Lua scripting

Lua scripts can be inserted and the document structure handler is bound to the lua context.

HEML	XML
1 <code>{root</code>	1 <code><root></code>
2 <code>{?script</code>	2 <code><elem>entry 1</entry></code>
3 <code>cpt=1</code>	3 <code><elem>entry 2</entry></code>
4 <code>while cpt<=3 do</code>	4 <code><elem>entry 3</entry></code>
5 <code> handler:openElement("elem")</code>	5 <code></root></code>
6 <code> handler:endAttributes()</code>	
7 <code> handler:addText("entry "..cpt)</code>	
8 <code> handler:closeElement()</code>	
9 <code>end</code>	
10 <code>?}</code>	
11 <code>}</code>	

3.9 Dictionary and keyword substitution

HEML	XML
<pre> 1 {?dic %src=kw.properties} 2 {root 3 {elem 4 {?k one} 5 } 6 {elem 7 {? two} 8 } 9 }</pre> Where kw.properties stores few keywords: <pre> 1 one=first sentence 2 two=second sentence</pre>	<pre> 1 <root> 2 <elem>first sentence</elem> 3 <elem>second sentence</elem> 4 </root></pre>

3.10 Miscellaneous features

- comments: text between {# and #} is ignored by the parser.
- as-is copy: text between {! and !} is not interpreted and forwarded unchanged to the output inside au <pre> element.
- escape character: any character following is forwarded to output as-is. Useful to include some of the HEML control char such as { into one element's text.
- SAX-like callback API: the current java implementation can be used as a library and programmer can define its own HEML handler by implementing the [ParserCallback interface](#)⁴.
- See this documents's [HEML source file](#)⁵ to see a complete use example.
- syntax highlight: markup is simple then syntax highlighting configuration should be easy for common editors. A [syntax highlight vim script](#)⁶ is included in HEML parser source distribution as an example.

4 Similar formats discussion

4.1 Markdown

Markdown is a precursor and may be the most popular lightweight and practical example of a lightweight document format designed to be edited with simple text file editor tools. But markdown is only supporting (simple) document and was not designed to be used to serialize any kind of data like XML is. HEML was inspired by markdown for paragraph and bullet list layout. But for anything else, it is closer to XML itself.

4.2 Wiki markups

Wiki markups are designed to allow easy in line edition from a simple text field. But the purpose is first to support documents writing. Trying to edit any kind of data like with XML is probably possible but leads to some kind of [brainfuck](#)⁷ like naming and coding convention above the standard markups to make any data processable.

See [Wiki markup wikipedia page](#)⁸ to know more about the many wiki markup languages.

4.3 YAML

YAML's basic purpose is very similar to HEML's one: enabling easy manual data edition. But it's primary focus is data, and promote lists to handle data. May be I didn't realized what YAML is before I started coding HEML. I just found YAML

⁴<https://github.com/seduvax/heml/blob/master/heml/src/net/eduvax/heml/ParserCallback.java>

⁵https://github.com/seduvax/heml/blob/master/_doc/src/heml_manual.67aa15e.heml

⁶<https://github.com/seduvax/heml/blob/master/heml/src/res/heml.vim>

⁷<http://en.wikipedia.org/wiki/Brainfuck>

⁸http://en.wikipedia.org/wiki/Wiki_markup

was not what I searched and hope HEML is more convenient particularly for documents writing. At least it is the case for me!

See also [YAML home page](#)⁹ .

4.4 txt2tags

Txt2tags can be view as wiki markup precursor and suffers the same limitation when used to edit data that are not readable documents.

See also [txt2tags home page](#)¹⁰ .

4.5 JSON

JSON features are very close to YAML and like YAML seems more focused to complex data serialization support and not best suited to direct and manual document edition. It is directly related to javascript, even it is usable with any programming language thanks to the many libraries available.

See also [JSON home page](#)¹¹ .

4.6 XML

Of course, you can edit directly [XML](#)¹² , HTML, XHTML... But it's quite boring because of its heavy markup (opening and ending tags, < and > for each, quotes for attributes). HEML is the same but requires fewer markups then less characters to type. Moreover, paragraph layout feature inspired from markdown, wiki markups and txt2tags keeps HEML document accessible to the human eye.

⁹<http://yaml.org>

¹⁰<http://txt2tags.org>

¹¹<http://json.org>

¹²<http://www.xml.org>