Advanced Forms and Menus in Web Development
Extending HTML 4.01 by Framework

Fredrik Reveny
Abstract

Advanced Forms and Menus in Web Development

Fredrik Reveny

HTML 4.01 has been around since 1999. In this report we specify and describe an implementation of a framework for web development which encapsulates overhead associated with setting up forms and menus in web development for modern web sites. The framework achieves a high level of abstraction without compromising versatility. The result is a product which increases productivity and decreases time-to-market.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Side Note on Nested Forms</td>
<td>13</td>
</tr>
<tr>
<td>5.2 Framework Benefits</td>
<td>13</td>
</tr>
<tr>
<td>5.3 Framework Limitations</td>
<td>14</td>
</tr>
<tr>
<td>5.4 Future Work</td>
<td>14</td>
</tr>
<tr>
<td>6 Bibliography</td>
<td>15</td>
</tr>
<tr>
<td>Appendices</td>
<td>17</td>
</tr>
<tr>
<td>A Recursive Destringify Algorithm</td>
<td>17</td>
</tr>
</tbody>
</table>
1 Introduction

“Way 26” is a company focused on selling back-office web interfaces. They use a template system to create customer-specific solutions. This template has had two major versions, internally named 1.0 and 1.5. The company was at the time of this project planning to release 1.5 and start working on 2.0 in the near future. They invited the author of this paper to create a framework that would hopefully ease some of their developer’s ailments in preparation for version 2.0.

1.1 Background

Web sites are built using the client-server model. A web browser (client) is serviced by a web server only on request, which usually invokes a complete reload and re-rendering of the page. Response times are crucial within this process, as reloading the whole page might be frustrating, and user interfaces that manage response times of less than 1 second allow the user to maintain a flow of thought (Nielsen, 2014). Such responsive interfaces would therefore seem to be preferable to the user. The goal of any good web development framework must be to help the developer reflect the user expectations in their user interfaces. It is with this intention this responsive framework is presented.

1.2 Problem Description

At the time of implementation HTML 4.01 was the latest available version. HTML 5 solves some of the problems listed in this paper, but was not an option at the time of implementation.

The most basic web interface setup consists of a web server that responds with pure HTML when a client connects. HTML 4.01 on its own has a large number of limitations by today’s end-user expectations. This chapter lists those that this framework has attempted to address.

1.2.1 Uploading Files

The element for uploading files with HTML 4.01 is limited to 1 file per input element (Refsnes Data, 2014) which means the number of files to be uploaded would have to be known before the page is loaded, or a separate query with the user would have to be performed.

Limiting the maximum size of a file cannot be done with HTML 4.01, which means the entire file has to be uploaded so that the server can verify file size.

1.2.2 Menus

In back-office interfaces it is fairly common to load menus, or specific parts of menus, based on the credentials of the current user. HTML 4.01 has no native method of identifying the current user, and as such implementing a menu system that loads the appropriate menu per user would be difficult in pure HTML.

Some menus may need to be editable by the end-user. Such menus would have to be stored somewhere in a small format. The HTML method of creating a menu separate from the main area of a web page with frames has the additional issue of each frame being treated as completely separate web pages. This causes problems for search engines, printing, bookmarking etc. (isolani, 2012)
1.2.3 Forms
Back-office interfaces need to be able to process a lot of information, and since Way 26 offers customized ("white-label") systems to customers, new forms are written frequently. The form element in HTML 4.01 is without certain features expected in modern web interfaces. When a typical form is submitted, the entire web page is reloaded. Any user data not submitted is lost, and the form is cleared. Also if the user goes to another page when a form contains data, it will by default be discarded. An accidental mouse click could lead to work being lost. This is not always the desired functionality, which has to be overcome in order to make an acceptable interface. Indeed, today many users expect the information to be retained in some way or the other, as better and better techniques exist for this purpose.

The original intention of HTML 4.01 as far as data validation goes is for the user to enter data, click submit, and let that data be handled in the new page or frame. Form validation would be quicker and more useful to the user if it was performed client-side. Each field’s element could then also include the most basic information about what defines that field’s data as valid, as opposed to maintaining a validation scheme server-side.

1.3 Limitations
These limitations were set to help fit the project within the planned time-span. The following features have had to be explicitly omitted to fit the project within the given constraints:

- Users and logins
- Security
- Database setup and handling
- Aesthetics

In future work these aspects would have to be addressed as well, but have deliberately been left out of this project.

1.4 Disposition
The paper begins by briefly describing the underlying technologies and their place in the project in the Technical Background.

The Specification describes feature requirements. I.e. what specific problems the project shall solve, or improve upon.

The Implementation describes the final product through examples, and gives an in-depth guide for users (developers) to utilize it. The chapter also contains a list of software prerequisites.

The Concluding Discussion discusses benefits, limitations of the project, features discarded in the specification-phase as well as potential future work to improve the product further.

1.5 Stakeholders
Features of the framework were meant to be integrated with the pre-existing template system and used during the development process of new white-label systems for Way 26 customers. As such, assuming no miracles, it will most likely only be utilized by Way 26 developers.
The framework is aimed at reducing certain repetitive tasks. This should increase the well-being of the developers which will in turn hopefully increase productivity, thus addressing another direct stakeholder for the framework, i.e., the Way 26 investors, as the potential increase in developer productivity might help reduce development time- and cost.
2 Technical Background
The description in this paper is aimed primarily at web developers. The reader is expected to understand the basics of programming, web development and database management. He or she should also be acquainted with the terminology concerning back-end computing.

The product associated with this paper was aimed to be the basis for the sequel of a pre-existing product that utilizes several well-established technologies. It was decided that by using the same technologies, engineers developing the product would be able to give the best support, and that integrating it with the pre-existing system would be easier.

This chapter briefly describes the benefits of the technologies involved, why they were selected or not, and reference to further information on them.

2.1 HTML 4.01
The customer and end-users will already be using HTML-oriented web browsers. It is the de facto standard for internet web pages, and hence comes as the natural choice for a web page representation language.

As mentioned in section 1.2, it is obvious that HTML version 5 holds many improvements, but since it was released after this project was finished, it was not an option. Any reference to HTML in this paper is therefore, unless explicitly stated, to HTML version 4.01. For a complete description of HTML, see (W3C, 1999).

2.2 Cascading Style Sheets (CSS)
Cascading Style Sheets (CSS) is mainly used to format the rendering of HTML-based web pages. Because aesthetics is omitted from the project (cf. 1.3 Limitations), CSS is not used in this project.

2.3 CodeIgniter
CodeIgniter is a PHP-based framework for generating dynamic web page content. CodeIgniter aims to let developers work with the Model-View-Controller (MVC) development pattern by separating data management routines (Model), page layout (View) and the flow of information (Controller) into separate file structures.

CodeIgniter was used in this project primarily because it is what the developers at Way 26 used, and they wanted to work with something as familiar as possible. The familiarity would help ease integration with their pre-existing system.

For a complete description of CodeIgniter, see (EllisLab, 2014).

2.4 PHP
PHP is a scripting language that runs on web servers. When a user connects to a website, the web server runs PHP scripts to generate HTML based on user data.

CodeIgniter is built on PHP and this is the primary reason for using PHP in the project, despite the somewhat bad reputation it has among some programmer groups, especially when it comes to security and program structuring. The use of a good coding framework such as CodeIgniter, however, supports the use of PHP, as well as other languages.
2.5 JavaScript
JavaScript is an interpreted programming language, which runs in the user’s web browser to perform actions on the client side after HTML has been retrieved from the web server. JavaScript applications run locally as part of the web browser but are able to perform further requests to the web server.

This makes it suitable for creating a more responsive user interface experience. It is also the de facto standard of browser-side scripting languages.

For a complete description of JavaScript, see (JavaScripter.net, 2011).

2.6 Asynchronous JavaScript (AJAX)
AJAX is a method for the web browser to send/receive data through transactions between the web page JavaScript and the web server, after the page has been loaded.

AJAX is basically a method that allows for only reloading parts of a page, rather than the whole page with all its static information. This is useful for instance when submitting a form and only a short response like “OK” or “Error” is needed from the server, as opposed to downloading, reloading and re-rendering the entire page.

For a complete description of AJAX, see (Garrett, 2005).

2.7 JavaScript Object Notation (JSON)
JSON is a minimalistic data container format that supports storing recursive data structures in a minimalistic format suitable for limited bandwidth transfer. JSON was selected for its recursive nature, which is excellent for storing menus and submenus, as well as the minimalistic format which will decrease bandwidth load compared to similar formats like XML.

For a complete description of JSON, see (ECMA, 2013).

2.8 jQuery
jQuery is a JavaScript library that uses its own tricks to essentially change the way JavaScript is used. Besides containing libraries for simplifying common JavaScript actions like AJAX transactions, it alters the way DOM element manipulation is performed by adding the “$” function.

For example, to set the contents of a div element with ID “example” to “Hello!” in regular JavaScript:

document.getElementById("content").innerHTML = "Hello!";

To do the same thing in jQuery:

$('#example').html("Hello!");

It was the opinion of the author and developers at Way 26 that JavaScript with jQuery is much more intuitive and readable than plain JavaScript, and this is the reason for using jQuery.

For a complete description of jQuery, see (The jQuery Foundation, 2014).
2.9 MySQL

MySQL is an open-source relational DBMS, which has become more or less another de facto standard when it comes to open source database development. As the name suggests, it uses SQL for data input/output. MySQL was originally selected for being free and it is well known and tested software. It was selected in this project for those reasons, and for ease of integration.

For a complete description of MySQL, see (Oracle Corporation, 2014).
3 Specification
This specification of the programming work was defined in collaboration with developers at Way 26 through 2 meetings in December of 2011. At the first meeting we set an approximate level of ambition for the project, a time scope, and discussed my abilities as a developer. For the second meeting the developers settled on a rough, informal draft of this specification. The framework specified in this chapter contains a subset of features for a framework the team at Way 26 had in mind for a long time.

The purpose of the framework is to decrease the amount of overhead for creating non-standard extended HTML components. These are components that require extra JavaScript and server-side scripting to function but which also yield the kind of complex features that users have come to expect from some common components of modern web sites like file uploaders that support more than one file at a time, forms that don’t reload the entire page etc.

The goal of this project is to create a framework of both client- and server-side tools that let developers write their forms and menus with such advanced features faster than what basic HTML allows. This is accomplished by requiring as little data as possible when adding new elements.

For example, a form is written as (in its simplest submittable form):

```html
<form><input type="submit" value="OK" /></form>
```

In a typical HTML web page, this form would reload the entire page when the “OK”-button is pressed, but using jQuery, the framework will transform the form behind the scenes when the page is loaded, to a more modern variant where pressing “OK” sends user-specified data and, only if desired, changes the active frame (leaving the state of other frames unchanged).

The framework is to support the following features:

- Multiple File Uploading,
- Storable menus,
- Isolated form submission,
- Forms with page change confirmation, and
- Forms with validation.

These will be described more thoroughly in the following sections.

3.1 Multiple File Uploading
HTML 4.01 requires developers to statically write one element per file to be uploaded. While this could be solved by server-side scripting, we want to reduce client-server communications as much as possible, to improve the user’s experience. The framework shall transparently extend HTML with an element that lets the user dynamically alter the number of files to be uploaded.

3.2 Storable Menus
A conventional menu can be stored in a tree structure, where leaves represent buttons, and non-tree nodes represent sub-menu categories. This is called the composite design pattern, which is how menus shall be stored.
The JSON file format will be used to store menus as strings in server-side databases.

Menus shall be loaded from the database through AJAX when the page is loaded.

Menus shall have a well-defined target DOM-object to alter when menu buttons are clicked, to avoid reloading the entire page.

### 3.3 Advanced Forms

“Advanced forms” refers to forms with extended capabilities that users have become acquainted with in most web sites, such as isolated submission, page change confirmation and validation. These are the three features described below.

Forms shall never reload the entire page, or frame, when the submit button is pressed. They shall only display the server response in a small iframe DOM element next to the submit button. Form submission shall be handled through AJAX.

When a user has entered data in a form and attempts to leave the current page, he shall be informed that data has been entered and asked to confirm the page change.

Developers shall be able to enter a JavaScript expression together with input text fields in HTML for validating user input on the client’s side. This will be done by integrating the jQuery Validation library.

### 3.4 General Considerations

Most problems specified here have been solved before. A large part of the goal with this framework is instead to bring these solutions together into a coherent structure. The implementation should re-use and integrate existing libraries and functionalities where possible.

The business of Way 26 is to provide applications containing a lot of forms, due to their rather uncommon business of delivering back office systems, using a single framework as a base. It should be considered when providing a standalone framework that developers at Way 26 will need to have a common, well-known, starting point, as well as an easy way of setting up their development environment. The implementation will need to consider the complexity of technologies used, and how to both set up a developer’s environment and how to deploy the system for the customer. Done properly, this should benefit all stakeholders by saving time and effort.
4 Implementation

The general concept of this framework is to dynamically replace specially tagged standard HTML elements client-side, so as to both provide a familiar syntax for developers, but also allow for further functionality and responsiveness. The developer places an element and tags it with a special attribute such as an ID or a class name. When the page is loaded in the web browser, the tagged element is replaced through JavaScript by an upgraded version of the HTML. These upgraded elements will be referred to as framework elements.

The framework provides everything in a template to minimize the amount of work required for a developer to have a web site up and running. Provided is a header file meant to cover all information contained in the mandatory <head> tag, an example menu/content file, and a footer file to close all tags left open by the header file.

Figure 1: Overview of the provided template project layout. Everything between the header and footer is part of the <body> element.

**Header:** Contains the head element which includes all the necessary JavaScript files and the opening <body> tag. Developers are free to alter the default header for their own purposes if extra style sheets or script files need to be included in the project.

**Application:** The application is specific for the project and may contain framework- or conventional HTML elements in any fashion the developer desires. The most typical layout is included in the template. There is a menu at the top, and a content area that is controlled by the menu.

**Menu:** A div element with id attribute “topMenu” will automatically be replaced by a developer-defined menu which is loaded from a database.

**Main Area:** A div element with id attribute “main”. Pressing links in the menu replaces the contents of this area using AJAX.

**Footer:** Closes all tags left open by the header.

The developer decides on what should be placed in the application area. Typically this will consist of a main menu framework element and a main content area framework element whose content can be altered by the menu element. This is what the template provides.

This chapter specifies all the requested features of the framework and how they are supposed to be used by a developer.
4.1 Multiple File Uploading
This is the feature described in 3.1 Multiple File Uploading. The new file uploader consists of a set of buttons for uploading multiple files using input elements (see figure 2). The user can add new file uploaders by pressing the “+”-button and remove specific uploaders by pressing the corresponding “-”-button.

![File Uploader](image)

Figure 2: A file uploader supporting multiple file uploads. The arrow indicates what happens when the ‘+’-button is pressed.

The framework looks for all div elements with the class attribute set to “fileUploader” and replaces them with input elements for uploading files. If these elements are placed in conjunction with a div element with the class attribute “addFile”, the element is extended to support adding more files for uploading, if desired.

There is one problem remaining, since the elements used are not able to detect if a file is above a certain size limit. It was simply not possible during the time of implementation, due to security concerns, but has since been made possible by the World Wide Consortium’s File API. (W3C, 2013) An updated version should incorporate this feature in the framework.

4.2 Storable Menus
This is the feature described in 3.2 Storable Menus. The framework JavaScript looks for all div elements with the ID attribute set to “topMenu”, queries the database for the menu and replaces the element with the menu defined in the database. The menus are now stored in the server database as JSON strings using JSON.stringify. (JSON, 2014)

When menus are loaded from the database and the JSON structure has been parsed using JSON.parse (JSON, 2014), an algorithm which recursively traverses the JSON tree is used to convert JSON objects to HTML. It finds objects on the form {"text": "Text", "link": "file.php"} and replaces them with HTML button elements labelled by the value for the “text”-key that link to the value for the “link”-key. Arrays of these objects, or other arrays, become submenus. Only JSON arrays with keys that start with “#” are converted to submenus, with the text following “#” becoming the name of the submenu. A pseudo code version for this algorithm can be found in Appendix A: Recursive Destringify Algorithm.

When the web site is initially loaded the framework JavaScript looks for all div elements with the ID attribute set to “main” and replaces them with what the developer sets as the default page. When menu buttons are clicked, this element is updated by the menu system through AJAX requests.
4.3 Advanced Forms

These are the features described in 3.3 Advanced Forms.

4.3.1 Isolated Submission

HTML 4.01 lets developers specify a target frame to update when a form is submitted. By creating an invisible frame and altering the form to target that frame on submission, a form can be submitted without reloading any of the data relevant to the user. This was solved by adding a small hidden frame next to the form and setting it as the target frame for server responses. This is automatically configured for all forms in the web site through client-side DOM element manipulation, instead of forcing the developer to set this up for each form.

Figure 4: When a form is submitted, the server response HTML is always output to a frame at the end of the form.

4.3.2 Page Change Confirmation

When the user has entered data into any input field of a form and then attempts to use the menu to change page, or go to a different web site entirely he is issued with a query stating that there is non-submitted data that will be lost, and asking for confirmation of a page change.

Figure 5: When the user tries to change page after entering data into a form, this window is displayed. This window is browser-specific in appearance and this example is from Google Chrome.

4.3.3 Validation

The framework enables developers to specify what constitutes valid data for each input element in the form of a Boolean JavaScript expression. This is done by calling the validate function on input elements, which takes a JavaScript expression and an error message to display if the expression evaluates to false.

The following is an example how to create an input element that only accepts values above 100:
A number greater than 100: <input id="ValidNumber" name="v1" type="text" />

If the user fails to enter a value that evaluates to be larger than 100, the field is marked red and the message “Should be a number greater than 100” is displayed next to the element.

The JavaScript library “jQuery Live Form Validation” was used. (GeekTantra, 2009) It is included in the provided template header-file and therefore available in all web pages created with the framework.

Each field is validated client-side when the selected field is changed or the form is submitted. If data is invalid according to the validation statement, the POST or GET is not sent and the specified error is presented to the user.

![Figure 6: The framework provides a user-friendly way to validate data client-side. The validation statement for the failing input element here is “return VAL == Math.sqrt(65536);”](image)

### 4.4 Software Specification

The framework was implemented with version 2.2E of WAMP. WAMP is a complete web development environment for Windows, aimed at developers that use Apache, PHP and MySQL, which are precisely the technologies used by developers at Way 26. Similar frameworks are available for all the main platforms, such as Linux (LAMP) or Mac OS X (MAMP). The functionality is the same on all platforms.

The WAMP 2.2E installer installs the following software:

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>5.5.24</td>
</tr>
</tbody>
</table>
This, together with the files attached to this paper is all that is required to start using the framework.

5 Concluding discussion of the resulting implementation

Compared to the capabilities of a purely HTML 4.01-based website, the framework provides many extra features and additions. However, as frameworks like this have already been developed by larger groups with both more time and experience, it seems to pale in comparison.

Examples of such frameworks are:

- Drupal (Buytaert, 2014)
- WordPress (WordPress, 2014)
- Joomla (Open Source Matters, Inc., 2014)

These include all the features specified for the framework described in this paper, and plenty more. They are on the other hand in some aspects probably a bit of overkill, for the limited purpose of Way 26.

Code to store and load menus to databases was deployed and maintained by Way 26. On-site developers found it to be suitable to their plans, while the rest of the code was to be used for reference when implementing their own framework using the same concepts described in the specification of this paper.

All was not for naught, however. The experience from creating this framework provides a much better insight into similar systems, and has also given me a better grasp of the area of CMS systems as information management tools.

5.1 Side Note on Nested Forms

One feature discussed early in the project was the ability to nest form tags in other form tags, e.g. “<form><form></form></form>”. This was a proposed solution for the Isolated Submission problem, the selected solution of which is specified in 3.3 Advanced Forms.

The idea was that this would be useful for large forms where other smaller forms can either be submitted individually, or all at once, as a whole. The idea was later discarded, as this is explicitly forbidden in XHTML 1.0. (W3C, 2002) An interesting point is that it is nowhere explained why it should be forbidden but merely accepted as some kind of general truth. This is something I would like to find a good explanation for in the W3C documents.

5.2 Framework Benefits

The developed framework has the following benefits over similar frameworks:
• JSON is a very minimalistic format, reducing server-side capacity requirements
• Menus can be written in a readable format in JSON and will be compacted as much as possible
• Easily deployable. HTML is written in the same old way as always, except “tagged” elements get extra features.
• Form validation statements are easy to use

5.3 Framework Limitations
Some of the major drawbacks in the developed framework compared to other web development frameworks include:

• Framework features are hard-coded instead of modular using a plugin-based architecture, causing some bandwidth waste as the code for these features are transferred to the client even if they are not used
• Aesthetics have not been considered
• File upload progress not displayed (some browsers do this anyway)
• If a file is too big it is not discovered until the file has finished uploading
• Required tag names for automatically replaced elements were not chosen with care and this forces developers to avoid naming their elements the same

5.4 Future Work
It is the opinion of the author that this framework should not be further developed, at least not under HTML version 4.01. There are also several well established free CMS frameworks available already, but under the assumption that one was to further develop this framework, the following points should be addressed:

• Choose static tag names with care in a consistent manner. “topMenu” and “main” are quite likely to already be used as identifiers in the same JavaScript context
• A file uploader that lets users select multiple files in their native operating system environment. I.e. don’t force users to select one file at a time. This is possible by using the “multiple” attribute available in input elements in HTML 5
• A file uploader that warns the user that the file they’re attempting to upload is too big before they attempt to
• Consider handling aesthetics within the Framework context, so as to provide a complete web development package with no further need for external modules
• Utilize form validation features of HTML 5
6 Bibliography


Refsnes Data, 2014. HTML input multiple Attribute. [Online]
Available at: http://www.w3schools.com/tags/att_input_multiple.asp

The jQuery Foundation, 2014. jQuery Foundation. [Online]
Available at: https://jquery.org/
[Accessed 29 January 2014].

The PHP Group, 2014. PHP: HyperText Preprocessor. [Online]
Available at: http://www.php.net/
[Accessed 29 January 2014].

Available at: http://www.w3.org/TR/html4/
[Accessed 29 January 2014].

Available at: http://www.w3.org/TR/xhtml1/#prohibitions
[Accessed 31 March 2014].

W3C, 2013. W3C File API. [Online]
Available at: http://www.w3.org/TR/FileAPI/
[Accessed 10 March 2014].

Available at: http://wordpress.org/
[Accessed 1 September 2014].
Appendices

A. Recursive Destringify Algorithm

This algorithm outputs a string of HTML. It is defined as:

\[
\text{\texttt{JSON2Menu}}(O) \\
C \leftarrow \text{empty string} \\
\text{for each string } S \in O \\
\text{if } O[S] \text{ is a dictionary} \\
\text{if } S \text{ starts with } \# \\
\quad C \leftarrow C + S + [ + \text{\texttt{JSON2Menu}}(O[S]) + ] \\
\text{else} \\
\quad C \leftarrow C + \text{\texttt{JSON2Menu}}(O[S]) \\
\text{else if } S \text{ is a string} \\
\quad C \leftarrow C + <\text{button} /> \\
\text{return } C
\]

See the function “\texttt{jsonToHtmlMenu}” in xMenu.js for the JavaScript version.

Submenus are constructed in the menu as buttons nested in ‘[‘ and ’]’ tags. This can easily be switched for whatever is appropriate for the framework user’s purposes, e.g. <div> and </div> to be used with CSS formatting. Keys to arrays starting with ’#’ signify submenus.

Example JSON menu:

\[
\text{"menu" : [} \\
\quad \text{{"#Menu System" : [} \\
\quad \quad \{"text": "Saving Menus to DB", "link": "test/TestView/A"\} \\
\quad \]}, \\
\quad \text{{"# Features" : [} \\
\quad \quad \{"text": "Uploading Files", "link": "test/TestView/B"\}, \{"text": "Form Validation", "link": "test/TestView/C"\}, \{"#Submenu": [ \{"text": "Error 404", "link": "test/asdf"\} \} \} \\
\quad\]}, \\
\quad \text{{"# Everything" : [} \\
\quad \quad \{"text": "Combined", "link": "test/TestView/D"\} \\
\quad\]} \}
\]

This represents the following menu:

Menu System[ Saving Menus to DB] Features[ Uploading Files Form Validation Submenu[ Error 404 ]] Everything[ Combined ]