jQuery Form Plugin

Overview

The jQuery Form Plugin allows you to easily and unobtrusively upgrade HTML forms to use AJAX. The main methods, `ajaxForm` and `ajaxSubmit`, gather information from the form element to determine how to manage the submit process. Both of these methods support numerous options which allows you to have full control over how the data is submitted. Submitting a form with AJAX doesn't get any easier than this!

Quick Start Guide

1 Add a form to your page. Just a normal form, no special markup required:

```
<form id="myForm" action="comment.php" method="post">
    Name: <input type="text" name="name" />
    Comment: <textarea name="comment"></textarea>
    <input type="submit" value="Submit Comment" />
</form>
```

2 Include jQuery and the Form Plugin external script files and a short script to initialize the form when the DOM is ready:

```
<html>
<head>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.js"></script>
    <script src="http://malsup.github.com/jquery.form.js"></script>

    <script>
        // wait for the DOM to be loaded
        $(document).ready(function() {
            // bind 'myForm' and provide a simple callback function
            $('#myForm').ajaxForm(function() {
                alert("Thank you for your comment!");
            });
        });
    </script>
</head>
...
```

That's it!

When this form is submitted the name and comment fields will be posted to comment.php. If the server returns a success status then the user will see a "Thank you" message.

Form Plugin API

The Form Plugin API provides several methods that allow you to easily manage form data and form submission.

ajaxForm

Prepares a form to be submitted via AJAX by adding all of the necessary event listeners. It does **not** submit the form. Use ajaxForm in your document's ready function to prepare your form(s) for AJAX submission. ajaxForm takes zero or one argument. The single argument can be either a callback function or an Options Object.
Chainable: Yes.

Note: You can pass any of the standard $.ajax options to ajaxForm

Example:

```
$('#myFormId').ajaxForm();
```

ajaxSubmit

Immediately submits the form via AJAX. In the most common use case this is invoked in response to the user clicking a submit button on the form. ajaxSubmit takes zero or one argument. The single argument can be either a callback function or an Options Object.
Chainable: Yes.

Note: You can pass any of the standard $.ajax options to ajaxSubmit

Example:

```
// attach handler to form's submit event
$('#myFormId').submit(function() {
    // submit the form
    $(this).ajaxSubmit();
    // return false to prevent normal browser submit and page navigation
    return false;
});
```

formSerialize

Serializes the form into a query string. This method will return a string in the format: name1=value1&name2=value2
Chainable: No, this method returns a String.

Example:

```
var queryString = $('#myFormId').formSerialize();

// the data could now be submitted using $.get, $.post, $.ajax, etc
$.post('myscript.php', queryString);
```

fieldSerialize

Serializes field elements into a query string. This is handy when you need to serialize only part of a form. This method will return a string in the format: name1=value1&name2=value2
Chainable: No, this method returns a String.

Example:

```
var queryString = $('#myFormId .specialFields').fieldSerialize();
```

fieldValue

Returns the value(s) of the element(s) in the matched set in an array. As of version .91, this method **always** returns an array. If no valid value can be determined the array will be empty, otherwise it will contain one or more values.
Chainable: No, this method returns an array.

Example:

```
// get the value of the password input
var value = $('#myFormId :password').fieldValue();
alert('The password is: ' + value[0]);
```

resetForm

Resets the form to its original state by invoking the form element's native DOM method.
Chainable: Yes.

Example:

```
$('#myFormId').resetForm();
```

clearForm

Clears the form elements. This method emptys all of the text inputs, password inputs and textarea elements, clears the selection in any select elements, and unchecks all radio and checkbox inputs.
Chainable: Yes.

```
$('#myFormId').clearForm();
```

clearFields

Clears field elements. This is handy when you need to clear only a part of the form.
Chainable: Yes.

```
$('#myFormId .specialFields').clearFields();
```

ajaxForm and ajaxSubmit Options

Note: Aside from the options listed below, you can also pass any of the standard $.ajax options to ajaxForm and ajaxSubmit.

Both `ajaxForm` and `ajaxSubmit` support numerous options which can be provided using an Options Object. The Options Object is simply a JavaScript object that contains properties with values set as follows:

beforeSerialize
Callback function to be invoked before the form is serialized. This provides an opportunity to manipulate the form before it's values are retrieved. The `beforeSerialize` function is invoked with two arguments: the jQuery object for the form, and the Options Object passed into ajaxForm/ajaxSubmit.

```
beforeSerialize: function($form, options) {
    // return false to cancel submit
}
```

Default value: `null`

beforeSubmit
Callback function to be invoked before the form is submitted. The 'beforeSubmit' callback can be provided as a hook for running pre-submit logic or for validating the form data. If the 'beforeSubmit' callback returns false then the form will not be submitted. The 'beforeSubmit' callback is invoked with three arguments: the form data in array format, the jQuery object for the form, and the Options Object passed into ajaxForm/ajaxSubmit.

```
beforeSubmit: function(arr, $form, options) {
    // The array of form data takes the following form:
    // [ { name: 'username', value: 'jresig' }, { name: 'password', value: 'secret' } ]

    // return false to cancel submit
}
```

Default value: `null`

clearForm
Boolean flag indicating whether the form should be cleared if the submit is successful
Default value: `null`

data
An object containing extra data that should be submitted along with the form.

```
data: { key1: 'value1', key2: 'value2' }
```

dataType
Expected data type of the response. One of: null, 'xml', 'script', or 'json'. The `dataType` option provides a means for specifying how the server response should be handled. This maps directly to the `jQuery.httpData` method. The following values are supported:

'xml': if dataType == 'xml' the server response is treated as XML and the 'success' callback method, if specified, will be passed the responseXML value

'json': if dataType == 'json' the server response will be evaluted and passed to the 'success' callback, if specified

'script': if dataType == 'script' the server response is evaluated in the global context

Default value: `null`

error
Callback function to be invoked upon error.

forceSync
Boolean value. Set to true to remove short delay before posting form when uploading files (or using the iframe option). The delay is used to allow the browser to render DOM updates prior to performing a native form submit. This improves usability when displaying notifications to the user, such as "Please Wait..."
Default value: `false`
Added in v2.38

iframe
Boolean flag indicating whether the form should always target the server response to an iframe. This is useful in conjuction with file uploads. See the File Uploads documentation on the Code Samples page for more info.
Default value: `false`

iframeSrc
String value that should be used for the iframe's src attribute when/if an iframe is used.
Default value: `about:blank`
Default value for pages that use `https` protocol: `javascript:false`

iframeTarget
Identifies the iframe element to be used as the response target for file uploads. By default, the plugin will create a temporary iframe element to capture the response when uploading files. This options allows you to use an existing iframe if you wish. When using this option the plugin will make no attempt at handling the response from the server.
Default value: `null`

Added in v2.76

replaceTarget
Optionally used along with the the `target` option. Set to `true` if the target should be replaced or `false` if only the target contents should be replaced.
Default value: `false`
Added in v2.43

resetForm
Boolean flag indicating whether the form should be reset if the submit is successful
Default value: `null`

semantic
Boolean flag indicating whether data must be submitted in strict semantic order (slower). Note that the normal form serialization is done in semantic order with the exception of input elements of `type="image"`. You should only set the semantic option to true if your server has strict semantic requirements **and** your form contains an input element of `type="image"`.
Default value: `false`

success
Callback function to be invoked after the form has been submitted. If a 'success' callback function is provided it is invoked after the response has been returned from the server. It is passed the following arguments:

1.) responseText or responseXML value (depending on the value of the dataType option).
2.) statusText
3.) xhr (or the jQuery-wrapped form element if using jQuery < 1.4)
4.) jQuery-wrapped form element (or undefined if using jQuery < 1.4)

Default value: `null`

target
Identifies the element(s) in the page to be updated with the server response. This value may be specified as a jQuery selection string, a jQuery object, or a DOM element.
Default value: `null`

type
The method in which the form data should be submitted, 'GET' or 'POST'.
Default value: value of form's `method` attribute (or 'GET' if none found)

uploadProgress
Callback function to be invoked with upload progress information (if supported by the browser). The callback is passed the following arguments:

1.) event; the browser event
2.) position (integer)
3.) total (integer)
4.) percentComplete (integer)

Default value: `null`

url
URL to which the form data will be submitted.
Default value: value of form's `action` attribute

Example:

```
// prepare Options Object
var options = {
    target:     '#divToUpdate',
    url:        'comment.php',
    success:    function() {
        alert('Thanks for your comment!');
    }
};
```

```
// pass options to ajaxForm
$('#myForm').ajaxForm(options);
```

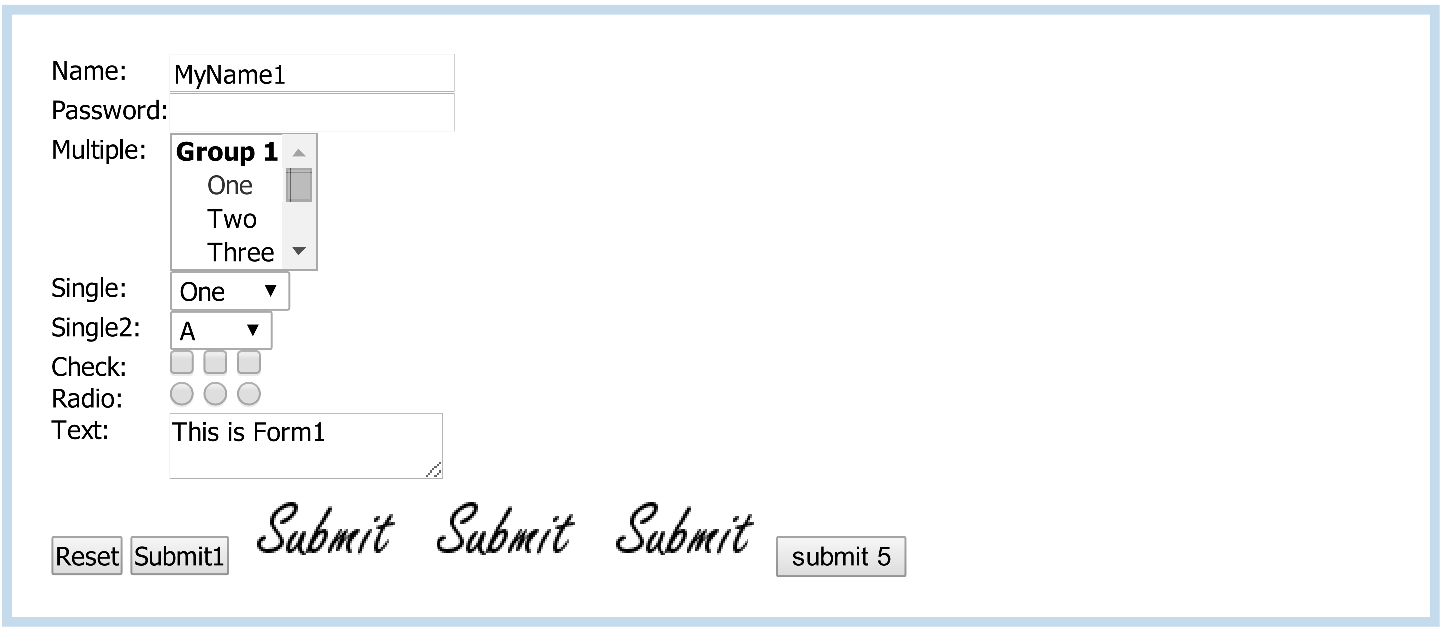
Note that the Options Object can also be used to pass values to jQuery's `$.ajax` method. If you are familiar with the options supported by `$.ajax` you may use them in the Options Object passed to `ajaxForm` and `ajaxSubmit`.

Code Samples

ajaxForm
ajaxSubmit
Validation
JSON
XML
HTML
File Uploads

The following code controls the HTML form beneath it. It uses `ajaxForm` to bind the form and demonstrates how to use pre- and post-submit callbacks.

```
// prepare the form when the DOM is ready
$(document).ready(function() {
    var options = {
        target:        '#output1',   // target element(s) to be updated with server response
        beforeSubmit:  showRequest,  // pre-submit callback
        success:       showResponse  // post-submit callback

        // other available options:
        //url:       url         // override for form's 'action' attribute
        //type:      type        // 'get' or 'post', override for form's 'method' attribute
        //dataType:  null        // 'xml', 'script', or 'json' (expected server response type)
        //clearForm: true        // clear all form fields after successful submit
        //resetForm: true        // reset the form after successful submit

        // $.ajax options can be used here too, for example:
        //timeout:   3000
    };

    // bind form using 'ajaxForm'
    $('#myForm1').ajaxForm(options);
});

// pre-submit callback
function showRequest(formData, jqForm, options) {
    // formData is an array; here we use $.param to convert it to a string to display it
    // but the form plugin does this for you automatically when it submits the data
    var queryString = $.param(formData);

    // jqForm is a jQuery object encapsulating the form element.  To access the
    // DOM element for the form do this:
    // var formElement = jqForm[0];

    alert('About to submit: \n\n' + queryString);

    // here we could return false to prevent the form from being submitted;
    // returning anything other than false will allow the form submit to continue
    return true;
}

// post-submit callback
function showResponse(responseText, statusText, xhr, $form)  {
    // for normal html responses, the first argument to the success callback
    // is the XMLHttpRequest object's responseText property

    // if the ajaxForm method was passed an Options Object with the dataType
    // property set to 'xml' then the first argument to the success callback
    // is the XMLHttpRequest object's responseXML property

    // if the ajaxForm method was passed an Options Object with the dataType
    // property set to 'json' then the first argument to the success callback
    // is the json data object returned by the server

    alert('status: ' + statusText + '\n\nresponseText: \n' + responseText +
        '\n\nThe output div should have already been updated with the responseText.');
}
```

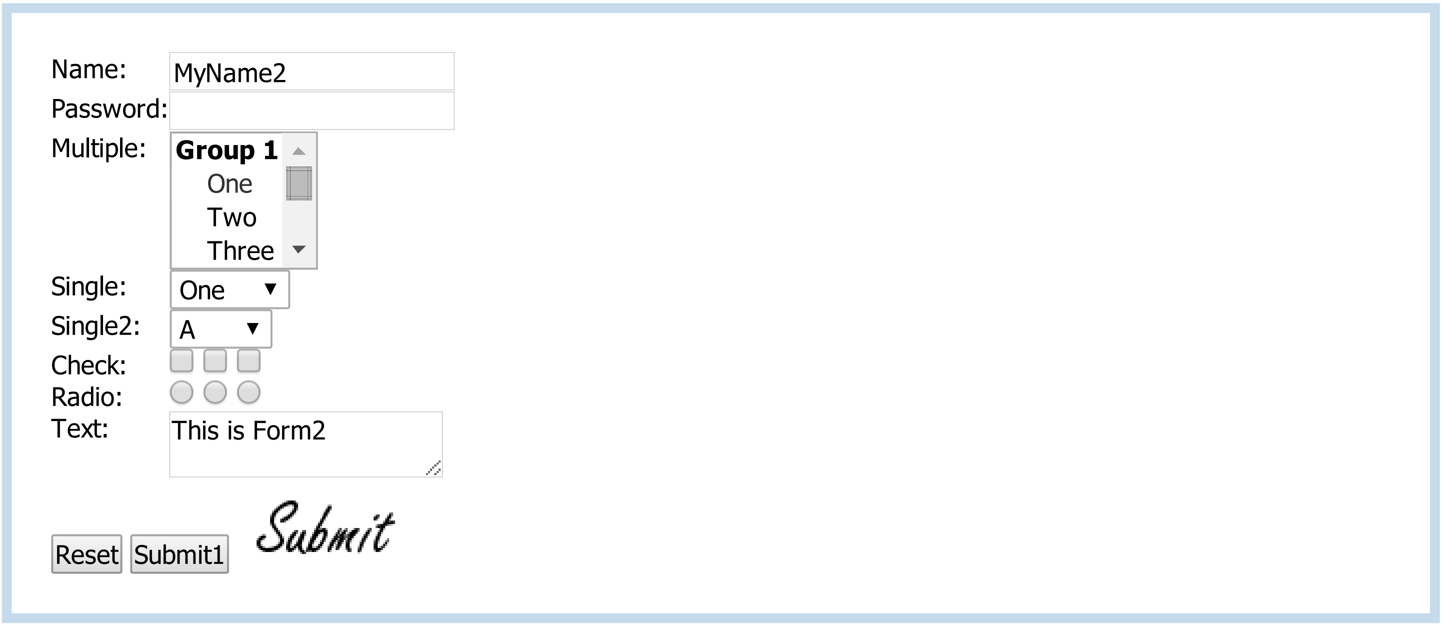

Name: MyName1
Password:
Multiple: Group 1 One Two Three
Single: One
Single2: A
Check:
Radio:
Text: This is Form1

Output Div (#output1):

AJAX response will replace this content.

The following code controls the HTML form beneath it. It uses `ajaxSubmit` to submit the form.

```
// prepare the form when the DOM is ready
$(document).ready(function() {
    var options = {
        target:        '#output2',   // target element(s) to be updated with server response
        beforeSubmit:  showRequest,  // pre-submit callback
        success:       showResponse  // post-submit callback

        // other available options:
        //url:       url         // override for form's 'action' attribute
        //type:      type        // 'get' or 'post', override for form's 'method' attribute
        //dataType:  null        // 'xml', 'script', or 'json' (expected server response type)
        //clearForm: true        // clear all form fields after successful submit
        //resetForm: true        // reset the form after successful submit

        // $.ajax options can be used here too, for example:
        //timeout:   3000
    };

    // bind to the form's submit event
    $('#myForm2').submit(function() {
        // inside event callbacks 'this' is the DOM element so we first
        // wrap it in a jQuery object and then invoke ajaxSubmit
        $(this).ajaxSubmit(options);

        // !!! Important !!!
        // always return false to prevent standard browser submit and page navigation
        return false;
    });
});

// pre-submit callback
function showRequest(formData, jqForm, options) {
    // formData is an array; here we use $.param to convert it to a string to display it
    // but the form plugin does this for you automatically when it submits the data
    var queryString = $.param(formData);

    // jqForm is a jQuery object encapsulating the form element.  To access the
    // DOM element for the form do this:
    // var formElement = jqForm[0];

    alert('About to submit: \n\n' + queryString);

    // here we could return false to prevent the form from being submitted;
    // returning anything other than false will allow the form submit to continue
    return true;
}

// post-submit callback
function showResponse(responseText, statusText, xhr, $form)  {
    // for normal html responses, the first argument to the success callback
    // is the XMLHttpRequest object's responseText property

    // if the ajaxSubmit method was passed an Options Object with the dataType
    // property set to 'xml' then the first argument to the success callback
    // is the XMLHttpRequest object's responseXML property

    // if the ajaxSubmit method was passed an Options Object with the dataType
    // property set to 'json' then the first argument to the success callback
    // is the json data object returned by the server

    alert('status: ' + statusText + '\n\nresponseText: \n' + responseText +
        '\n\nThe output div should have already been updated with the responseText.');
}
```



Name:	MyName2
Password:	
Multiple:	**Group 1** One Two Three
Single:	One
Single2:	A
Check:	☐ ☐ ☐
Radio:	○ ○ ○
Text:	This is Form2

Output Div (#output2):

AJAX response will replace this content.

This page gives several examples of how form data can be validated before it is sent to the server. The secret is in the `beforeSubmit` option. If this pre-submit callback returns false, the submit process is aborted.

The following login form is used for each of the examples that follow. Each example will validate that both the username and password fields have been filled in by the user.

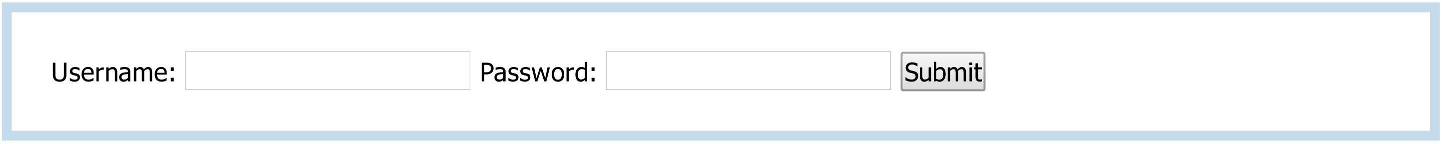
Form Markup:

```
<form id="validationForm" action="dummy.php" method="post">
    Username: <input type="text" name="username" />
    Password: <input type="password" name="password" />
    <input type="submit" value="Submit" />
</form>
```

First, we initialize the form and give it a `beforeSubmit` callback function - this is the validation function.

```
// prepare the form when the DOM is ready
$(document).ready(function() {
    // bind form using ajaxForm
    $('#myForm2').ajaxForm( { beforeSubmit: validate } );
});
```

Validate Using the `formData` Argument



```
function validate(formData, jqForm, options) {
    // formData is an array of objects representing the name and value of each field
    // that will be sent to the server;  it takes the following form:
    //
    // [
    //     { name:  username, value: valueOfUsernameInput },
    //     { name:  password, value: valueOfPasswordInput }
    // ]
    //
    // To validate, we can examine the contents of this array to see if the
    // username and password fields have values.  If either value evaluates
    // to false then we return false from this method.

    for (var i=0; i < formData.length; i++) {
        if (!formData[i].value) {
            alert('Please enter a value for both Username and Password');
            return false;
        }
    }
    alert('Both fields contain values.');
}
```

Validate Using the `jqForm` Argument

```
function validate(formData, jqForm, options) {
    // jqForm is a jQuery object which wraps the form DOM element
    //
    // To validate, we can access the DOM elements directly and return true
    // only if the values of both the username and password fields evaluate
    // to true

    var form = jqForm[0];
    if (!form.username.value || !form.password.value) {
        alert('Please enter a value for both Username and Password');
        return false;
    }
    alert('Both fields contain values.');
}
```

Validate Using the `fieldValue` Method

Username: [ ] Password: [ ] Submit

```
function validate(formData, jqForm, options) {
    // fieldValue is a Form Plugin method that can be invoked to find the
    // current value of a field
    //
    // To validate, we can capture the values of both the username and password
    // fields and return true only if both evaluate to true

    var usernameValue = $('input[name=username]').fieldValue();
    var passwordValue = $('input[name=password]').fieldValue();

    // usernameValue and passwordValue are arrays but we can do simple
    // "not" tests to see if the arrays are empty
    if (!usernameValue[0] || !passwordValue[0]) {
        alert('Please enter a value for both Username and Password');
        return false;
    }
    alert('Both fields contain values.');
}
```

Note
You can find jQuery plugins that deal specifically with field validation on the jQuery Plugins Page .

This page shows how to handle JSON data returned from the server.

The form below submits a message to the server and the server echos it back in JSON format.
Form markup:

```
<form id="jsonForm" action="json-echo.php" method="post">
    Message: <input type="text" name="message" value="Hello JSON" />
    <input type="submit" value="Echo as JSON" />
</form>
```

Message: Hello JSON Echo as JSON

Server code in `json-echo.php`:

```
<?php  echo '{ "message": "' . $_POST['message'] . '" }';  ?>
```

JavaScript:

```
// prepare the form when the DOM is ready
$(document).ready(function() {
    // bind form using ajaxForm
    $('#jsonForm').ajaxForm({
        // dataType identifies the expected content type of the server response
        dataType:  'json',

        // success identifies the function to invoke when the server response
        // has been received
        success:   processJson
    });
});
```

Callback function

```
function processJson(data) {
    // 'data' is the json object returned from the server
    alert(data.message);
}
```

This page shows how to handle XML data returned from the server.

The form below submits a message to the server and the server echos it back in XML format.
Form markup:

```
<form id="xmlForm" action="xml-echo.php" method="post">
    Message: <input type="text" name="message" value="Hello XML" />
    <input type="submit" value="Echo as XML" />
</form>
```

Message: Hello XML Echo as XML

Server code in `xml-echo.php`:

```
<?php
// !!! IMPORTANT !!! - the server must set the content type to XML
header('Content-type: text/xml');
echo '<root><message>' . $_POST['message'] . '</message></root>';
?>
```

JavaScript:

```
// prepare the form when the DOM is ready
$(document).ready(function() {
    // bind form using ajaxForm
    $('#xmlForm').ajaxForm({
        // dataType identifies the expected content type of the server response
        dataType:  'xml',

        // success identifies the function to invoke when the server response
        // has been received
        success:   processXml
    });
});
```

Callback function

```
function processXml(responseXML) {
    // 'responseXML' is the XML document returned by the server; we use
    // jQuery to extract the content of the message node from the XML doc
    var message = $('message', responseXML).text();
    alert(message);
}
```

This page shows how to handle HTML data returned from the server.

The form below submits a message to the server and the server echos it back in an HTML div. The response is added to this page in the `htmlExampleTarget` div below.
Form markup:

```
<form id="htmlForm" action="html-echo.php" method="post">
    Message: <input type="text" name="message" value="Hello HTML" />
    <input type="submit" value="Echo as HTML" />
</form>
```

Message: Hello HTML

Server code in `html-echo.php`:

```
<?php
echo '<div style="background-color:#ffa; padding:20px">' . $_POST['message'] . '</div>';
?>
```

JavaScript:

```
// prepare the form when the DOM is ready
$(document).ready(function() {
    // bind form using ajaxForm
    $('#htmlForm').ajaxForm({
        // target identifies the element(s) to update with the server response
        target: '#htmlExampleTarget',

        // success identifies the function to invoke when the server response
        // has been received; here we apply a fade-in effect to the new content
        success: function() {
            $('#htmlExampleTarget').fadeIn('slow');
        }
    });
});
```

htmlExampleTarget (output will be added below):

This page demonstrates the Form Plugin's file upload capabilities. There is no special coding required to handle file uploads. File input elements are automatically detected and processed for you.

Browsers that support the XMLHttpRequest Level 2 will be able to upload files seamlessly and even get progress updates as the upload proceeds. For older browsers, a fallback technology is used which involves iframes since it is not possible to upload files using the level 1 implmenentation of the XMLHttpRequest object. This is a common fallback technique, but it has inherent limitations. The iframe element is used as the target of the form's submit operation which means that the server response is written to the iframe. This is fine if the response type is HTML or XML, but doesn't work as well if the response type is script or JSON, both of which often contain characters that need to be repesented using entity references when found in HTML markup.

To account for the challenges of script and JSON responses when using the iframe mode, the Form Plugin allows these responses to be embedded in a `textarea` element and it is recommended that you do so for these response types when used in conjuction with file uploads and older browsers.

It is important to note that even when the dataType option is set to 'script', and the server is actually responding with some javascript to a multipart form submission, the response's Content-Type header should be forced to `text/html`, otherwise Internet Explorer will prompt the user to download a "file".

Also note that if there is no file input in the form then the request uses normal XHR to submit the form (not an iframe). This puts the burden on your server code to know when to use a textarea and when not to. If you like, you can use the `iframe` option of the plugin to force it to always use an iframe mode and then your server can always embed the response in a textarea. But the recommended solution is to test for the 'X-Requested-With' request header. If the value of that header is 'XMLHttpRequest' then you know that the form was posted via ajax. The following PHP snippet shows how you can be sure to return content successfully:

```
<?php
$xhr = $_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest';
if (!$xhr)
        echo '<textarea>';
```

```
?>

// main content of response here

<?php
if (!$xhr)
        echo '</textarea>';
?>
```

The form below provides an input element of type "file" along with a select element to specify the dataType of the response. The form is submitted to `files.php` which uses the dataType to determine what type of response to return.

File: 选择文件 未选择任何文件 Return Type: html ▼ Submit 1 Submit 2

Output:

Examples that show how to display upload progress:

Progress Demo 1
Progress Demo 2
Progress Demo 3

Working With Fields

This page describes and demonstrates the Form Plugin's `fieldValue` and `fieldSerialize` methods.

fieldValue

`fieldValue` allows you to retrieve the current value of a field. For example, to retrieve the value of the password field in a form with the id of 'myForm' you would write:

```
var pwd = $('#myForm :password').fieldValue()[0];
```

This method **always** returns an array. If no valid value can be determined the array will be empty, otherwise it will contain one or more values.

fieldSerialize

`fieldSerialize` allows you to serialize a subset of a form into a query string. This is useful when you need to process only certain fields. For example, to serialize only the text inputs of a form you would write:

```
var queryString = $('#myForm :text').fieldSerialize();
```

Demonstration

Enter a jQuery expression into the textbox below and then click 'Test' to see the results of the `fieldValue` and `fieldSerialize` methods. These methods are run against the test form that follows.

jQuery expression: :text (ie: textarea, [@type='hidden'], :radio, :checkbox, etc)
✓ Successful controls only [1]

Test

Test Form

<input type="hidden" name="Hidden" value="secret" />
<input name="Name" type="text" value="MyName1" /> MyName1
<input name="Password" type="password" />

```
<select name="Multiple" multiple="multiple">

<select name="Single">
<input type="checkbox" name="Check" value="1" />
<input type="checkbox" name="Check" value="2" />
<input type="checkbox" name="Check" value="3" />
<input type="checkbox" name="Check2" value="4" />
<input type="checkbox" name="Check2" value="5" />
<input type="checkbox" name="Check3" />
<input type="radio" name="Radio" value="1" />
<input type="radio" name="Radio" value="2" />
<input type="radio" name="Radio" value="3" />
<input type="radio" name="Radio2" value="4" />
<input type="radio" name="Radio2" value="5" />
<textarea name="Text" rows="2" cols="20"></textarea>

<input type="reset" name="resetButton" value="Reset" />
<input type="submit" name="sub" value="Submit" />
```

[1] http://www.w3.org/TR/html4/interact/forms.html#successful-controls .

By default, `fieldValue` and `fieldSerialize` only function on 'successful controls'. This means that if you run the following code on a checkbox that is not checked, the result will be an empty array.

```
// value will be an empty array if checkbox is not checked:
var value = $('#myUncheckedCheckbox').fieldValue();
// value.length == 0
```

However, if you really want to know the 'value' of the checkbox element, even if it's unchecked, you can write this:

```
// value will hold the checkbox value even if it's not checked:
var value = $('#myUncheckedCheckbox').fieldValue(false);
// value.length == 1
```

Frequently Asked Questions

What versions of jQuery is the Form Plugin compatible with?
The Form Plugin is compatible with jQuery v1.5 and later.

Does the Form Plugin have any dependencies on other plugins?
No.

Is the Form Plugin fast? Does it serialize forms accurately?
Yes! See our comparison page for a look at how the Form Plugin compares to other libraries (including Prototype and dojo).

What is the easiet way to use the Form Plugin?
`ajaxForm` provides the simplest way to enable your HTML form to use AJAX. It's the one-stop-shopping method for preparing forms.

What is the difference between `ajaxForm` and `ajaxSubmit`?
There are two main differences between these methods:
`ajaxSubmit` submits the form, `ajaxForm` does not. When you invoke `ajaxSubmit` it immediately serializes the form data and sends it to the server. When you invoke `ajaxForm` it adds the necessary event listeners to the form so that it can detect when the form is submitted by the user. When this occurs `ajaxSubmit` is called for you.
When using `ajaxForm` the submitted data will include the name and value of the submitting element (or its click coordinates if the submitting element is an image).

How can I cancel a form submit?

You can prevent a form from being submitted by adding a 'beforeSubmit' callback function and returning false from that function. See the Code Samples page for an example.

Is there a unit test suite for the Form Plugin?
Yes! The Form Plugin has an extensive set of tests that are used to validate its functionality. Run unit tests.

Does the Form Plugin support file uploads?
Yes!

Why aren't all my input values posted?
jQuery form serialization aheres closely to the HTML spec. Only successful controls are valid for submission.

How do I display upload progress information?
Demo

Download

The Official Form Plugin is available here: jquery.form.js or from the plugin's Github repository .

Minified version: jquery.form.min.js

There are many other useful Form Plugins available from the jQuery Plugins page.

Support

Support for the Form Plugin is available through the jQuery Google Group . This is a very active group to which many jQuery developers and users subscribe.

Contributors

Development of the Form Plugin was a community effort with many people contributing ideas and code. The following people have made contributions of one kind or another:

John Resig
Mike Alsup
Mark Constable
Klaus Hartl
Matt Grimm
Yehuda Katz
Jörn Zaefferer
Sam Collett
Gilles van den Hoven
Kevin Glowacz
Alex Andrienko

Send me an email if I've forgotten someone.

This site is maintained by Mike Alsup.