Integrating CMS Features Into The HTML Markup Language

Ismaila Ikani Sule
Zaafirah Web and Media Design, Aberdeen, Scotland, United Kingdom

Abstract - *This paper looks at the concept idea for further developing the HTML markup language by introducing attributes and tags which would enable content editing and management by final web page users while making it easier for designers and developers to build Content Management System features into their codes. One editor page would work with a corresponding web page and CSS file to display editable content to the user. The web designer/developer would code pages as usual setting out editable portions within the HTML to be accessed by the user merely calling up the editor versions of the same pages on his/her browser.*

Keywords - HTML, content management, CMS, CSS, web browser, code.

1 Introduction

Web designing has come a long way over the past decade with more dynamic and easy-to-use web sites being designed and developed for the World Wide Web. The HTML scripting language has provided the key framework upon which a majority of pages for web sites are built [1]. The latest incarnation, HTML5 has been revamped to give web designers even richer tools for coding their web pages.

The basic way to build a web page is simply via a text editor such as Microsoft's Notepad. The relevant HTML markup syntax is typed out and saved with the .htm or .html file extension and the resulting web page can then be viewed in a browser. Modern web pages have HTML working with CSS (Cascading Style Sheets) which help share out the task of laying out page framework (HTML) and handling the display and appearance of page contents like text, pictures, colours and so on (CSS). Thus in the portion of code below, you can have HTML specifying the display of a paragraph of text while the CSS aspect sets styling for the font size and colour of the text:.

```
<p style=”font-size: 14pt; color:
     red;”>Roses are red</p>
```

Coupled with the use of other scripting languages such as JavaScript, PHP and the .NET framework, web designers and developers now can produce a wider variety of web sites for their clients be it for personal or organisational use, e-commerce, social networking or even gaming.

However, while more tools are being made for the web coders, another trend gained popularity in recent times. The creation of editing tools for the web site owners and users themselves who generally have little or no knowledge of the HTML and CSS aspects of their web pages has been booming.

This class of people aren't interested in the codes for their web pages – they are interested in the content of their web pages. So emerged the era of *Content Management Systems* (CMS) allowing them to edit the web pages produced for them by others or even use set templates to produce the pages on their own.

2 The CMS Challenge

Web designers and developers worldwide today have to meet the demands of their clients to come up web pages which can be edited and updated through some form of CMS or the other, ever more frequently. They have had to build pages in such a way that allows content to be accessed, edited and updated on other pages. A variety of methods exists for web coders to give users these CMS control features and a number of them will be examined in this paper.

The question, however, is: why not have CMS features integrated into the HTML language itself so designers and developers can code web pages and corresponding editing pages without the need for scripting separate CMS bundles or using third-party CMS packages?
HTML can work with CSS internally within its codes or externally linked to a CSS file to produce the visual appearance of the web page displayed on a browser. HTML5 comes with additional features for the coder to manipulate graphics and media directly from within HTML.

The concept being proposed here would, thus, enable the coder to build a web page as usual then set editable sections of the HTML codes. Another HTML file (on the same web server) containing only a link to the first corresponding page would use CSS to build up its own content and display those identified portions in editable form to the user on a browser. This way an authorised user can access this *editor* page with content displayed as on the other public page and when the user clicks on any part of the page which is editable, he/she can go on to modify the content [2].

3 Basic CMS Structure

An understanding of the basic features of a typical CMS would help us visualise the kinds of tags and attributes we would be needing to add to the HTML markup language to produce simple controls for both coders and end users. While the CRUD functions [3] - *Copy, Retrieve, Update* and *Delete* - detail the basic editing capabilities available to a CMS user, it would be helpful to study the steps involved in using the CMS as well and then design controls based around a typical user experience [4].

Common steps involved in using a CMS can be summarised as follows:
i) user logs into secure CMS editing page/environment
ii) user selects web pages and content to be edited
iii) user edits or updates selected web page content
iv) user can preview changes made
v) user saves changes to page(s)
vi) user logs out of secure CMS editing page/environment

4 Methods of Providing Users with CMS Capabilities for Their Web Pages

4.1 Use of CMS packages

Web pages are built using CMS software or application packages such as *WordPress*, *Joomla* and *Drupal*, amongst others. These CMS packages come with preset themes and features for web pages which can be customised by web designers and developers and also later edited by the user who is the authorised web administrator.

Pros:
i) Pre-packaged scripts and templates ready for quick use and customization.
ii) No need for the designer or developer to build complex codes for editing features from scratch.

Cons:
i) One may be limited to the set of templates and features available unless you can build up your own codes then integrate as new theme templates.
ii) Usually these packages need to be installed and set up first on a local system and/or server.
iii) There may be less room for creativity than when one has total control over code manipulation for pages.

4.2 Custom scripting the CMS package

Separate CMS packages are built by the developer for editing the web pages via scripting languages such as XML, PHP and ASP.NET in association with databases. Rich text editor scripts, such as *TinyMCE* and *Aloha*, can be incorporated into the finished packages.

Pros:
i) Customizable codes with direct manipulation by the developer.
ii) Re-useable solutions like code libraries can be incorporated into the product.
iii) CMS codes can be scripted directly by the developer without the need for special software or applications being installed.

Cons:
i) Long periods of coding, testing and debugging might be required for developing the CMS package in addition to the web pages.
ii) The database to be used will need to be installed.
iii) The scripted codes, like those for PHP and ASP.NET, cannot be run and used except on a server or within a framework environment.

4.3 Use of web editing software

Web editing software such as Adobe's Dreamweaver can be used to design, develop and edit web pages.

Pros:
i) Professional tools and features are provided for building the website.
ii) A graphical user interface makes building pages easy with or without direct code editing.

Cons:
i) The software package needs to be bought and installed.
ii) Pages need to be re-uploaded each time they are edited.
iii) Not all users are savvy enough to use such software and master techniques involved.

4.4 Use of online CMS editors

Another option would be to build a web page then mark out editable sections of code which can be accessed online using web CMS services like *CushyCMS* (http://www.cushycms.com) and *SurrealCMS* (http://www.surrealcms.com).

Pros:
i) Free versions available for use.
ii) Web pages can be custom coded using a text-editor then editable sections simply marked out.
iii) No installations required.

Cons:
i) The CMS account exists on one server while the user's web pages are hosted on another server which can sometimes lead to communication problems between them.
ii) The CMS web account requires FTP access to the webhosting account in order to access the codes which may sometimes be restricted.
ii) These CMS tools rely on rich-text editors built on JavaScript which may be switched off on some users' web browsers.
iii) Content editing can sometimes alter the web pages' codes in a way unintended by the designer or developer.

5 Proposed Concept for CMS Features to Be Added to HTML and CSS

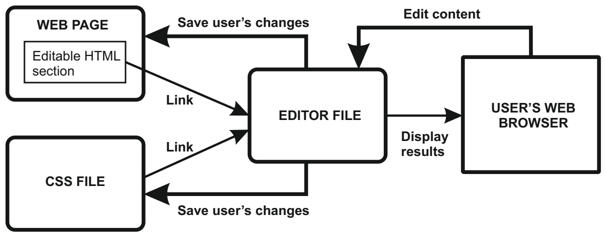

Figure 1: Illustration of how the concept would work

Figure 1 above shows the interaction between web files and the user's browser within the proposed concept. A web page created has sections in its HTML codes marked out as editable. Another web page file acts as the editor file which is linked to both the first web page and a CSS file. Contents from the first web page are displayed on the user's web browser as editable text, images and other media on the page, when the user accesses the editor file. The user can click on these contents, edit and update them.

The editor file would be writing changes directly into the web page and related CSS file.

This system would enable web designers and developers to:

i) focus primarily on building their HTML and CSS files as usual, merely marking out editable elements within the web pages' content

ii) have editor files not requiring complicated coding and linked to the related files

iii) have the web files and editor files stored together in the same location

iv) give users a means of directly editing their web pages' content without the need for databases storing users' inputted values first

v) have a much lighter CMS setup involving just the files created at the time of coding (no complex setup procedures or installations requiring numerous other additional CMS files) [5].

Everything would be taking place basically within the HTML codes. One could conceptualise a web page and its editor file codes to look something as in the following examples.

```
<html>
<head>
<title>Welcome to Webpage 1</title>
<link href="cssfile.css"
rel="stylesheet" type="text/css"/>
</head>
<body>
<p id="main_content_section"
editable="on">Zaafirah and the Ibrahim
kids shouting 'Hello World!'</p>
<img id="Aberdeen_Hybrid"
src="abhybrid.jpg" width="400"
height="100" title="Aberdeen's Hybrid
celebrity" editable="on"/>
</body>
</html>
```

Figure 2: Proposed HTML codes for the web page

The codes in Figure 2 are for a typical web page called 'webpage1.htm'. Content on this page consists of text in the paragraph block marked by the id *main_content_section* and an image marked by the id *Aberdeen_Hybrid*. Both have the proposed *editable* function switched on and so can be edited. We assume that the CSS file, *cssfile.css*, controls the display styles for the page's content (fonts, font colour, image positioning and so on).

Publicly viewing webpage1.htm displays a normal web page and reveals none of the editing features.

```
<html>
<head>
<link href="cssfile.css"
rel="stylesheet" type="text/css"/>
<link href="webpage1.htm"
rel="webpage" type="text/html"/>
</head>
<body>
<edit>
<text id="main_content_section"
update="font-family; color; align;" />
<image id="Aberdeen_Hybrid"
update="size" />
<input type="submit" value="Save" />
</edit>
</body>
</html>
```

Figure 3: Proposed HTML codes for the editor file

The editor file accessing webpage1.htm could be called webpage1_edit.htm and the codes in Figure 3 above show it being linked to both webpage1.htm and *cssfile.css*. While the same paragraph block and image from the first web

page are displayed in the same format in editor file, one can notice the new tags now surrounding them.
The layout and display of content on webpage1_edit.htm would be controlled by the webpage1.htm and *cssfile.css* files leaving only the editing options on the page for the user. Thus, a hypothetical <*edit*> tag is added to the HTML codes surrounding other elements to be edited, namely the text and image from webpage1.htm.
Given that we would not need to repeat the same HTML tags from the original web page file (as this file would already have some control over the display of its contents in the editor file), the content to be edited can be represented by another set of hypothetical tags <*text*> and <*image*> for the text and image respectively. Both and any others to used would be contained within the <*edit*> environment.
In this example, the text content from the paragraph block to be edited would be represented as:

```
    <text id="main_content_section"
 update="font-family; color; align;" />
```

where the text is identified by the id, *main_content_section*, marking the paragraph block in webpage1.htm. An *update* attribute allows the web coder to set the editing options that would be available to the user in updating this block of text content. In this instance, the user can change the font type (that is, font family), the text colour and alignment.
Just as with regular form elements in HTML , these editing options would be displayed in graphical mode for the user. So, say the user were to click on the editable text on the page displayed on his/her browser, drop-down options, buttons or the like would be displayed. The user can then view, click on and implement the options desired for font, colour and alignment of the text.
For the editable image, we have:

```
        <image id="Aberdeen_Hybrid"
              update="size" />
```

where the image bearing the id *Aberdeen_Hybrid* can have it's dimensions modified by the user as set in the *update* attribute. The *size* option would allow this. Other plausible options definable within the image's *update* attribute could include *upload* (so when the user clicks on the image, there is an option to upload a new image), *title* (so the user can change the title attached to the image) or *quality* (so the user can adjust the tone, shade, brightness, contrast and so on).

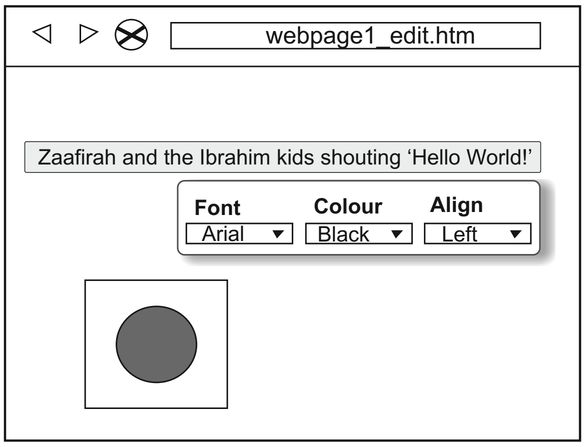

Figure 4: User browser view for editing text in the editor file

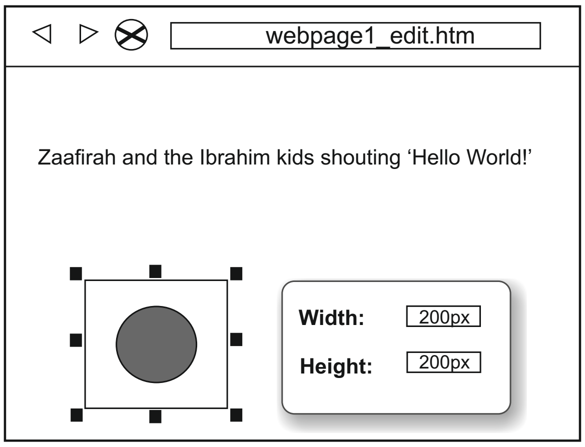

Figure 5: User browser view for editing the image in the editor file

Figures 4 and 5 above illustrate what the editor page could look like when viewed on the user's browser

6 Requirements for the Proposed Concept

i) HTML codes will need to be able to specify editable attributes for sections of code to be identified for editing.
ii) Editable code sections need to be capable of being displayed on web browsers for manipulation by users via HTML coding using a minimal number of files (that is, not building or using an entire CMS package in addition).
iii) Edited pages need be capable of being saved.
iv) The editing process needs to focus on just the editable portions of the page alone and not have to keep re-writing the entire page each time the user makes changes. The system needs to offer similar efficiency levels or more as

with one working with a database [6].

v) Access to web pages for editing needs to be secure.

vi) Adequate web browser support and compatibility will be needed on the different browser platforms available to users.

7 'localStorage' and 'contenteditable' Features in HTML5

Two exciting developments in HTML5 are those of the *localStorage* object [7] and *contenteditable* attribute [8] which allow HTML to store and recall data as well as define aspects of code which are identified as editable. No database is used nor cookies.

The HTML5 code for a sample editable web page below uses Google's jQuery API (online access needed) to highlight how *contenteditable* and *localStorage* work [9]:

```
<!DOCTYPE HTML>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/
libs/jquery/1.4.2/jquery.min.js"></scr
ipt>
</head>
<body>
<p id="editsection"
contenteditable="true">Hi, edit this
text!!!</p>
<script type="text/javascript">
$(function() {
var editsection =
document.getElementById('editsection')
;
$(editsection).blur(function() {
localStorage.setItem('user_edit',
this.innerHTML);
});
if (localStorage.getItem('user_edit'))
{
  editsection.innerHTML =
localStorage.getItem('user_edit');
}
});
</script>
</body>
</html>
```

Figure 6: HTML5 example with *contenteditable* and *localStorage*

Saving this code as an HTML file from a text-editor produces an editable web page where the text *“Hi, edit this text!!!”* has been identified using the id *editcontent*. *Contenteditable* is set to “true” so on the resulting page the user can edit the text by clicking on it.

The modified text is then passed as a variable also called *editcontent* unto a JavaScript function making use of *localStorage* to store the data locally on the user’s computer while updating the web page on the browser.

With the changes now stored locally, the user can refresh the web page and still get to see the modified text.

The modified text remains intact even when the user returns to view the page at a later date after turning off his/her computer.

There are currently limitations to using *contenteditable* and *localStorage*, however.

i) The HTML5 example in figure 6 relies on JavaScript which is a client-side script – it executes functions on the user's browser on a local system and does not apply changes to the web page itself on the web server. For the proposed CMS model to work, a means of getting content edited via HTML and being saved on the web pages is needed. This may require new functions and attributes being developed for HTML allowing secure server-side storage of data. Using XML, PHP or ASP.NET scripted pages amongst others on the web servers would work but the goal is integrate CMS features in HTML itself.

ii) The user has to be using an HTML5 compatible web browser in order for these functions to work. Fortunately a lot of the modern browsers now, including Explorer 9, support HTML5.

iii) On its on so far, the user cannot use *contenteditable* to carry out formatting tasks like setting fonts, font size, colour and so on. Such capabilities could be added using JavaScript but that would mean more coding by the designer or developer.

iv) Using *contenteditable* along with *localStorage* as demonstrated would also mean if the user changed his/her computer, the changes made to the web page on the previous computer would no longer be reflected. The same is the case if the web page file is transferred on to a different computer. The actual HTML codes remain unaltered and changes are not saved within them.

Nevertheless, even with these highlighted limitations of features available in HTML5, the possibilities of an improved method for an HTML/CSS-based CMS are highlighted.

8 Conclusion

Having a simplified HTML/CSS-based CMS coding structure works to the benefit of both web designers and developers and well as the users of their products. Recent achievements with HTML5 show such a structure is attainable with further research and development on the markup language.

9 References

[1] Roger Lipera. “Introduction to HTML/XHTML, Handout Companion to the Interactive Media Center’s Online Tutorial”. University at Albany, State University of New York, 2008.

[2] David R. Karger, Scott Ostler, and Ryan Lee. “The Web Page as a WYSIWYG End-User Customizable Database-Backed Information Management Application”. Proceedings of UIST’09, 257 -260, October , 2009.

[3] Brian P. Hogan. “HTML5 and CSS3, Develop with Tomorrow’s Standards Today”. Pragmatic Programmers, LLC, 2010.

[4] Báscones, P. and Carreras, C. “Managing Memory Institutions Portals: from HTML to CMS and Towards Applications in XML for Multi-platforms”. Int. J. Digital Culture and Electronic Tourism, Vol. 1, No. 1, 18–36, 2008.

[5] David Thomas Dudek and Heidi A. Wieczorek. “A Simple Web Content Management Tool as the Solution to a Web Site Redesign”. Proceedings of SIGUCCS ’03, 179 - 181, September, 2003.

[6] Edward Benson, Adam Marcus, David Karger and Samuel Madden. “Sync Kit: A Persistent Client-Side Database Caching Toolkit for Data Intensive Websites”. Proceedings of WWW 2010, 121 - 130, April, 2010.

[7] Web3Schools. “HTML5 Web Storage”. Available at http://www.w3schools.com/html5/html5_webstorage.asp

[8] Web3Schools. “HTML5 Global contenteditable Attribute”. Available at http://www.w3schools.com/html5/att_global_contentedita ble.asp

[9] Jeffrey Way. “Quick Tip: Learning about HTML5 Local Storage”. Net Tuts Plus, 2010. Code modified in this paper. Available at http://www.youtube.com/watch?v=h0uZIljjElo