LML 0.1 Specification

Sebastian Unger

September 23, 2020

Abstract: This specification defines the first pre-version of a new way of working with multi-language web- and multimedia content: the language markup language (LML). This version describes core elements of the language, that will help to archive the goals of the new markup language: separate web design, content creation and translation, as well as providing a way for content sharing and teamwork.

Contents

1 Introduction

This section is non-normative.

1.1 Background

Until today the WWW core markup languages are HTML and CSS, assisted by script languages like JavaScript and PHP. Both markup languages were designed to separate content and design. The objective was to make modification of one part possible that do not affect the other one.

However, things changed and web design is a large part of today's content creation. Companies, people and organisations with a need for attention require a special design representing their own positions and ideas. When going to mobile devices, more and more complex designs came up, needing special in-text markers and containers to tell which element should be styled.

This led to a linking process between HTML and CSS. Massive use of DIVs and Classes made it impossible, that each file stands for itself. This made working procedures with content more complicated. Formerly easy processes like translating, sharing and archiving now need to consider the design. Also design changes need a propagation to all content files, which is currently handled by large script-work.

LML should change this situation and bring back content files, that are decoupled from design. This should make life easier for all people involved in web content creation and management.

1.2 Scope

This specification does only specify the semantic level of the markup language, as well as interpretation procedures and the connection to existing web technologies. It does not cover implementation details, hardware specifications and software systems.

1.3 Suggested to read

- HTML 5.2. W3C Recommendation, 14 December 2017 - `www.w3.org/TR/html52/`

- CSS Snapshot 2018, W3C Working Group Note, 22 January 2019 - `www.w3.org/TR/CSS/`
- UTF-8, a transformation format of ISO 10646, November 2003 - `tools.ietf.org/html/rfc3629`
- ZIP File Format Specification, July 15 2020 - `pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT`

1.4 How to read this specification

This specification has normative and non-normative parts. Non-normative parts are marked (like this introduction-section) at the beginning of the section with *This section is non-normative.* Non-normative parts do not follow any special rules, as they only add additional information. The following rules are followed in the normative section:

Mandatory instructions are written like this, in normal paragraph style. They should be followed on any implementation.

```
This box show the definition of semantic.
Semantic in black shows which characters are fixed.
Red marks variable text and states its structure.
```

Excursion box

This box shows larger examples or describes design decisions. The content is non-normative, the box content is for transparency about the design process and for further information. The title of this box can differ to fit to the content.

2 Basics

In this section the specification describes the basic requirements and knowledge about LML. This is normative, so the described wording, references and implementations should be followed in any implementation.

2.1 Terminology

- LML model: The LML model describes the whole topic about LML, defined in this document. It includes the data formats, file formats, data structures and the interpretation and rendering process.
- Data format: layout of stored or processed raw data, that is not already present in a data structure. There are two data formats available in the LML model: the LML data format and the ELML data format.
- File format: This defines the structure of stored files. There are three file formats part of the LML model: LML files, ELML files and XLML package files.
- Data structure: An object-oriented representation of LML for processing the data.
- Interpreter: The software and process that interprets LML and ELML files.
- Rendering: The software and process of generating HTML or other content from LML.
- Content data: Data written in the content part of the tags, that describes the content information, as well as the metadata.

2.2 Character encoding

Because of internationality, the character encoding is fixed to UTF-8. There should be no reason to use any other charset for now, if recommended charset is changing, the default charset could be changed in future versions of LML.

Design decision

In HTML there is a special tag for declaring the charset. However, this was because HTML developed over time and needed to support old files, and browsers use this tag or complex algorithms to decide which encoding they will use. Now, in 2020, Unicode is the way to go as it is an international standard supporting nearly every letter existing. A new markup language today needs to make cuts to old technology for simplifying interpretation algorithms.

3 LML data format

The LML data format describes the content that should be saved and displayed. It stores the content itself, references to multi media data, a marker for correctness and a hint to the corresponding design. The data is structured in tags.

3.1 Tags

Tags are written as less-than- and greater-than-signs (<, >) with defined content in between. The correct characters would be U+003C at the beginning and U+003E at the end, like described in the UTF-8 table. The tags have a name stated in this specification and place for content data. Name and all content data fragments are divided with U+002D (-). There are three categories of tags:

The first category, called paired tags, have an opening tag and a corresponding closing tag, which is marked with a slash (/), U+002F, after the < sign. Between the opening and closing tags, there is space for related tags. Only tags defined as related and their related tags are allowed to be placed there.

```
<specified name-content data>
  related tags
</specified name>
```

A second tag category is stand-alone tags. They only could appear without ending tag, so they do not have a slash (/) anywhere except the content data needs to have one. Mostly this are tags that do not need related tags.

```
<specified name-content data>
```

The third and last tag category are flexible tags, that could have related tags, but do not need them. They could be written in a short form, that requires a slash at the end.

```
<specified name-content data/>
```

Doctype

Doctype is a stand-alone-tag. The doctype delivers information about the LML structure. It shows the interpreter, that it's content is LML and what version should be used. This tag looks always the same and needs to be in the first 32 byte of the LML. The interpreter should search within these 32 byte with any algorithm to determine the doctype if needed.

```
<!DOCTYPE lml 0.1>
```

Element tag

The element tag is part of the paired tags, but can be reduced (flexible tag). Element tags start with e- followed by the name of the element.

```
<e-name>
  some language tags
</e>
```

or

```
<e-name/>
```

Here is a list of names, that could be chosen for the element name with a description how they might be interpreted. The description is not binding, the designer could choose to design the elements in any other way. Also, beside of this list, the designer could use own names for any other purpose. However, this is for compatibility reasons to reduce work for content transfer between designs.

- h1, h2, h3, h4:

Variable tag

The variable tag is part of the paired tags category and starts with v- as the specified name, followed by any letter-combination as a name. The related tag for this is the language tag. The name can be freely chosen by the designer, content editors need to use those names (defined in ELML).

```
<v-name>
  some language tags
</v>
```

Variable tag design and usage examples

The variable tag is designed for content meta-information, that could be interpreted by the page designer in any way outside of the content part, for example title, keywords and descriptions. Variable tags are also a way to store strings that are part of more complex structures, for example scripts and forms. The value can be accessed by name from the file, and be placed at any location, or used as a script input. Because of interchangeability the name should describe the purpose of the content.

The possibility to use any name is still a huge downside of LML, because this leads to reworking the names inside the LML file if they are used with different designs. However, this specification does not state a list of binding names. The reason for this is, that in HTML the specified tag names like h1, h2, p and so on were misused for the design for many reasons. So when LML content is moved, the person responsible for the content, needs to fit it to the own design structure.

4 ELML data format

ELML data should follow this rules: The name of each element is at the beginning of the line. After the name there is a colon followed by the output content. This could contain any parameter names between @-symbols. If an @-symbol should be in the output, there should be no text between two @ (@@ -> @). If there is more space needed for the element definition, the content could be stretched to many more lines. by intending them either with tabs or spaces.

5 File formats

6 Data structure

7 Interpretation and rendering