Contents

Overview

MyUI is a JavaScript framework that provides a set of components to improve the web user experience. It was developed to be very easy to use and understand. Actually for most of the components you don't even need to have JavaScript knowledge, as they actually modify the behavior of normal HTML elements in a transparent way. Usually this is achieved by marking these elements with CSS classes or custom attributes.

At the base of MyUI is a core JavaScript cross-browser framework. But this is not done by adding an extra layer on top of the current one, but instead porting the W3C standard methods (like addEventListener or removeEventListener) and properties (like style.opacity) to the non-supporting browsers. This was the preferred approach so that the learning curve to be reduced to a minimum level. Not all the standards are ported to all the non-supporting browsers, but the future intent is to.

What we mean by browsers? Usually I refer to Firefox >= 2, IE >= 6 and Safari >= 3. Also Google Chrome is in the scope, but not focused on. These are practically all the browsers used on the Internet space, all others having a market share smaller than 1%. We better focus our efforts in adding new capabilities, than supporting almost unused browsers.

One of the main cares is also directed into the documentation, which should be as short as possible, gradual, easy to understand and with enough commented examples. MyUI has a lot of features, but please use only the ones in this documentation, not the ones learned by inspecting the code. Although this is not forbidden in any way, the undocumented features are always subject to change. From this document you should read the Overview, Installation and then you can jump to a specific chapter, depending on which component you want to use.

Installation

First of all you have to download MyUI from the official site: my-ui.sf.net. Then you must copy the `myui` directory from the distribution under your site, let's say under `myui` directory.

In order to enable MyUI for your web page, all you need is to reference the main JavaScript file. Below is the minimal code needed to add all the MyUI features to your webpage:

```
<script src="myui/myui.js" defer="false"></script>
```

It is HIGHLY recommended to add this in the `HEAD` tag. It is also highly recommended (and even mandatory in IE8) to have a DOCTYPE for your document. My personal preference is XHTML.
This will automatically include all the necessary scripts and style sheets, but they need to be located in the same directory as the `myui.js` file, in this case `myui`.
What about if you need only one functionality from MyUI? Why loading all the scripts and stylesheets? To improve performance you can specify which modules should be loaded. This is done using the `modules` parameter in the URL for the myui.js file, which represents a comma-separated list of the needed modules.

```
<script   src="myui/myui.js?modules=popup,   slideshow,   validation"
defer="false"></script>
```

This will load all the necessary files for the popup and slideshow modules, along with all the dependencies.
The recognized modules are:

- autossugest
- cloud
- date
- daySelector
- effects
- frames
- log4js
- popup
- progressbar
- slideshow
- tabletree
- validation

Logging

Logging is a very important part of an application, including web ones.
MyUI offers a logging framework very similar to the Java one: java.util.logging. This framework is however more lightweight.

Message architecture

Logging is used as a special communication system. Every message has a level associated. The standard levels are:

- `FATAL`, accessed with `Level.FATAL`
- `ERROR`, accessed with `Level.ERROR`
- `WARN` or `WARNING`, accessed with `Level.WARN` or `Level.WARNING`
- `INFO`, accessed with `Level.INFO`
- `DEBUG`, accessed with `Level.DEBUG`

Logging a message

This part is actually very simple. You will use the method `log` from `Logger`. But first you have to obtain a logger using the method `getLogger` and a logger name. The logger name is hierarchical and the separator is dot(.), like the Java packages.

```
Logger.getLogger("com.x.y").log(Level.INFO,   "Just   testing   the
logging");
```

Hierarchically, the logger `com.x` is the parent of `com.x.a` and `com.x.b`. There is also a root logger, ancestor of all loggers, obtained by passing no parameter to the `getLogger` method:

```
Logger.getLogger().log(Level.INFO, "I'm logging on the root");
```

There are also the convenience methods `debug`, `info`, `warn`, `error`, `fatal` to help you log messages with a specific level.

```
Logger.getLogger("com.x.y").info("Just testing the logging");
```

is equivalent with the previous statement.

Setting a logger

Every logger has an associated level. If the messages have the level lower than the one of the logger, they are logged, otherwise dropped. You can access the level of a logger using the getter/setter methods: `getLevel/setLevel.` For actually displaying the message, the logger is using log handlers, that can be added or removed to a logger using `addHandler` and `removeHandler`.

Log levels

Every log level has a name and a value associated with it. The default levels are: **DEBUG(500)**, **INFO(400)**, **WARN(300)**, **ERROR(200)**, **FATAL(100)**. You can access them using `Level.DEBUG`, `Level.INFO`, `Level.WARN`, `Level.ERROR` and `Level.FATAL`.
You can also create your own custom levels:

```
var myLevel = new Level("BLAH", 450);
Logger.getLogger("com.x").setLevel(Level.DEBUG);
Logger.getLogger("com.x").log(myLevel, "Blah blah");
```

Log handlers

These are used for publishing log messages. This is in fact an object which has the `publish` method associated with it. Every time a message should be logged by a logger, this calls the method `publish`. You can easily define your own handler, in this case to display an alert with the message:

```
function AlertMessageHandler() {
    this.publish = function(logger, level, message, time) {
        alert("[" + level + "] " + message);
    }
}
```

```
Logger.getLogger("com.x").addHandler(new AlertMessageHandler());
Logger.getLogger("com.x").debug("Hello");
```

The above code will display an alert with the message `[DEBUG] Hello`.
For your convenience the below handlers are already defined:

`WindowLogHandler`
: Logs messages to a window. You can pass a window to the constructor or a default one will be used.

`NoEchoLogHandler`
: No output.

`ElementContentLogHandler`
: Appends messages to the content of an HTML element. The HTML element or its ID must be passed as the first argument of the constructor.

`ListLogHandler`
: Appends messages as list items to an UL/OL HTML element. The HTML list element or its ID must be passed as the first argument of the constructor.

`AlertLogHandler`
: Displays a box for every message.

`WindowStatusLogHandler`
: Logs messages to a window status bar. You can pass a window to the constructor or the current one will be used.

`FileLogHandler`
: Logs messages to a file. The file path must be passed as the first parameter of the constructor. As this is using ActiveX, it works only on IE.

Date formatting and parsing

Date formatting and parsing is performed using the `DateFormat` class. Such an object is created using

```
var dateFormat = new DateFormat();
```

or

```
var dateFormat = new DateFormat(pattern);
```

where pattern describes the date format pattern. The main two methods are `parse(text)` which parses the given text returns a date or undefined if the text isn't in the right format, and `format(date)` which formats the given date and returns a string The date pattern syntax in described below:

Element	Syntax	Format	Examples
Year	yyyy	4 digits	1978, 2005
	yy	2 digits	78, 05
	y	2 or 4 digits	
Month	MMM	name or abbreviation	March, September
	MM	2 digits	03, 09, 12

	M	1 or 2 digits	3, 9, 12
	NNN	abbreviation	March, September
Day of Month	dd	2 digits	05, 13, 27
	d	1 or 2 digits	5, 13, 27
Day of Week	EEE	name	Monday, Saturday
	EE	abbreviation	Mon, Sat
	E	first letter	M, S
Hour (1-12)	hh	2 digits	01, 05, 12
	h	1 or 2 digits	1, 5, 12
Hour (0-23)	HH	2 digits	00, 04, 23
	H	1 or 2 digits	0, 4, 23
Hour (0-11)	KK	2 digits	00, 04, 11
	K	1 or 2 digits	0, 4, 11
Hour (1-24)	kk	2 digits	01, 05, 24
	k	1 or 2 digits	1, 5, 24
Minute	mm	2 digits	03, 23, 57
	m	1 or 2 digits	3, 23, 57
Second	ss	2 digits	07, 09, 34
	s	1 or 2 digits	7, 9, 34
AM/PM	A	uppercase	AM, PM
	a	lowercase	am, pm

As you saw, in the formatting and parsing, it appears some month and weekday names and abbreviations. The value of these can be customized using the `DateFormat` properties. The properties are simple arrays with the corresponding number of elements (month names will be an array of 12 elements). This is useful especially for localization. E.g. the below code will create a `DateFormat` for formatting and parsing in Romanian:

```
    this.monthNames = new Array("Ianuarie", "Februarie", "Martie",
"Aprilie", "Mai", "Iunie", Iulie", "August", "Septembrie",
"Octombrie", "Noiembrie", "Decembrie");
    this.monthShortNames = new Array("Ian", "Feb", "Mar", "Apr",
"Mai", "Iun", Iul", "Aug", "Sep", "Oct", "Nov", "Dec");
    this.dayOfWeekNames = new Array("Luni", "Marti", "Miercuri",
"Joi", "Vineri", "Simbata", "Duminica");
    this.dayOfWeekShortNames = new Array("Luni", "Marti",
"Miercuri", "Joi", "Vineri", "Simbata", "Duminica");;
    this.dayOfWeekSymbols = new Array("L", "M", "M", "J", "V", "S",
"D");
```

For all of you, example lovers check the below table:

Pattern	Example
yyyy.MM.dd HH:mm:ss z	2004.09.05 12:07:35
EE, MM d	Wed, Jul 4
h:mm A	12:08 PM

Auto suggest component

MyUI provides a very flexible and easy to use auto complete (suggest) component.
What is an auto suggest component? A free text input field associated with a drop down menu from where the user can choose possible options to fill in the input field. Usually the options are influenced and changing depending on what the user was already typed in.
How do you implement an auto suggest component? An auto suggest component is actually an INPUT tag augmented with a few special attributes. The attribute that actually transforms an INPUT tag into an auto suggest component is `autoSuggest`. The `autoSuggest` attribute can have the following values:

`list:[comma separated list of strings]`
: The comma separated list of strings contains the options for the auto suggest component. The drop down menu is populated with the ones that starts with what the user entered in the field.

`javascript:[custom code]`
: This value enables full customization of the options that appears in the drop down menu. The `custom code` is a JavaScript code which is executed when the user input is changed to obtain the new options in the drop down menu. The returned value from this code must be an array containing different objects, usually strings. The displayed value in the menu is the string or the `displayValue` attribute, if defined.

`url:[url]`
: This will retrieve (using GET method) the options from the given URL. The returned content from the URL must be a list of matching options, each on a separate line. The value of the field is passed as a parameter with the name `value`.

The drop down menu for the auto suggest component is not shown immediately after the user is typing in the field, but after a latency time. This amount of time can be controlled using the `autoSuggestLatency` attribute, which represents the number of milliseconds after which the drop down menu is shown (the value must be an integer). The default value for this is 500 milliseconds.
A minimum number of characters entered in the field after which the suggestions are displayed can be also customized through the `minLengthForSuggestion` attribute. If the attribute is not specified a default value of 2 is used.
The suggestions drop down menu can display only a limited amount of suggestions. This can be customized using `maxDisplayedSuggestions` attribute. If the attribute is not specified then a default value of 10 is used. If the suggestions exceeds this, three dots (...) are shown at the end of the suggestions list.

Examples

And now just to make things a little bit more clear a few examples:

```
<input   id="test1"   type="text"   autosuggest="list:abc,abd,acd,xy"
autoSuggestLatency="100"/>
```

The possible auto suggest options are `abc,abd,acd,xy`. So if the user will enter `ab`, the options displayed in the drop down menu will be `abc` and `abd`.

```
<input  id="test2"  type="text"  autosuggest="javascript:  return  new
Array('Always','the same');"/>
```

No matter what user will enter in the text field the options displayed in the drop down menu will be `Always` and `the same`.

```
<input  id="test3"  type="text"  autosuggest="javascript:  return  new
Array(this.value + 's', this.value + 'ing');"/>
```

The options displayed in the drop down menu will be the entered value suffixed with `s` and `ing`. If the user will enter `rat` the component will suggest `rats` and `rating`.

CSS customization

The layout for the drop down menu can be easily customized using the following CSS classes:

`.autosuggest.menu.popup`	the entire drop down menu with the autosuggest options
`.autosuggest.list`	the list containing the auto suggest options
`.autosuggest.suggestion`	an auto suggest option
`.autosuggest.suggestion.selected`	the currently selected auto suggest options

So if I want a blue background autosuggest menu with yellow option and the one selected in white and red background I will have a CSS like the one below:

```
.autosuggest.list {
    background-color: blue;
}

.autosuggest.suggestion {
    color: yellow;
}

.autosuggest.suggestion.selected {
    background-color: red;
    color:white;
}
```

Field validation

MyUI offers some features so you can validate the value of the fields in a form. All the validations are customized from the fields or form attributes. JavaScript coding is necessary only for custom validation or custom error reporting.

Validators

MyUI features a set of predefined validations. To enable a validation for a field you have to define an attribute with the same name.

`isNotBlank`

If the value is `true` then the field value cannot be blank. A string of spaces is considered a blank value. If you want to test for the length of the value you better use `minLength` or `maxLength`. If the value is false, then a blank value is always validated, ignoring the other validators. If the value is not specified, then the other validators are executed.

`minLength`

If the value is positive then the field value must be at least `minLength` long. A negative value of `minLength` will be ignored.

`maxLength`

If the value is positive then the field value must be at most `maxLength` long. A negative value of `maxLength` will be ignored. Please note that you will not be able to enter more characters in the field.

`isFloat`

If the attribute value is `true` or `yes` then the field value must be an real number. This field uses the `radix` attribute for parsing the value in the given radix. If `radix` attribute is not specified a default value of 10 is considered.

`isInteger`

If the attribute value is `true` or `yes` then the field value must be an integer number. If this attribute is specified, the `isFloat` attribute is ignored. This field uses the `radix` attribute for parsing the value in the given radix. If `radix` attribute is not specified a default value of 10 is considered.

`minValue`

The field value must be a number and it must be equal or bigger than `minValue`. Please note that you'll be able to enter only digits.

If this attribute is specified it overrides the `isInteger` and `isFloat` attributes according to its type.

This field uses the `radix` attribute for parsing the value in the given radix. If `radix` attribute is not specified a default value of 10 is considered.

`maxValue`

The field value must be a number and it must be equal or smaller than `maxValue`. Please note that you'll be able to enter only digits.

If this attribute is specified it overrides the `isInteger` and `isFloat` attributes according to its type.

This field uses the `radix` attribute for parsing the value in the given radix. If `radix` attribute is not specified a default value of 10 is considered.

`pattern`

The field value must match the regular expression. The syntax of the regular expression is the Javascript one.

`dateFormat`

The field value must be a date specified in the given format. For the date format syntax please see the Date format section.

`minSelected`

The field must be a `SELECT` and it has to have at least `minSelected` options selected.

`maxSelected`

The field must be a `SELECT` and it has to have at most `maxSelected` options selected.

`allowedChars`

The user input should contain only characters from the ones specified in this attribute value. Actually this is not a validation, instead it allows the user to enter only those characters. If the `minValue` or `maxValue` attribute is defined then the allowed characters are only the numerical ones from this attribute.

`forbiddenChars`

The user input should not contain characters like the ones specified in this attribute value. Actually this is not a validation, instead it doesn't allow the user to enter those characters.

Examples

Below you can see some examples:

- A field that accepts numeric values between 0 and 100 (could be a percentage).

```
<input name="percent" type="text" minValue="0" maxValue="100"/>
```

- A mandatory non-blank field that can have at most 20 characters in length.

```
<input name="name" type="text" isNotBlank="true" maxLength="20"/>
```

- Allows user to enter only radix 16 digits unsigned.

```
<input name="name" type="text" allowedChars="0123456789abcdef"/>
```

- It doesn't allow the user to enter spaces and other special characters (this could be useful when entering an email address.

```
<input name="name" type="text" forbiddenChars=" ()[]`~|"/>
```

Validation flow

When does the validation take place? When any of the below events occurs:

- the field value is changed
- the form is submitted
- the validation of a field can be called programmatically: `field.validate()`
- the validation of all the fields in a form can be called programmatically: `form.validate()`

The `field.validate()` and `form.validate()` methods returns a boolean value indicating if the field and all the fields in the form, respectively, contain valid values.

What exactly happens during a field validation? Below is the list of actions taken:

1. If the input field has `onbeforevalidate` attribute defined then the attribute value is executed as JavaScript code. The scope of this code is the field itself, so you can use inside `this` to refer to the field. This code returns if the validation will take place or not, so if this code returns `false`, no further actions will be taken. Any other value will be ignored.
2. If the form in which resides the input field has `onbeforevalidate` attribute defined then the attribute value is executed as Javascript code. The scope of this code is the form itself, so you can use inside `this` to refer to the form. This code returns if the validation will take place or not, if this code returns `false`, no further actions will be taken. Any other value will be ignored.
3. The actual validation takes place. Please keep in mind that the result of validation is the boolean value of true or an [array of] error messages. For how the error messages are generated please see the next section.
4. If the field attribute `errorClass` or the form attribute `errorClass` is defined, then the CSS class of the field is changed to the given attribute value. The CSS class will change back to the

initial value when the field will contain again a valid value. If both attributes are defined then the field attribute takes precedence.

5. The error message is displayed according to the `displayError` attribute value.

   `alert`
   An alert message containing the list of error messages is displayed.
   `div[:`*`divId`*`]`
   The error messages will be displayed as a list in the `DIV` with the specified id or if no id is specified the the default value is the field name followed by `ErrorDiv`. If the error div has an `errorClass` attribute defined then its CSS class name will be set with the attribute value. This practically allows to hide an error div when the field value is valid and display it in red when invalid.
   `javascript:`*`javascript code`*
   Instead of displaying the error message, the given JavaScript code will be executed and it will be entirely its responsability to display the error message. Inside the JavaScript code you can use `errorMessage` to refer to the error message.

What exactly happens during a form validation?

All the fields are validated in the order in which they appear in the form. If a field contains an invalid value, then the validation of all the other subsequent fields is made according to the value of the `continueOnError` attribute of the form. If the value is `false` or no, then the validation process is stopped, otherwise continues with all the other fields. If the attribute is not specified a default value of `true` is used.

Error messages

As said before the result of the field value validation is either the boolean value of true or an array of error messages. Further we will explain how these messages are generated. For every predefined validator an error message is generated if the value doesn't follow the constraint. These messages are predefined or it can be customized by using:

- field attribute with the name of the constraint followed by `ErrMsg`; e.g. `minCharsErrMsg` or `patternErrMsg`
- field attribute with the name `errMsg`
- form attribute with the name of the constraint followed by `ErrMsg`;
- form attribute with the name `errMsg`

If more than one of these attributes are defined, it takes precedence the first one in the above order.

Programatic validation

A form or a field can be validated programmatically. The following methods are available for fields and forms:

`field.validate()`	Invokes the validation for a field.	Returns if the field contains a valid value according to the validators.
`form.validate()`	Invokes the validation for a form.	Returns true if all the fields contain a valid value according to the validators.
`form.validateAndSubmit()`	Invokes the validation for a form and if it returns true, the form is submitted.	Returns true if all the fields contain a valid value according to the validators.

Complex examples

And here some much more complex examples:

```
<input name="dontvalidate" type="checkbox"/> Don't validate
<input       name="address"       type="text"       maxLength="100"
onbeforevalidate="return !this.form.dontvalidate;"/>
```

A field that is not validated if a checkbox is checked.

```
<style>
  .invalid {
    border: solid 1px red;
  }
</style>

<input name="max30" type="text" maxValue="30" errorClass="invalid"/>
```

If the field will not contain a number or if the number is bigger than 30 then around the field will be displayed a red box of 1 pixel.

```
<style>
  DIV.error {
    display: block;
    color: red;
  }

  DIV.invisible {
    display: none;
  }
</style>

<input name="email" type="text"
  pattern="[.0-9a-z]+@[0-9a-z]+.[.0-9a-z]+"
  patternErrMsg="Email should contain a valid email address."
  displayError="div:emailErrorDiv"/>

<div id="emailErrorDiv" class="invisible" errorClass="error"></div>
```

A field for validating an email using the pattern regexp constraint. The error message is customized using the `patternErrMsg` attribute and it will be displayed inside a div. Please note that attribute `displayError="div:emailErrorDiv"` value here has the same effect as `displayError="div"`, as the div id has the default value. The error div is not displayed if the field value is valid and it is displayed in red if the value is invalid.

Cloud (lightbox)

MyUI integrates a very easy way to implement a customizable **cloud** component, also known as lightbox. A cloud is like a modal window shown over the web page. The user cannot interact with the web page

anymore, but only with the cloud window, until the page is unclouded. There can be only one cloud at a given time.
Any HTML element can be shown as a cloud using the method `document.cloud`.

```
...
<div id=”cloud”>
This is a cloud
</div>
...
<a href=”javascript:document.cloud(‘cloud’)”>cloud me</a>
```

The method `document.cloud(element, center, interactive, scrollable)` accepts the following parameters:

`element`
: the element or element ID that will be the cloud; if none specified, the document is clouded but with no cloud window

`center`
: if true, the cloud is automatically centered in the window; optional, by default true

`interactive`
: if true, the document can be unclouded clicking outside the cloud or pressing `ESCAPE` key; optional, by default true

`scrollable`
: if true, the document can be scrolled while clouded; optional, by default false

To uncloud a web page, simply call `document.uncloud()`.
The example above will become

```
...
<div id=”cloud”>
This is a cloud.
<a href=”javascript:document.uncloud()”>close</a>
</div>
...
<a href=”javascript:document.cloud(‘cloud’)”>cloud me</a>
```

The last line is identical to

```
<a
href=”javascript:document.cloud(document.getElementById(‘cloud’))”>c
loud me</a>
```

There is also an easier way to do it, by specifying the `cloud` value to the `rel` attribute to A tag

```
<a href=”#cloud” rel=”cloud”>cloud it</a>
```

This way you can use even an image or another document that will be loaded in an `IFRAME`

```
<a href=”img/photo.jpg” rel=”cloud”>image cloud</a>
<a href=”sample.html” rel=”cloud”>web page cloud</a>
```

The HTML document is hidden by another window, initially invisible, that sits between the HTML document and the cloud, called the **sky**. The appearance of the sky can be customized using the `sky` CSS class.
The cloud box can be customized using the **cloud** CSS class.
And now to complete the example we have to add the following lines:

```
<style>
    .sky {
        background-color: #999999;
        opacity: 0.7;
    }

    #cloud, .cloud {
        background-color: #333333;
        border: solid 1px #67CFCF;
        display: none;
        height: 300px;
        width: 400px;
    }
</style>
```

Day selector component

Overview

The day selector component allows you to select a date or a set of dates in a graphical way. The user interface will display a calendar from which you can select a date. Below you have a screenshot of the interface and all the components.

1. Main area with cells containing month days
2. Controls area which contains buttons for navigating to the next/previous month/year, buttons for invoking popups to select a specific day or month
3. Week days header area
4. Week index header area

Integration

The day selector is displayed in two ways: directly embedded on the HTML page or in a popup. If displayed in a popup, the selector should have an invoker link associated with it.

The embedded day selector is displayed directly on a web page in a `DIV`. The `DIV` must have the CSS class `daySelector`.

The popup day selector is displayed in a popup (see Popups) and is ensuring some kind of “`Browse…`” functionality for input fields with date constraints. Associating the selector with the field is done through a link (A tag) which invokes the day selector and populates the field accordingly when a date is selected. The link must have the HREF attribute in the form “`#[id of the input field]`”. For that field you have to define a date format validator (see Date format and Field validation). Please be aware that the `onchange` event of the text field is fired every time you select a new date.

The div for the day selector supports the following custom attributes:

`dateFormat`
: the date format used to display the dates in the tooltips (see Date format for more information)

`date`
: the default displayed date; the value is given in the format specified by the `dateFormat` attribute; if not specified today is displayed

`dateSelectionModel`
: the date selection model used by the day selector component; the value should be specified as Javascript and the result of evaluating the code will be the selection model itself. If unsure, please don't specify this value.

The link that invokes a day selector supports the following custom attributes

`hideOnSelect`
: this attribute is available only for links and if it is `true` or `yes` the day selector popup is hidden right after selecting a date. By default a value of `true` is considered.

Example

A simple day selector in a webpage:

```
<div class="daySelector"></div>
```

A day selector with a date (March 5, 2005) selected by default and using `dd/MM/yyyy` as date format:

```
<div class="daySelector" dateFormat=”dd/MM/yyyy” date=”5/3/2005”>
</div>
```

A date field with a Browse like functionality implemented (the date format is the one used in the field and the date selected is the one specified in the field):

```
<input id="dateField" name="dateField" dateFormat="dd/MM/yyyy”/>
<a href=”#dateField”>
    <img src="browseDay.gif" border="0" />
</a>
```

CSS Customization

The look of the day selector component can be customized using the following CSS classes:

`daySelector`
: the entire day selector container

`daySelectorControls`
: the header containing the controls like buttons for going to the previous month or next year

`daySelectorControl`
: any of the cells header containing the controls like buttons for going to the previous month or next year

`dayCell`
: any of the cells containing a date

`dayOfWeekHeader`
: any of the cells containing the day of week header

`weekHeader`

any of the cells containing the week of year header

`week1, week2, week3, week4, week5, week6`
: a header cell containing the week index header or a cell containing the date in the specified day of week

`previousMonth, currentMonth, nextMonth`
: the date cells in the given month

`selected`
: a selected day cell

`mon, tue, wed, thu, fri, sat, sun`
: a header cell containing the day of week header or a cell containing the date in the specified day of week

`today`
: the cell containing the today's date

These CSS classes can be combined. So for example, if you want the today's date to be green if unselected and blue if selected you'll do like this:

```
.today {
 color: green;
}

.today.selected {
 color: blue;
}
```

The customization of the day selector component is limited only by your imagination.

Effects

MyUI comes equipped with a set of ready to use visual effects. An effect is basically a set of actions that happens at a specific interval. The actions refer to the alteration of visual properties of a specific HTML element. For example, if you modify the x or y coordinate of an element it will appear like the element is moving. Or if you increase the transparency of an element, it will appear like the element is fading out.
Using an effect has two steps: creating it and starting it. An already started effect cannot be started again, until it is finished. An effect can be stopped at any time using the `cancel` method.
More likely to obtain these kind of effects you will use `PropertyEffect` which alters the properties of an HTML element.
To create a property effect use the constructor

```
PropertyEffect(elem, final, totalTime, speed, acceleration)
```

where

`elem`
: the element (or its HTML ID) on which the effect will be applied

`final`
: an object containing a list of final values for the element properties. The recognized element properties are:
  - `width` – the element width, integer value, expressed in pixels
  - `height` – the element height, integer value, expressed in pixels
  - `x` – the element X coordinate relative to the top-left corner, integer value, expressed in pixels

- y – the element Y coordinate relative to the top-left corner, integer value, expressed in pixels
- `opacity` – the element opacity (as opposed to transparency), float value between 1 (completely opaque) and 0 (completely transparent)
- `scrollLeft` – where the scrolling in an element begins relative to the top-left corner, integer value, expressed in pixels
- `scrollTop` – where the scrolling in an element begins relative to the top-left corner, integer value, expressed in pixels
- `clipTop, clipBottom, clipRight, clipLeft` – the clipping rectangle of the element, integer values, expressed in pixels relative to the top-left corner of the element
- `marginTop, marginBottom, marginRight, marginLeft` – element margins, integer values, expressed in pixels

Except the integer/float values given for properties, you can also specify values in the following formats:

- [-][percent]% - increases/decreases the current value with the given percentage
- +[-][amount] – increases/decreases the current value with the given amount

`totalTime`

how much time the effect should run

`speed`

the speed of the effect in fps (frames per seconds) Higher values can result in smoother animation, but requires more performance

`acceleration`

a value or an array of values on how the changes on the values of the element properties are distributed over time.

Let’s clarify things a little bit with a few examples:

- The element with the box HTML ID will move to the y-coordinate equal to 500.

```
new PropertyEffect(document.getElementById("box"), {y: 500})
```

- The box will fade out almost entirely in 2 seconds. The animation speed is increased to 30 fps to be much smoother.

```
new PropertyEffect(document.getElementById("box"), {opacity: 0.1},
2000, 30)
```

- The box will move accelerated with 500 pixels on the right in 500 milliseconds.

```
new PropertyEffect(document.getElementById("box”), {x: +500}, 500,
30, 1.1)
```

- The box will move decelerated with 10% from the current position to the left.

```
new PropertyEffect(document.getElementById("box”), {x: -10%}, 500,
30, -0.7)
```

- The box will move 200 pixels from the current position to the left. In the first 600 ms the acceleration will be 2.1, then in the next second the object will move with no acceleration and in the last 400 ms the object will move decelerated.

```
new PropertyEffect(document.getElementById("box”), {y: +200}, 2000,
30, [{acceleration: 2.1, time: 0.3}, {acceleration: 0, time: 50%},
{acceleration: -0.9, time: 0.2}])
```

More effects can be combined into one effect using `SerialEffect` (effects will be executed one after another), `ParallelEffect` (effects will be executed in parallel) or `RandomEffect`(one effect is chosen randomly and executed).
To create such an effect you have to pass to the constructor effects or arrays holding effects objects.

```
var serialEffect = new SerialEffect(
    new  PropertyEffect(document.getElementById("box"),  {x:  "100%",
y:500, opacity: "+-0.6", width: 500, clipLeft: 30}),
    new  PropertyEffect(document.getElementById("box"),  {x:  "-50%",
y:300, opacity: "+0.6", width: 400, clipLeft: 10})
)

var parallelEffect = new ParallelEffect(new Array(
    new  PropertyEffect(document.getElementById("box"),  {x:  "100%",
y:500, opacity: "+-0.6", width: 500, clipLeft: 30}),
    new  PropertyEffect(document.getElementById("box"),  {x:  "-50%",
y:300, opacity: "+0.6", width: 400, clipLeft: 10})
));
```

Frames

Frames/windows could be created using the constructor `new Frame(doc, x, y, width, height)`, where the attributes represents the following

`doc`
    the owner document
`x`
    the horizontal axis coordinate
`y`
    the vertical axis coordinate
`width`
    the window width
`height`
    the window height

The created element supports the following methods and properties:

Name	Method	Parameters
getX()	returns the horizontal coordinate	
setX(x)	sets the horizontal coordinate	
getY(y)	returns the vertical coordinate	
setY(y)	sets the vertical coordinate	
setPosition(x, y)	sets the window position on horizontal and vertical axis; if x and y are null, then the frame is centered on screen	
getWidth()	returns the window width	

setWidth(width)	sets the window width	
getHeight()	returns the window height	
setHeight(height)	sets the window height	
setSize(width, height)	sets the window size	
scrollToVisible()	scroll the document so that this window could become visible; if the window is not visible, the document will be scrolled, but the window will still not be visible	
isIconified()	returns if this window is iconified	
setIconified(iconified)	Sets the iconified state of this window	
isVisible()	returns if this window is visible	
setVisible(visible)	if the argument is true, makes this window visible (if not already); hides it otherwise	
setTitle(elem)	sets the title to appear on the window title bar	elem is an HTML element or a string containing text or HTML content
setStatus(elem)	sets the status to appear on the window status bar	elem is an HTML element or a string containing text or HTML content
setContent(elem)	sets the window content	elem is an HTML element or a string containing text or HTML content
addContent(elem)	adds the parameter to the window as content	elem is an HTML element or a string containing text or HTML content
resizable = *true/false*	get/set if the window can be resized by the user	
positionLocked = *true/false*	get/set if the window can be moved by the user or not	*true* - the window cannot be moved by the user, *false* - the window can be moved by the user

You can easily add/remove event listeners to every frame using addEventListener/removeEventListener, as the well known standard methods. The following events are recognized:

- `framemove` – fired when the position of the frame is changed
- `frameresize` – fired when the width or height of the frame is changed
- `frameiconify` – fired when the frame is changed to an iconified status
- `framedeiconify` – fired when the frame is changed to an deiconified status
- `frameshow` – fired when the position of the frame is shown
- `framehide` – fired when the position of the frame is hidden

Frame actions like iconifying use some sort of visual effects. This can be disabled with

```
Frame.useEffects = false;
```

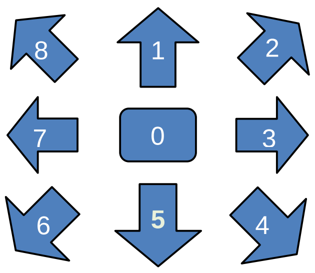
and re-enabled with

```
Frame.useEffects = true;
```

Gestures

A gesture is a mouse movement executed while holding the left button down. This movement resembles some kind of shape.

How would you use it? First you will define a set of gestures (you can use the gesture editor packed with the demos). Then you will add an event listener to an object. Whenever a gesture is started from that object the listener will be executed. In the listener handler you have the opportunity to compare the gesture with your predefined set of gesture. If a match is found, you can start whatever action, depending on the match.



A gesture has a name (optional) and a shape, encoded as a set of directions, as in the figure in the right. A gesture must match not only the shape, but also the direction of the movement.

Taking this into account to create a gesture is fairly easy:

```
  new Gesture(“Up”, “111”);
```

To add a gesture listener is done like with any other event:

```
element.addEventListener(“gesture”, function() {
 // display the gesture executed by the user
 // drawing a gesture works only in Mozilla and Safari
 event.gesture.displayInWindow(window);
}, false);
```

And to match a gesture again a set of predefined ones you need to use the `mostLikely` method. Putting all these together, we will have:

```
var gestures = new Array();
gestures.push(new Gesture("Up","111"));
gestures.push(new Gesture("Right","333"));
gestures.push(new Gesture("Down","555"));
gestures.push(new Gesture("Left","777"));
element.addEventListener("gesture", function(event) {
 event.gesture.displayInWindow(window);
 var gestureMatch = event.gesture.mostLikely(gestures);
 if (gestureMatch == null) {
       // no gesture is matched
       return;
 }
 switch(gestureMatch.name) {
       case "Up"
             // do something on gesture Up
             break;
       case "Right"
             // do something on gesture Right
             break;
       case "Down"
             // do something on gesture Down
             break;
       case "Left"
             // do something on gesture Left
```

```
            break;
 }
}, false);
```

Pop-ups

Pop-ups are small window that appear usually as a consequence of a contextual right click. Such a popup can be created using the constructor `new Popup(doc)`, where `doc` is the document that will contain the popup.

Pop-up methods

`show(x, y)`	Shows the popup at the given coordinates relative to the top-left corner of the document.
`hide()`	Hides the popup.
`isVisible()`	Returns true if the popup is currently visible, false otherwise
`setContent(elem)`	Sets the content of the popup. The parameter is either an HTML element or a string containing HTML code

If you want to setup any popup as the contextual popup of an HTML element, document or window you can use the method

```
element.setRightClickPopup(popup)
```

The pop-ups appearance can be easily customized using the `popup` CSS class name. E.g. if you want pop-ups with a gray background, you can use

```
.popup {
    background-color: #555555;
}
```

Progress bar / slider

To create a progress bar/slider using MyUI is very easy. Just mark an element with the `progressbar/slider` CSS class and that's it. The only difference between a progress bar and a slider is that the slider has an element to mark the current position. The slider can be used as a scrollbar too.
Every progress bar has a **percent** and a **value** associated with it. The percent is the **actual completed percentage** and the value is the **actual (completed) value**. The percent is an integer number between 0 and 100. The value is a float between `minValue` and `maxValue`, where `minValue` and `maxValue` are the values of the DIV attributes with the same name. If these are missing the default are between 0 and 100. The default value of the progress bar is the integer number enclosed in the DIV, by default 0. A progress bar can be also modified by user interaction, by adding the `editable` CSS class to the DIV. The user can click on the progress bar or drag it and thus will change its value accordingly. This ensures slider/scroll bar functionality.
As for any MyUI components, the progress bar can be easily customized through CSS classes:
`completed`

for the portion of the progress bar that is completed. E.g. setting a blue background for the completed part of a progress bar can be easily done with

```
.completed {
    background-color: blue;
}
```

For a slider, it would make sense to completely hide this part.

```
.completed {
    display: none;
}
```

`label`
: for the text displayed over the progress bar, at the center.

`face`
: this is available only for sliders and represents the element to mark the current position.

```
.face {
    background-color: gray;
    border: solid 1px lightgrey;
}
```

By default a progress bar is horizontal, but it can be made vertical if you add the CSS class `vertical`. If you customize it using background images, remember that these are not automatically rotated. So you have to have a different version for the vertical orientation.

```
.completed {
    background: blue url(bluegradient.gif) repeat-x;
}
.vertical .completed {
    background: blue url(bluegradient-vertical.gif) repeat-y;
}
```

The label displayed on the progress bar can be easily customized using the `label` attribute which can have one of the following values:

- `none` – no label is displayed
- `value` – the completed value of the progress bar is displayed
- `percent` – the completed percent of the progress bar is displayed
- `text:[text]` – the given text is displayed, regardless of the completed value of the progress bar.
- `javascript:[expression]` – the given expression is evaluated every time the completed percent is changed and the result is displayed.You can refer to the progress bar inside the expression using `this`. The following will display both the completed value and percent

```
<div class="progressbar" label="javascript: this.getValue() + ' (' +
this.getPercent() + '%)'">30</div>
```

Scroll pane

A

Slide show

Using MyUI you can take advantage of a full featured slideshow. The easiest way to include such a slideshow in your web page is by marking a DIV with the slideshow CSS class.

```
<div class=”slideshow”>
    <div>Slide 1</div>
    <div>Slide 2</div>
</div>
```

This will not do you too much good. You have the slideshow, but at this moment it will seem that it is doing nothing. So next we have to configure it.

Configuration parameters

The slideshow configuration is made through HTML attributes on the slideshow DIV. You also need to specify a width and height for it. Please find below a list, with the name, possible values and description.

Name	Description	Possible values
autostart	If specified, this should be a numerical value representing the number of milliseconds after which the slideshow will automatically start	by default, the slideshow will not automatically start
circular	If true, if the slideshow reaches the final slide will restart with the first one.	true/false, by default true
orientation	The orientation of the slideshow and how the slides are aligned	horizontal/vertical, by default horizontal
slideInterval	If this slideshow is automatically running this the pause interval between slides. This interval does not include the transition time.	Numeric, by default 3000
source	The slideshow source. Please see the next paragraph.	default is new `ChildrenSlideShowSource(this)`
transition	The transition between slides type.	slide/fade/none/back-forth/back-forth-[angle], by default slide
transitionAcceleration	The transition between slides acceleration.	Numeric, by default 50
transitionSpeed	The transition between slides speed in fps (frames per seconds)	Numeric, by default 20
transitionTime	The transition between slides total time in milliseconds	Numeric, by default 1000
updateAddress	If false, the address bar is not automatically updated with the slide permalink	true/false, by default true

For transition, `transitionAcceleration`, `transitionSpeed` and `transitionTime` you better consult the Effects, as this is heavily based on.

Slideshow links

In the document you can have links to different slides in a slideshow or links to move to the previous or next slide.

These links are special and their reference (HREF attribute) should be one of the following:

- `#[slideShowId]#[slideIndex]` – move to the slide with the given index(0 for the first slide) in the slideshow with the given HTML ID
- `#[slideShowId]#[slideId]` – move to the slide with the given HTML ID in the slideshow with the given HTML ID
- `#[slideShowId]#next` - move to the next slide in the slideshow with the given HTML ID
- `#[slideShowId]#previous` - move to the previous slide in the slideshow with the given HTML ID

where `slideShowId` is the HTML ID of the slideshow element, `slideId` is the HTML ID of the slide element and `slideIndex` is the zero-based index of the slide.
Below you can find some examples:

```
<a href="#slideshow1#0"> first slide</a>

<a
href="javascript:document.getElementById('slideshow1').setCurrent(1)
;"> second slide</a>

<a href="#slideshow1#1">second slide</a>

<a href="#slide0">first slide</a>

<a href="#slideshow1#previous">previous slide</a>

<a href="#slideshow1#next">next slide</a>
```

Slideshow source

The slideshow can retrieve its slides from all kind of different sources. A slideshow source is practically a class that implements two methods: size() which returns the number of slides in the presentation and get(index) which returns an HTML element representing the slide with the given index.
This is configured using the source attribute of the slideshow element, of which string value is evaluated as a JavaScript expression.
You can create your own source or use an existing one already provided in the distribution.

- `ChildrenSlideShowSource` – the slides are the DIV children of the element passed in the constructor. The constructor also accepts the ID of the parent element instead of the element itself. Example:

```
new ChildrenSlideShowSource(document.getElementById(“slideshow”))
```

- `ElementsSlideShowSource` – the slides are passed in the constructor as elements or HTML ID. Example:

```
new ElementsSlideShowSource(“s1”, document.getElementById(“s2”));
```

Programatic usage

A slideshow comes equipped with some JavaScript methods for your convenience.
`next()` – advance to the next slide.
`previous()` – back to the previous slide.
`getCurrent()` – returns the current slide index.

`setCurrent(index)` – go to the slide with the given index. The parameter represents the zero-based index of the slide.
`setCurrentSlide(slideId)` – go to the slide with the given HTML ID. The parameter represents the HTML id of the slide element.
start(after) – will automatically start and advance slides after the specified number of milliseconds.
stop() – will automatically stop the auto slideshow.

Example

Below is a more complex example.

```
<html>
<head>
    <script src="../src/myui.js"></script>
    <style>
        A {
            outline: 0;
        }

        #slide0 {
            background-color: navy;
        }
        #slide1 {
            background-color: lightgreen;
        }

        #slide2 {
            background-color: lightblue;
        }
        #slide3 {
            background-color: forestgreen;
        }

        #s1 {
            width: 500px;
            height: 350px;
            border: solid 1px #222222;
        }

        .slideshow {
            width: 500px;
            height: 350px;
            float: left;
            background-color: lightyellow;
        }

        .previousSlide, .nextSlide {
            display: block;
            float: left;
            width: 50px;
            height: 100%;
        }

        .nextSlide {
            margin-left: -50px;
        }
        .previousSlide {
            margin-left: -100%;
        }
    </style>
</head>
<body>
    <a
href="javascript:document.getElementById('sshow1').setCurrent(0);">1
</a>
    <a href="#sshow1#1">2</a>
    <a
href="javascript:document.getElementById('sshow1').setCurrent(2);">3
</a>
    <a href="#sshow1#slide3">4</a>

    <br />

    <div id="s1">
        <div id="sshow1" class="slideshow" transition="slide">
            <div id="slide0">1</div>
            <div id="slide1">2</div>
            <div id="slide2">3</div>
            <div id="slide3">4</div>
        </div>
        <a href="#sshow1#previous" class="previousSlide">&laquo;</a>
        <a
href="javascript:document.getElementById('sshow1').next()"
class="nextSlide">&raquo;</a>
    </div>

</body>
</html>
```

This will create a slideshow with 4 slides. We also attached previous and next buttons over the slides, at the left and right of the slideshow. These can be tuned up to your imagination, adding images, transparency, hovering effects. Using the links at the top of the page you can go directly to a specific slide.

Table-tree

The table-tree component is a combination of table and tree. It is a tree of which items are rows in a table and the information concerning the tree items is organized into columns. Creating such a component is very straightforward. Any HTML table can be easily enhanced with the tree functionality, by adding just a few more attributes.

First of all, the table. For the table you must define the following attributes:

`levelIndent`
: The indentation for each level in the tree in pixels. Every row in the table will appear indented according to its level multiplied by this value. Mandatory.

`iconExpanded`
: The URL to the image representing the icon that appears in front of an expanded node.

`iconCollapsed`
: The URL to the image representing the icon that appears in front of an collapsed node.

Now that we have the table tree in place, we have to define its hierarchy. Every row must contain a cell (`TD` or `TH`) with just one attribute defined: `treeLevel`. This is an integer and represents the level in the tree of the containing row.

As a hierarchy, all the rows that follows a row and have a smaller level are in the same subtree.

Finally, the example. The following HTML code:

```
<table          levelIndent="10"          iconExpanded="expanded.gif"
iconCollapsed="collapsed.gif">
    <tr> <td treeLevel="0">1</td> <td>MyUI</td> </tr>
    <tr> <td treeLevel="1">1.1</td> <td>is</td> </tr>
    <tr> <td treeLevel="2">1.1.1</td> <td>the</td> </tr>
    <tr> <td treeLevel="2">1.1.2</td> <td>best</td> </tr>
    <tr> <td treeLevel="1">1.2</td> <td>web</td> </tr>
    <tr> <td treeLevel="2">1.2.1</td> <td>framework</td> </tr>
    <tr> <td treeLevel="1">1.3</td> <td>!!!</td> </tr>
</table>
```

will render as

```
 - 1               MyUI
     - 1.1         is
         - 1.1.1   the
         - 1.1.2   best
     - 1.2         web
         - 1.2.1   framework
     - 1.3         !!!
```

URL parameters

If you specify the attribute `copyValuesFromParameters` to a form, then all the fields will be prefilled with the parameters in the URL. Please keep in mind that these are only the GET parameters, the POST ones will be ignored.

The field value will be copied from the parameter with the same name as the field name. This will be applied for text fields, text areas, checkboxes, radio buttons, password fields, hidden fields, drop-down and multiple select components.

If we want to match the fields with parameters in other way than with the name, we have to specify the attribute `parameterMatchingAttribute` on the form. Then every field will be populated with the value of the parameter of which names is equal to the value of the field's attribute with the name specified by the form attribute `parameterMatchingAttribute`.

Example

1. If we have the URL `http://a.site.com/prefill.html?name=User%20Interface&public=yes&color=green&size=medium` and the html page `prefill.html`

```
<html>
<head>
    <script src="myui.js"></script>
</head>
<body>
<form method="get" prefillFromParameters="true">
<table>
<tr><td>Name:</td><td><input name="name" type="text"/></td></tr>
<tr><td>Public:</td><td><input                          name="public"
type="checkbox"/></td></tr>
<tr><td>Color:</td><td>
<input name="color" type="radio" value="red"/> red<br>
<input name="color" type="radio" value="green"/> green<br>
<input name="color" type="radio" value="blue"/> blue<br>
</td></tr>
<tr><td>Size:</td><td>
<select name="size">
<option value="small">small</option>
<option value="medium">medium</option>
<option value="large">large</option>
</td></tr>
<tr>
    <td colspan="2" align="right">
        <input  type="submit"  id="submitButton"  name="submitButton"
value="submit"/>
    </td>
</tr>
</table>
</form>
</body>
</html>
```

then the first field will be populated with `User Interface` value, the checkbox will be checked, the second radio button will be selected and the `medium` option will be selected in the dropdown.

2. If we have the URL `http://a.site.com/prefill.html?firstName=John&lastName=Doe` and the html page `prefill.html`

```
<html>
<head>
    <script src="myui.js"></script>
</head>
<body>
<form           method="get"           prefillFromParameters="true"
parametersMatchingAttribute="id">
Name:<input name="firstName" id="lastName" type="text"/>
</form>
```

```
</body>
</html>
```

then the field will be filled in with the `Doe` value.

Contact

If you have any questions or suggestions about MyUI please feel free to use the dedicated forum at http://p.sf.net/my-ui/forum.