

A MODEL FOR THE AUTOMATION OF HTML FORM CREATION AND VALIDATION

Dragos-Paul Pop¹
Adam Altar²

Abstract

Forms are an essential part of web applications, but handling large forms proves to be very time consuming. This article proposes a model for automating HTML form creation and validation and shows how it could be used to greatly speed up web application development.

Keywords: html, form, web, automation, validation, class, model.

Introduction

One of the most common tasks in web application development is form processing. Along with data presentation (tabular data most of the times), form building and processing takes the most amount of time and work. HTML form elements look, feel and act different. This is because the HTML standard had a rough evolution. Elements don't really follow a clear pattern. They don't have the same structure, attributes, behavior and don't send data to the server in the same way. Furthermore, certain elements became deprecated over time and others changed ever so slightly. Designing an automated system that builds forms and validates them would help save developer time and effort and also ensure that the markup follows standards.

Forms are important because they allow users to enter data. But the HTML standards are quite counterintuitive. For example, one can define a text box by using the *input* element with the value *text* set for the *type* attribute. That seems normal, but when trying to create a text box that spans multiple rows, the *textarea* element needs to be used. The differences don't stop here. While the *input* element doesn't have a closing tag, the *textarea* has one.

Html Standards And Form Elements

Below, there is a list of classical form elements (prior to HTML 5), their status and the time they were introduced in the HTML standards, as listed by Wikipedia:

Element	Description
form	Creates a form. The form element specifies and operates the overall action of a form area, using the required action attribute. Standardized in HTML 2.0; still current.
button	A generic form button which can contain a range of other elements

¹ Ph.D. Student at the Academy of Economic Studies and Teaching Assistant at the Romanian-American University, Bucharest, Romania; email: dragos_paul_pop@yahoo.com

² Ph.D. Student at the Academy of Economic Studies and Teaching Assistant at the Romanian-American University, Bucharest, Romania; email: adamaltar@gmail.com

	to create complex buttons. Standardized in HTML 4.0; still current.
fieldset	A container for adding structure to forms. For example, a series of related controls can be grouped within a fieldset, which can then have a legend added in order to identify their function. Standardized in HTML 4.0; still current.
isindex	isindex could either appear in the document head or in the body, but only once in a document. Isindex operated as a primitive HTML search form; but was de-facto obsoleted by more advanced HTML forms introduced in the early to mid-1990s. Represents a set of hyperlinks composed of a base URI, an ampersand and percent-encoded keywords separated by plus signs. ISINDEX existed in HTML Tags; standardized in HTML 2.0; deprecated in HTML 4.0 Transitional; invalid in HTML 4.0 Strict.
label	Creates a label for a form input (e.g. radio button). Clicking on the label fires a click on the matching input. Standardized in HTML 4.0; still current.
legend	A legend (caption) for a fieldset. Standardized in HTML 4.0; still current.
option	Creates an item in a select list. Standardized in HTML 2.0; still current.
optgroup	Identifies a group of options in a select list. Standardized in HTML 4.0; still current.
select	Creates a selection list, from which the user can select a single option. May be rendered as a dropdown list. Standardized in HTML 2.0; still current.
textarea	A multiple-line text area, the size of which is specified by cols (where a col is a one-character width of text) and rows attributes. The content of this element is restricted to plain text, which appears in the text area as default text when the page is loaded. Standardized in HTML 2.0; still current.
input	input elements allow a variety of standard form controls to be implemented. Standardized in HTML 2.0; still current.

The different input types are listed below:

type="checkbox"	A checkbox. Can be checked or unchecked.
type="radio"	A radio button. If multiple radio buttons are given the same name, the user will only be able to select one of them from this group.
type="button"	A general-purpose button. The element <button> is preferred if possible (i.e. if the client supports it) as it provides richer possibilities.
type="submit"	A submit button.
type="image"	An image button. The image URL may be specified with the src attribute.
type="reset"	A reset button for resetting the form to default values.

type="text"	A one-line text input field. The size attribute specifies the default width of the input in character-widths. maxlength sets the maximum number of characters the user can enter (which may be greater than size).
type="password"	A variation of text. The difference is that text typed in this field is masked - characters are displayed as an asterisk, a dot or another replacement. It should be noted, however, that the password is still submitted to the server as clear text, so an underlying secure transport layer like HTTPS is needed if confidentiality is a concern.
type="file"	A file select field (for uploading files to a server).
type="hidden"	hidden inputs are not visible in the rendered page, but allow a designer to maintain a copy of data that needs to be submitted to the server as part of the form. This may, for example, be data that this web user entered or selected on a previous form that needs to be processed in conjunction with the current form.








HTML 5 introduces some new elements and further improves existing ones. Still, some features are unsupported in most browsers, but this is because HTML5 is a work in progress. Below there are some tables with the improvements that HTML5 brings to forms, as described by the World Wide Web Consortium and the www.html5rocks.com website. Below each table there is browser support table outlining how each feature is supported across modern browsers. The + sign means “this version and above”, the – sign means “this version and below”. A green (or light gray) color denotes that the feature is supported. Red (or dark gray) marks the lack of browser support, while orange (or gray) was used to mark partial support. The browser support tables are found at <http://wufoo.com/html5/>.

HTML5 introduces 5 new elements related to input and forms.

Element	Purpose	Notes
progress	Represents completion of a task.	The progress element could represent the progress of a file being uploaded.
meter	Represents a scalar measurement within a known range.	The meter element could be used to represent something like a temperature or weight measurement.
datalist	Represents a set of option elements that can be used in combination with the new list attribute for input to make dropdown menus.	When the input with the associated datalist gets focus, a dropdown menu appears and contains the values from the datalist.
keygen	A control for key-pair generation.	When the form is submitted, the private key gets stored in the local keystore, and the public key is sent to

the server.








output Displays the results of a calculation. An example use of the output element could be to display the sum of the values of two input elements.

								
		Firefox	Safari	Safari	Chrome	Opera	IE	Android
Meter	?	11–	5.2+	5–	6+	11+	10–	2.3–
Progress	?	6+	5.2+	5–	6+	10.6+	10+	2.3–
Output	?	6+	5.1+	5+	13+	9.2+	10+	2.3+
Keygen	?	3.6+	4+	5–	6+	10.6+	10–	2.3+

HTML5 introduces 13 new input types. When viewed in a browser that doesn't support them, these input types fall back to text input.

Input Type	Purpose	Notes
tel	For entering a telephone number.	tel does not enforce a particular syntax, so if you want to ensure a particular format, you can use pattern or setCustomValidity() to do additional validation.
search	To prompt users to enter text that they want to search for.	The difference between search and text is primarily stylistic. Using an input type of search might result in the input field being styled in a way that is consistent with that platform's search fields.
url	For entering a single URL.	url is intended for entering a single, absolute URL, which represents a pretty wide range of values.
email	For entering either a single email address or a list of email addresses.	If the multiple attribute is specified, then multiple email addresses can be entered, separated by commas.
datetime	For entering a date and time with the time zone set to UTC.	
date	For entering a date with no time zone.	
month	For entering a date with a year and a month, but no time zone.	
week	For entering a date that consists of a week-year number and a week number, but no time zone.	An example of this format is 2011-W05 for the fifth week of 2011.








time	For entering a time value with hour, minute, seconds, and fractional seconds, but no time zone.	
datetime-local	For entering a date and time with no time zone.	
number	For numerical input.	Valid values are floating point numbers.
range	For numerical input, but unlike number, the actual is not important.	The implementation of the range control is a slider in most browsers that support it.
color	For choosing color through a color well control.	The value must be a valid lowercase simple color such as #ffffff.

								
		Firefox	Safari	Safari	Chrome	Opera	IE	Android
Email	?	4+	5+	3.1+	6+/10+	10.6+	10+	2.3-
Tel	?	4+	5+	3.1+	6+	10.6+	10+	2.3+
Url	?	4+	5+	3.1+	6+/10+	10.6+	10+	2.3-
Search	?	3.6/4+	5+	4+	6+	10.6+	9/10+	2.3-
Color	?	11-	5.2-	5-	20+	11+	10-	2.3-
Number	?	11-	4/5.2+	4+	9-/10+	11+	10-	2.3+
Range	?	11-	4+	5+	6+	9+	10+	2.3-
Date	?	11-	5+	5-	10/17/20+	10.6+	10-	2.3-

HTML5 also introduces several new attributes for the input and form elements.

Attribute	Purpose	Notes
autofocus	Focuses the input on the element when the page is loaded.	autofocus can be applied to input, select, textarea, and button.
placeholder	Gives the user a hint about what sort of data they should enter.	The placeholder value is displayed in light text until the element gets focus and the user enters some data. It can be specified on input and textarea.
form	Specifies one or more forms to which the input element belongs.	By using the form attribute, the input elements can be placed anywhere on the page, not just within the form element. Also, a single input element can be associated with more than one form.
required	A boolean attribute that means the element is required.	The required attribute is helpful for doing browser-based validation

autocomplete	For specifying that a field should not autocomplete or be pre-filled by the browser based on a user's past entries.	without using custom JavaScript. The autocomplete attribute for fields like a credit card number or one-time password, which you don't want autocomplete. By default, autocomplete is in the on state, so if you want to disable it, set it to off.
pattern	For validating an element's value against a regular expression.	When using a pattern, you should also specify a title value to give the user a description of the pattern that's expected.
dirname	For submitting the directionality of the control with the form.	For example, if the user entered text data with right-to-left directionality and the input element contained the dirname attribute, then an indication of the right-to-left directionality would be submitted along with the input value.
novalidate	For disabling form submission validation when specified on a form element.	
formaction	For overriding the action attribute on the form element.	This attribute is supported on input and button elements.
formenctype	For overriding the enctype attribute on the form element.	This attribute is supported on input and button elements.
formmethod	For overriding the method attribute on the form element.	This attribute is supported on input and button elements.
formnovalidate	For overriding the novalidate attribute on the form element.	This attribute is supported on input and button elements.
formtarget	For overriding the target attribute on the form element.	This attribute is supported on input and button elements.

		 Firefox	 Safari	 Safari	 Chrome	 Opera	 IE	 Android
Placeholder	?	4+	4/5+	4+	10+	11.10/11.50	10+	2.3+
Autofocus	?	4+	5+	5-	6+	11+	10+	2.3-
Maxlength	?	3.6/4+	5+	4+	6+	11+	9/10	2.3+
List (Datalist)	?	4+	5-	5-	20+	10.6+	10+	2.3-
Autocomplete	?	4+	5.2+	N/A	17+	10.6+	10-	2.3-
Required	?	6+	5-	5-	6+	10.6+	10+	2.3-
Pattern	?	4+	5-	5-	6/10+	10.6/11+	10+	2.3-
Spellcheck	?	3.6+	4+	5-	10+	11+	10+	N/A
Novalidate	?	4+	5-	5-	6+	10.6+	10+	2.3-
Formnovalidate	?	4+	5.2-	5-	6+	10.6+	10+	2.3-
Formaction	?	4+	5.2+	5+	10+	10.6+	10+	2.3-
Formmethod	?	4+	5.2+	5+	10+	10.6+	10+	2.3-
Formtarget	?	4+	5.2+	5+	10+	10.6+	10+	2.3-
Formenctype	?	?	?	?	?	?	?	?
Accept	?	11+	5.2+	N/A	10+	10.6+	10+	2.3-
Multiple	?	3.6+	5+	N/A	6+	11+	10+	2.3-
Min / Max / Step	?	11-	5+	5-	6+	10.6+	10+	2.3-

The Model For Automated Form Creation And Validation

Before starting to build the model, a few key requirements have to be set for the model to work. These are:

- The ability to build a form as a standalone HTML element
- The ability to build a form by supplying a collection of form elements
- The ability to build a form based on a database table
- The ability to add a certain field to an already built form
- The ability to retrieve an element from the form by id or elements by attribute value or tag name
- The ability to set filters on form fields, such as the field being required (the form will not validate if the field is empty), certain minimum and maximum lengths for the field values, validating an email, equality between two fields in the same form and so on
- The ability to build user-friendly forms
- The form also has to validate file uploads

Some of the requirements will be met if the model extends from the HTMLInputElement model introduced in an earlier article (JISOM 5.2). The HTMLInputElement has the following structure:

HTMLInputElement	
Property	Description
element	The type of the element itself; can be any of the supported (X)HTML elements, like an anchor, a paragraph, an image, a list and so on.
attributes	The attribute list of the element. Implemented as an array of key=>value pairs, with the key being the name of the attribute.
innerHTML	The inner text of the html element. This would be text if the element is a paragraph, for example, but it may also be html text like the list items in an unordered list. Elements that don't have a closing tag are not allowed to have innerHTML.
children	An array of instances of the HTMLInputElement class that are direct descendants of the current element. This greatly helps in keeping track and managing tree-like structures.
parent	A reference to the HTMLInputElement parent of the current element, if it has any. Also helps with nesting of html elements.
code	The rendering of the element as a string to be placed in a web page.
Method	Description
constructor	Optionally receives the tag name and a list of attributes for the element to be created
add_child	Requires an HTMLInputElement as an argument and appends that element to the end of children array of the current element. In effect it adds a child just before the closing tag of the current element.
add_child_after	Requires an HTMLInputElement and a neighbor id. Searches among the children of the current element for the child with the id given as the second argument and adds the first argument as a neighbor after it.
prepend_child	Requires an HTMLInputElement as the only argument and adds it as the first child of the current element.
get_child_by_id	Searches among the children of the current element and returns the one that has an id equal to the argument that the method received.
get_child_by_property	Searches for the first child of the current element that has an attribute with a value given as arguments to the method.

get_children_by_property	Returns all children with the attribute=>value pair supplied as arguments.
get_children_by_tag_name	Returns all children of the current element that are instances of a certain html element. Requires the tag name as an argument.
code	Method responsible with rendering the html string representation of the element with all attributes and children.

Building on the above structure, the Form model adds the following:

Form	
Property	Description
model or table	If the form being built reflects the structure of a database table or a model in an mvc architecture, this would be the name of that object.
fields	An array containing the names of the fields from the table or model that should be included in the form.
filters	An array of filters against which the form is validated. This array could be implemented as a series of arrays indexed by field id and having name, value and error message dimensions. It could also be implemented as objects of a type Filter.
submit	The string label of the submit button
description	An array of descriptions for fields in the form.
Method	Description
constructor	This is actually a very complex method. It builds the form as an HTMLElement and adds all the attributes and the children. If the form is not built from a database table, a set of options can be passed to the constructor. These options are the fields of the form, following a structure that provides the type of the element and other attributes.
from_model	If the form is being built from a table or model, this method reads the structure of the table or model and builds the elements based on column type. For example, for a MySQL table, fields of type varchar, int, date would be rendered as input elements with the type attribute set to text, enum fields would be rendered as select lists, text fields as textareas and so on.
check	The method receives data posted from the client and validates it against the filters set during the building of the form. If all filters pass validations the method returns true, otherwise it returns false. If any given field fails validation, the error message set when the filter was

	applied gets added to the form and classes are added or removed from the form element to provide more visual awareness to the user.
add_element	Add a new form element to an existing form. The element can be added at the end of the form, at beginning of the form, after a given element or before a given element.

Test Implementation Of The Form Model

Following are some examples of how the form class is used after the model has been implemented using the PHP programming language. Due to length, we cannot show the entire codebase for the implementation, but examples should help make an idea of how it works.

The following is an example of a form built using an array of options. The form contains text fields, radio buttons, file upload fields and select elements. Also, there are some filters applied that restrict the PNC field to exactly 13 characters long and only allow it to contain numbers and the file field is set to only accept jpeg and gif pictures. In addition, some fields are marked as required.

```
$form = new Form(
    array("method"=>"post", "action"=>"employees/add", "class"=>"add",
        "enctype"=>"multipart/form-data"
    ),
    array(
        "Personal information"=>
            array(
                array("element"=>"input", "type"=>"text", "id"=>"fname",
                    "name"=>"first_name", "label"=>"First name: ",
"required"=>true),
                array("element"=>"input", "type"=>"text", "id"=>"lname",
                    "name"=>"last_name", "label"=>"Last name: ",
"required"=>true),
                array("element"=>"input", "type"=>"radio",
                    "options"=>array("M"=>"M", "F"=>"F"), "id"=>"sex",
"name"=>"sex",
                    "label"=>"Sex: ", "required"=>false),
                array("element"=>"input", "type"=>"text", "id"=>"pnc",
"name"=>"pnc",
                    "label"=>"Personal num. code: ", "required"=>true),
                array("element"=>"input", "type"=>"file", "id"=>"avatar",
"name"=>"avatar",
                    "label"=>"Avatar: ", "required"=>false),
            "Employment information"=>
                array(
                    array("element"=>"select",
"options"=>array("1"=>"CEO", "2"=>"worker"),
                    "id"=>"id_position", "name"=>"id_position",
"label"=>"Position: ",
                    "required"=>false),
                    array("element"=>"input", "type"=>"text", "id"=>"hiring_date",
"name"=>"hiring_date", "label"=>"Hiring date: ",
```

```

        "readonly"=>"readonly", "class"=>"calendar",
"required"=>false)
    )
    ),
    "Add employee"
);
$form->filters = array(

    "pnc"=> array(
        "min"=>array("value"=>13, "error_message"=>"The PNC must be exaclty 13
chars"),
        "max"=>array("value"=>13, "error_message"=>"The PNC must be exaclty 13
chars "),
        "type"=>array("value"=>"numeric", "error_message"=>"Only numbers
allowed")
    ),
    "avatar"=> array(
        "file_type"=>array("value"=>array("gif","jpeg","jpg"),
"error_message"=>"The file must be gif or jp(e)g.")
    )
);
if($form->check($_POST, $_FILES)
    //form passed validation
else
    //form didn't pass validation and now contains error messages

```

Another usage scenario would be to build the form from a database table or an MVC model. The following example follows this scenario.

```

$form = new Form();
$form->model = "user";
$form->class = "input_edit";
$form->submit_label = "Modify!";
$form->method = "post";
$form->action = "cont/edit/cont";
$form->fields = array("username", "parola");
$form->required = array("parola");
$form->description = array("username"=>"The username for the account",
    "parola" => "The password");
$form->filters = array(
    "parola2" => array("equal" => array("value"=>"parola",
"error_message"=>"The password fields need to be identical."));
$form->from_model();
$parola2 = new HtmlElement("input", array("type"=>"text",
"id"=>"parola2", "name"=>"parola2"));
$parola2->label = "Retype password: ";
$parola2->description = "Please retype the password";
$parola2->required = "yes";
$parola2->type = "password";
$form->add_element($parola2, "append", "parola");
$form->get_child_by_id("parola")->type = "password";
$form->get_child_by_property("for","parola")->innerHTML = "Password: ";
$form->get_child_by_id("username")->disabled = "disabled";
$form->get_child_by_property("element", "legend")->innerHTML = "Account
info:";
if($form->check($_POST, $_FILES)
    //form passed validation

```

```
else
    //form didn't pass validation and now contains error messages
```

Conclusion

The proposed model makes form generation and validation a really easy job and takes a lot of work off the shoulders of developers. Automated validation is a huge gain when dealing with large forms or large numbers of forms. Also, because no HTML is actually written by the developer (aside from the `HTMLElement` class) standards compliance is ensured. Furthermore, when the model gets updated, all forms created using it automatically follow the updates.

The model could easily be used to create standalone classes or modules that are part of a greater MVC architecture.

Acknowledgement

This work was co-financed from the European Social Fund through Sectorial Operational Program Human Resources Development 2007-2013, projects POSDRU/107/1.5/S/77213 and POSDRU/88/1.2/S/55287 „Ph.D. for a career in interdisciplinary economic research at the European standards”

Bibliography

1. M. Pilgrim, HTML5: Up and Running, O'Reilly Media, August, 2010
2. <http://wufoo.com/html5/>
3. <http://www.html5rocks.com/en/tutorials/forms/html5forms/>
4. W. J. Gilmore, Easy PHP Websites, Columbus, Ohio: W.J. Gilmore, LLC, 2009.
5. D. Ciccarelli, „Web 2.0 Definition,” 16.09.2006. http://blogs.voices.com/thebiz/2006/09/web_20_definition.html
6. K. McArthur, Pro PHP Patterns, Frameworks, Testing and More, Apress, 2008.
7. A. Freeman, S. Sanderson, Pro ASP.NET MVC 3 Framework, Apress, 2011.
8. J. Galloway, P. Haack, B. Wilson, K. S. Allen, Professional ASP.NET MVC 3, John Wiley & Sons, Inc., 2011.
9. <http://thinkvitamin.com/code/fun-with-html5-forms/>