



Universitat d'Alacant
Universidad de Alicante

Advanced Web Programming

Jaume Aragonés Ferrero

**Department of Software
and Computing Systems**



A compact JavaScript framework

MOOTOOLS



Index

- What is MooTools?
- Where to find?
- How to download?
- Hello World example.
- A little bit of knowledge.
- Creating a console.
- Form data validation.
- Exercises.



What is MooTools?

- MooTools is a JavaScript framework
 - MooTools is a lightweight abstraction layer between the browser and the code you write.
- It is Object-Oriented
- It allows to write cross-browser code.
- It respects standards and it's extensively documented.
- It is released under Open Source MIT license



What is MooTools?

- **Wikipedia:**
 - MooTools is an open source, lightweight, modular object-oriented programming JavaScript web application framework.
 - It includes built-in functions for manipulation of CSS, DOM elements, native JavaScript objects, Ajax requests, and more.



What is a Framework?

- It is an abstraction in which common code providing generic functionality can be overridden by user code providing specific functionality.
- The idea is similar to SW libraries: they are reusable abstractions of code in a well defined API.
- In other words: it is a SW layer between the programmer and the developing platform in order to help building programs.



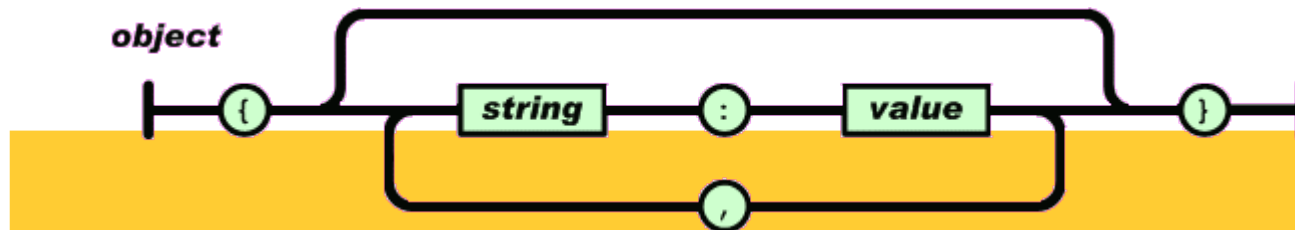
What can I do?

- Some examples of things you can do with MooTools:
 - Code Javascript easily: Mootools features improve Javascript capabilities.
 - Code Client-Side Object Oriented: Create your own classes with Javascript.
 - Work with JSON: sending and receiving data.
 - Create FX with Javascript: user interaction, dynamic menus, transitions (tween, morphs), slides, scrolls, etc.
 - Work with AJAX and interact with your server scripts.



What is JSON?

- JSON stands for Javascript Object Notation
- It is a data interchange format
- JSON information is in text format
- It is supported by many languages: Javascript, C, PHP, ASP.NET, Java, Python, etc.
- JSON data takes this form:





Where to find?

- <http://mootools.net>, home page.
- <http://mootools.net/download>, download page.
- <http://mootools.net/docs>, documentation page.
- <http://www.mootorial.com/wiki>, a good tutorial



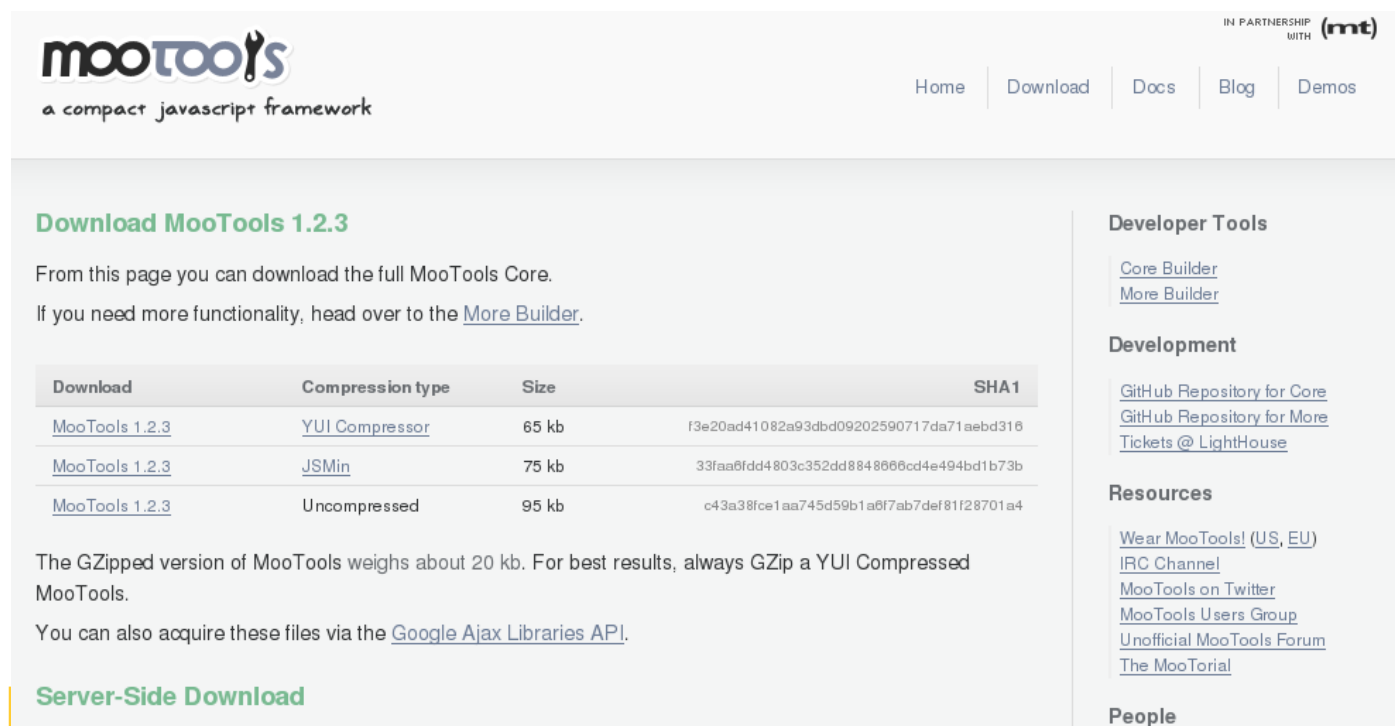
How to download?

- Go to www.mootools.net/download
- Here you can download the MooTools core.
- Use the uncompressed version for development and the compressed one for the real site.
- The core download includes: library base, ajax working, effects, classes and extensions.



How to download?

- The download page:



The screenshot shows the MooTools website's download page. At the top left is the MooTools logo with the tagline 'a compact javascript framework'. To the right is a navigation menu with links for Home, Download, Docs, Blog, and Demos. In the top right corner, it says 'IN PARTNERSHIP WITH (mt)'. The main content area features a heading 'Download MooTools 1.2.3' followed by instructions on how to download the full MooTools Core or the More Builder. Below this is a table with columns for Download, Compression type, Size, and SHA1. The table lists three download options: MooTools 1.2.3 with YUI Compressor (65 kb), MooTools 1.2.3 with JSMIn (75 kb), and MooTools 1.2.3 Uncompressed (95 kb). To the right of the table are sections for Developer Tools, Development, Resources, and People, each with several links to external resources.

IN PARTNERSHIP WITH (mt)

Home | Download | Docs | Blog | Demos

Download MooTools 1.2.3

From this page you can download the full MooTools Core.
If you need more functionality, head over to the [More Builder](#).

Download	Compression type	Size	SHA1
MooTools 1.2.3	YUI Compressor	65 kb	f3e20ad41082a93dbd09202590717da71aebd316
MooTools 1.2.3	JSMIn	75 kb	33faa0fdd4803c352dd8848666cd4e494bd1b73b
MooTools 1.2.3	Uncompressed	95 kb	c43a38fce1aa745d59b1a0f7ab7def81f28701a4

The GZipped version of MooTools weighs about 20 kb. For best results, always GZip a YUI Compressed MooTools.

You can also acquire these files via the [Google Ajax Libraries API](#).

Server-Side Download

Developer Tools

- [Core Builder](#)
- [More Builder](#)

Development

- [GitHub Repository for Core](#)
- [GitHub Repository for More](#)
- [Tickets @ LightHouse](#)

Resources

- [Wear MooTools! \(US, EU\)](#)
- [IRC Channel](#)
- [MooTools on Twitter](#)
- [MooTools Users Group](#)
- [Unofficial MooTools Forum](#)
- [The MooTorial](#)

People



How to download?

- The 'more Builder' link allows you to get the MooTools Add-ons:
 - Classes
 - Native Objects, Element classes
 - Forms
 - FX, Drag, Request
 - Utilities, Interface, etc.
- Just check the scripts you want, choose a compression type and then download the include file.

How to download?

- The 'more Builder' page with some options:

Core			
<input type="checkbox"/>	More	Defines MooTools.More.version.	?
<input type="checkbox"/>	Lang	Enables classes to contain text that can be localized to specific languages.	?
<input type="checkbox"/>	Log	A default logger for MooTools. Logs to Firebug or a similar console unless it is not present.	?
Class			
<input type="checkbox"/>	Class.Refactor	Allows for a class to extend itself without damaging it's namespace.	?
<input type="checkbox"/>	Class.Binds	Adds the Binds Mutator to all classes.	?
<input type="checkbox"/>	Class.Occlude	Mixin class for preventing a class from applying itself to the same element twice.	?
<input type="checkbox"/>	Chain.Wait	Adds a delay method for chaining that allows you to easily introduce pauses.	?
Native			
<input type="checkbox"/>	Array.Extras	Extends the Array native object to include useful methods to work with arrays.	?
<input type="checkbox"/>	Date	Extends the Date native object to include methods useful in managing dates.	?
<input type="checkbox"/>	Date.Extras	Extends the Date native object to include extra methods (on top of those in Date.js).	?
<input type="checkbox"/>	Hash.Extras	Extends the Hash native object to include getFromPath which allows a path notation to child elements.	?
<input type="checkbox"/>	String.Extras	Extends the String native object to include methods useful in managing strings (query strings, urls, etc).	?
<input type="checkbox"/>	String.QueryString	...	?

[GitHub Repository for More](#)
[Tickets @ LightHouse](#)

Resources

[Wear MooTools! \(US, EU\)](#)
[IRC Channel](#)
[MooTools on Twitter](#)
[MooTools Users Group](#)
[Unofficial MooTools Forum](#)
[The MooTorial](#)

People

[MooTools Developers](#)



How to download?

- Tip: choose the whole library for development only if you don't know which script you will need to use.
- Compression types remove extra blank spaces and rename variables in a shorter way, making a much smaller file.
 - YUI compressor is the most efficient. It will be very useful when releasing your code.



Before start working...

- Firebug:
 - Extension for Firefox
 - allows debugging, editing, and monitoring CSS, HTML, DOM, and JavaScript code, and provides other Web development tools.
 - <https://addons.mozilla.org/en-US/firefox/addon/1843>



Before start working...

- Web developer toolbar:
 - Extension for Firefox
 - adds a menu and a toolbar with various web developer tools.
 - <https://addons.mozilla.org/en-US/firefox/addon/60>



How to add to my pages

- It's easy, just include the MooTools file with the 'script' tag.

```
<script type="text/javascript"
src="path/to/MooTools.js"></script>
```

- You can also include your own javascript files or even code javascript lines inside the web page.
- The MooTools file include must be the first one in your web page.



How to add to my pages

- Here is an example document

```
<!DOCTYPE ...>
<html ...>
<head>...
<script type="text/javascript"
  src="path/to/MooTools.js"></script>
<script type="text/javascript"
  src="path/to/MyJSFile.js"></script>
<script>My own Javascript code</script>
</head>
<body>...</body>
</html>
```



Hello world example

- First, write a 'Hello world' page with only HTML.
- After, write a second page using javascript&DOM.
- Finally, let's display a 'hello world' message from MooTools code:

```
<script>
window.addEvent("domready", function() {
  $('text').appendText("Hello World! (from
    MooTools code!).");
});
</script>
```

- And the body code:

```
<p id="text"></p>
```



Hello world example

- `AddEvent`: allows to match a handler function to an HTML element event.
- `Domready`: is the event triggered when the DOM is ready and it has loaded all code and data.
- `$('#id-of-tag').method-or-property`: when we want to access to an element's method or property.
- `AppendText`: a method allowing us to add some text inside an element.



A little bit of knowlegde

- <http://mootools.net/docs/core/Element/Element>
- `$('#theID')`, select one element identified by 'theID'.
- `$$('something')`, select one or more elements matching 'something' string.
 - It returns an array or collection of elements
 - You can use tags names, classes, identifiers, etc.
- Event 'DomReady', It fires when the DOM (and all of its objects) is ready. So, in this moment you are sure you can work with all objects in the page.
 - This event is only available from the window Element.

A little bit more of knowlegde

- `$('#myID').getElement(),`
- `$('#myID').getElements()`
- `$('#myID').getElementsById()`
- Get children elements from our 'myID' element.
- `$('#myID').get('property')`
- `$('#myID').set('attribute', 'value')`
 - Set a value into a property of our 'myID' element.

A little bit more of knowlegde

- `myElement.erase(property)`, deletes the element's property.
- `myParent.adopt(newChild, ...)`
 - insert 'newChild' as child of 'myParent' element.
- `myElement.inject(element2, where)`
 - Insert 'myElement' in 'where' position from 'element2'.
- Others: `appendText`, `dispose`, `replaces`, etc.



Core functions

- <http://mootools.net/docs/core/Core/Core>
- `$$('aTag').each(new function {something...})`
 - Calls the function for each element in the collection retrieved by `$$`.
- `$chk`, `$defined`, `$type`, `$random`, etc.
- `$A`, creates an iterable array.
- `$H`, creates a hash table (`=Hash()`)



Creating a console

- Let's create a 'console', that is, a layer allowing us to put some text in order to debug or to display results of some actions.
- Here is the needed code:

```
function toConsole(theText) {  
    if (!$('console')) {  
        var element = new Element('div',  
        {'id': 'console'});  
        $$("body")[0].adopt(element);  
    }  
    var element = new Element('div');  
    element.appendText(theText);  
    $("console").adopt(element);  
}
```



Creating a console

- You can write this code directly inside the HTML file or put it into an external javascript file, and then include it from the web page (the best choice).
- Once you have typed the previous code, let's test it by creating a new page with
 - a command button (`<input type="button" ...`) with an 'id' attribute.
 - coding the 'window.onload' event and attaching a handler function to our button's 'onclick' event.



Creating a console

- The button code:

```
<input type="button" value="click me!"  
  id="write" />
```

- The handler code:

```
window.onload = function() {  
    var button=$('#write');  
    button.onclick=function() {  
        toConsole('You have clicked on  
the button, so I am writing in the  
console...');  
        return false;  
    }  
}
```



Try to do yourself...

- What happens if you click the button several times?
- Modify the code in order to put a counter counting the times the button is clicked.
- Modify the code in order to clean the console each time the button is clicked.



Defining classes

```
Var myClass= new Class({  
  MyAttr: 'value',  
  ...  
  Initialice: function(...){  
    ...},  
  MyMethod: function(...) {  
    ...}  
});  
Var myObj = new myClass(...);  
myObj.MyMethod(..);
```



Form data validation

- One of the most useful things we can do with javascript is to validate correctness of HTML forms.
- Using MooTools we can also validate HTML forms.
- We can build a class containing methods for validate every type of field in our forms:
 - _ Mandatory fields (text boxes, selects, radiobuttons)
 - _ Text boxes containing specific data (numeric, date, an interval of values, etc.)
 - _ Text boxes and text areas with a maximum length.
 - _ And any other kind of validation.



Form data validation

- Let's create our own form validation class:
 - First, we will design a one mandatory field form called user name, and a command button to send the form data.

Testing a form

Your name:



Form data validation

- The next is to implement our class

```
var cTestOK = new Class ({  
  error1: 'This field cannot be empty.',  
  isEmpty: function(pField) {  
    var field = $(pField);  
    if (field.get("value")== "") {  
      alert(this.error1);  
      return true;  
    } else {  
      return false;  
    }  
  }  
});
```

This class has

- one attribute (a message error)
- and one method: a function receiving a field name to test its emptiness.



Form data validation

- Finally, we are going to handle the onclick button event to trigger our class method.

```
window.addEventListener("domready", function() {  
    var oTesting = new cTestOK("myForm");  
    $("sending").addEventListener("click", function()  
    {oTesting.isEmpty("pName");});  
});
```

- This instruction allows us to link a function to the onclick button event.
- In this way, we will invoke the method 'isEmpty' of our class and we send it, as an input argument, the name of the field we want to test its correctness.



Try to do yourself...

- Add a select control with a product list.
 - This field is also mandatory.
- Add a new text field to the form asking for the amount of products the user wants to buy.
 - This field will have a default value: 1.
 - This field is also mandatory.
 - This field is a numeric value starting from 1.
- Finally, add a textarea field asking for any comment about the user order.
 - This field is optional.
 - This field has a maximum length of 200 chars.



References

- Webs:
 - mootools.net
 - www.mootorial.com
 - json.org
 - wikipedia.org
- Books:
 - MootTools Essentials, Aaron Newton. Apress / firstPress. 2008



Exercices

- Managing select options: changing options depending on other select value.
- Accordion FX.
- Tween FX.
- Sorting lists
- A game: guess the words I've thought about...



Managing select options

- Let's make a web page with two select fields, the content of one of them depending on the other's selected option.
- Suppose we have two selects like shown in the figure:

choose a country ↕

choose a city ↕



Managing select options

- First of all, we declare an array of cities for each country in the select.

```
var theWorld = [  
  $H({1: 'Warszawa', 2: 'Bialystok', 3:  
    'Lublin', 4: 'Krakow'}),  
  $H({1: 'Madrid', 2: 'Barcelona', 3:  
    'Alicante', 4: 'Valencia'}),  
  $H({1: 'San Francisco', 2: 'Washington',  
    3: 'New York', 4: 'Los Angeles'})];
```



Managing select options

- Next, we attach a handler to the 'onchange' event of the country select.

```
window.addEvent("domready",  
    function() {  
var country = $("countries");  
  
country.addEvent("change", function()  
    {putCities();});  
});
```

Managing select options

- Finally, we implement the handler.
- It will change all the options in the city select loading the selected country cities.
 - Test if the selected country is the 'non-selection' value '-1', and decide what to do if so.



Accordion menu

- Let's create a dynamic menu with expanding/contracting options, a.k.a. Accordion menu.
- First: type the HTML code of our menu.
 - The first level options
 - For each first level option, type its second level options.
- Second: label all the first level options with a CSS class, i.e.: level1.
 - Do the same with each element containing the second level options: 'level2' CSS class.



Accordion menu

- Finally: create and Accordion object in the 'Domready' event.

```
window.addEvent("domready",  
function() {  
    var myAccordion = new  
    Accordion($$("li.level1"), $$  
    ("ul.level2"));  
});
```



Tween FX

- Tween FX change a CSS property from one value to another.
- We can set the duration, start and end values, property to change, etc.

```
window.addEventListener("domready", function(){  
var myFX = new Fx.Tween($('myP'), {property:  
  'left', duration: 3000})  
myFX.start(0, 100);  
});
```

- **And in the Body**

```
<p id="myP" style="position: absolute; left:0px;  
top:0px;">Hello world!</p>
```



Sorting lists

- Design a list of items with HTML.
- Add two buttons to each item.
 - One of them will be the up button and the other the down one.
- Implement client-side code with MooTools to add the necessary functionality



Sorting lists

- Design a list of items with HTML

```
<div id="toSort">  
<div id="1" class="item">First item</div>  
<div id="2" class="item">Second Item</div>  
<div id="3" class="item">Third item</div>  
<div id="4" class="item">Fourth item</div>  
</div>
```



Sorting lists

- Add the up&down buttons
- Each item will change as follows:

```
<div id="1" class="item">  
<div class="img UP"></div>  
<div class="img DW"></div>  
First item  
</div>
```



Sorting lists

- Define some CSS styles to adjust the layout:

```
.img {  
float:left;  
width: 17px;  
height: 17px;  
padding: 2px;  
background-repeat: no-repeat;  
background-position: 0px 0px;}  
.UP { background-image: url('up.png'); }  
.DW { background-image: url('down.png'); }  
.item { clear: both; vertical-align: top; }
```



Sorting lists

- Program the sorting functionality.
 - For each image, add a handler to its event 'onclick'.
 - If it is a get-up image, add a 'get up' event
 - If not, then add a 'get down' event.
 - Implement both previous events: 'get up' and 'get down'.
 - 'get up' event will move the list item to the next, if exists.
 - 'get down' event will put the clicked list item before its previous, if exists.