SCRIPT.ACULO.US - QUICK GUIDE

SCRIPT.ACULO.US - OVERVIEW

What is script.aculo.us?

script.aculo.us is a JavaScript library built on top of the *Prototype JavaScript Framework*, enhancing the GUI and giving Web 2.0 experience to the web users.

script.aculo.us was developed by Thomas Fuchs and it was first released to the public in June 2005.

script.aculo.us provides dynamic visual effects and user interface elements via the Document Object Model (DOM).

The Prototype JavaScript Framework is a JavaScript framework created by Sam Stephenson that provides an Ajax framework and other utilities.

How to Install script.aculo.us?

It is quite simple to install the script.aculo.us library. It can be set up in three simple steps −

- Go to the download page to download the latest version in a convenient package.
- Unpack the downloaded package and you will find the following folders −
  - **lib** − contains prototype.js file.
  - **src** − contains the following 8 files −
    - builder.js
    - controls.js
    - dragdrop.js
    - effects.js
    - scriptaculous.js
    - slider.js
    - sound.js
    - unittest.js
  - **test** − contains files for testing purpose.
  - **CHANGELOG** − File that contains the history of all the changes.
  - **MIT-LICENSE** − File describing the licensing terms.
  - **README** − File describing the installation package. including the installation instructions.
- Now put the following files in a directory of your website, e.g. /javascript.
  - builder.js
  - controls.js
  - dragdrop.js
  - effects.js
  - scriptaculous.js
  - slider.js

    - prototype.js

NOTE − The sound.js and unittest.js files are optional

How to Use script.aculo.us Library?

Now you can include *script.aculo.us* script as follows −

```
<html>
   <head>
      <title>script.aculo.us examples</title>
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script >
   </head>

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

By default, scriptaculous.js loads all of the other JavaScript files necessary for effects, drag-and-drop, sliders, and all the other script.aculo.us features.

If you don't need all the features, you can limit the additional scripts that get loaded by specifying them in a comma-separated list, e.g. −

```
<html>
   <head>
      <title>script.aculo.us examples</title>
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js?
load=effects,dragdrop"></script>
   </head>

   <body>
      ........
   </body>

</html>
```

The scripts that can be specified are −

- effects
- dragdrop
- builder
- controls
- slider

NOTE − Some of the scripts require that others be loaded in order to function properly.

How to Call a script.aculo.us Library Function?

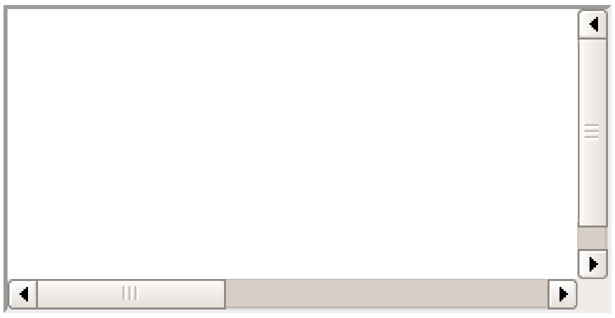
To call a script.aculo.us library function, use HTML script tags as shown below −

```
<html>
   <head>
      <title>script.aculo.us examples</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,dragdrop"></script>

      <script type="text/javascript" language="javascript">
```

```
         // <![CDATA[
         function action(element){
            new Effect.Highlight(element,
               { startcolor: "#ff0000", endcolor: "#0000ff", restorecolor: "#00ff00",
duration: 8 });
         }
         // ]]>
      </script>

   </head>

   <body>

      <div >
         Click on this and see how it change its color.
      </div>

   </body>

</html>
```

Here we are using the Effect module and we are applying *Highlight* effect on an element.

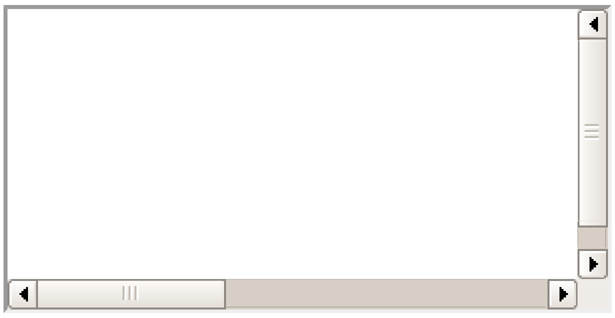
This will produce following result −



Another easy way to call any module's function is inside event handlers as follows −

```
<html>
   <head>
      <title>script.aculo.us examples</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript"   src="/javascript/scriptaculous.js?
load=effects,dragdrop"></script>

   </head>

   <body>

      <div onclick="new Effect.BlindUp(this, {duration: 5})">
         Click here if you want this to go slooooow.
      </div>

   </body>

</html>
```

This will produce following result −



SCRIPT.ACULO.US - MODULES

script.aculo.us is divided into modules, each with its own JavaScript file. These modules are explained here −

Effects

The effects module comes with more than twenty-five visual effects and seven transition modes.

Drag and Drop

You will use the drag and drop module to make any element *draggable*, turn it into a drop zone, or even make an entire series of elements sortable so that you can rearrange them by dragging and dropping.

Sliders

A slider is a sort of small rail or track, along which you can slide a handle. It translates into a numerical value. With script.aculo.us, you can create such sliders with a lot of control.

Autocompletion

Autocompleter controls allow Google-Suggest style, local and server-powered autocompleting text input fields.

In-place Editing

You can make any text or collection of items editable in-place by simply clicking it.

Builder

A helper to build DOM fragments in JavaScript. This is a developer tool that eases DOM creation considerably.

Sound

Version 1.7.1 introduced a sound system that lets you play sounds easily, queue them up, use multiple tracks, and so on.

SCRIPT.ACULO.US - VISUAL EFFECTS

The script.aculo.us effects are divided into two groups −

Core Effects

The following six core effects are the foundation of the script.aculo.us Visual Effects Java Script library.

- Effect.Opacity
- Effect.Scale
- Effect.Morph

- Effect.Move
- Effect.Highlight
- Effect.Parallel

All the core effects support various common parameters as well as effect-specific parameters and these effect names are case-sensitive.

All the effect-specific Common Parameters have been discussed in this tutorial along with the effects.

Combination Effects

All the combination effects are based on the five Core Effects, and are thought of as examples to allow you to write your own effects.

Usually these effects rely on the parallel, synchronized execution of other effects. Such an execution is readily available, hence creating your own combined effects is very easy. Here is a list of Combination Effects −

- Effect.Appear
- Effect.Fade
- Effect.Puff
- Effect.DropOut
- Effect.Shake
- Effect.SwitchOff
- Effect.BlindDown
- Effect.BlindUp
- Effect.SlideDown
- Effect.SlideUp
- Effect.Pulsate
- Effect.Squish
- Effect.Fold
- Effect.Grow
- Effect.Shrink

Additionally, there's the **Effect.toggle** utility method for elements you want to show temporarily with an Appear/Fade, Slide or Blind animation.

- Effect.toggle

Library Files Required for Effects

To use the effects capabilities of script.aculo.us, you will need to load the effects module. So, your minimum loading for script.aculo.us will look like this:

```
<html>
   <head>
      <title>script.aculo.us effects</title>
      <script type="text/javascript"  src="/javascript/prototype.js"></script>
      <script type="text/javascript"  src="/javascript/"effects.j"></script>
   </head>
   <body>
      ...
   </body>
</html>
```

Syntax to Call Effect Functions

The proper way to start a core effect is usually with the **new** operator. Depending on your preferences, you can use one of two syntaxes −

```
new Effect.EffectName(element [, requiredArgs ] [ , options ] )

OR

element.visualEffect('EffectName' [, requiredArgs ] [,options])
```

These two syntaxes are technically equivalent. Choosing between the two is mostly about your personal sense of code aesthetics.

Example

Here are two equivalent calls, so you can see how the syntaxes are related, which are very much interchangeable −

```
new Effect.Scale('title', 200, { scaleY: false, scaleContent: false });

OR

$('title' ).visualEffect('Scale', 200, { scaleY:false, scaleContent:false });
```

SCRIPT.ACULO.US - DRAG & DROP

The most popular feature of Web 2.0 interface is the drag and drop facility. Fortunately script.aculo.us comes with an inherent capability to support drag and drop.

To use the dragging capabilities of script.aculo.us, you'll need to load the **dragdrop** module, which also requires the **effects** module. So your minimum loading for script.aculo.us will look like this:

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
<script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,dragdrop"></script>
```

Dragging Things Around

It is very simple to make an item draggable using script.aculo.us. It requires creating of an instance of the *Draggable* class, and identifying the element to be made draggable.

Draggable Syntax

```
new Draggable( element, options );
```

The first parameter to the constructor identifies the element to be made draggable either as the *id* of the element, or a reference to the element. The second parameter specifies optional information on how the draggable element is to behave.

Draggable Options

You can use one or more of the following options while creating your draggable object.

Option	Description	Examples
revert	If set to *true*, the element returns to its original position when the drag ends. Also specifies whether the *reverteffect* callback will be	Example

	invoked when the drag operation stops. Defaults to *false*.	
snap	Used to cause a draggable to snap to a grid or to constrain its movement. If false (default), no snapping or constraining occurs. • If it is assigned an integer x, the draggable will snap to a grid of x pixels. • If an array [x, y], the horizontal dragging will snap to a grid of x pixels and the vertical will snap to y pixels. • It can also be a function conforming to *Function*(x , y , draggable) that returns an array [x, y].	Example
zindex	Specifies the CSS z-index to be applied to the element during a drag operation. By default, the element's z-index is set to 1000 while dragging.	Example
ghosting	Boolean determining whether the draggable should be cloned for dragging, leaving the original in place until the clone is dropped. Defaults to *false*.	Example
constraint	A string used to limit the draggable directions, either *horizontal* or *vertical*. Defaults to *null* which means free movement.	Example
handle	Specifies an element to be used as the handle to start the drag operation. By default, an element is its own handle.	Example
starteffect	An effect called on element when dragging starts. By default, it changes the element's opacity to 0.2 in 0.2 seconds.	Example
reverteffect	An effect called on element when the drag is reverted. Defaults to a smooth slide to element's original position.Called only if *revert* is true.	Example
endeffect	An effect called on element when dragging ends. By default, it changes the element's opacity to 1.0 in 0.2 seconds.	Example

Callback Options

Additionally, you can use any of the following callback functions in the options parameter :

Function	Description	Examples
onStart	Called when a drag is initiated.	Example
onDrag	Called repeatedly when a mouse moves, if mouse position changes from previous call.	Example
change	Called just as onDrag (which is the preferred callback).	Example
onEnd	Called when a drag is ended.	Example

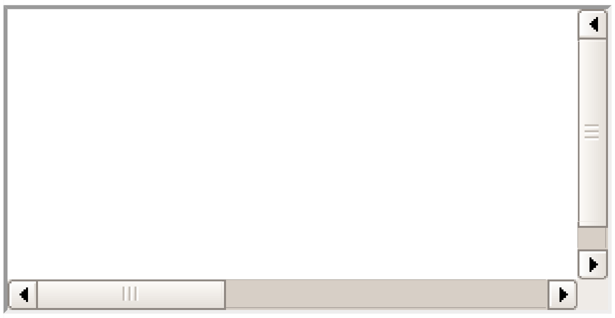
Except for the "change" callback, each of these callbacks accepts two parameters: the Draggable object, and the mouse event object.

Draggable Example

Here, we define 5 elements that are made draggable: three <div> elements, an <img> element, and a <span> element. The purpose of the three different <div> elements is to demonstrate that regardless of whether an element starts off with a positioning rule of static (the default), relative, or absolute, the drag behavior is unaffected.

```
<html>
   <head>
      <title>Draggables Elements</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>
      <script type="text/javascript">

         // Take all the elements whatever you want to make Draggable.
         var elements = ['normaldiv', 'relativediv', 'absolutediv', 'image', 'span'];

         // Make all the items drag able by creating Draggable objects
         window.onload = function() {
            elements.each(function(item) { new Draggable(item, {});});
         }

      </script>
   </head>
   <body>

      <div >
         This is a normal div and this is dragable.
      </div>

      <div >
         This is a relative div and this is dragable.
      </div>

      <div >
         This is an absolute div and this dragable.
      </div>
      <br />

      <img />

      <p>Let part <span >
         This is middle part</span> Yes, only middle part is dragable.</p>

   </body>
</html>
```

This will produce following result –



Dropping Dragged Things

An element is converted into a drop target via a call to the *add()* method within a namespace called *Droppables*.

The Droppables namespace has two important methods: *add()* to create a drop target, and *remove()* to remove a drop target.

Syntax

Here is the syntax of the add() method to create a drop target. The add() method creates a drop target out of the element passed as its first parameter, using the options in the hash passed as the second.

```
Droppables.add( element, options );
```

The syntax for remove() is even more simpler. The remove() method removes the drop target behavior from the passed element.

```
Droppables.remove(element);
```

Options

You can use one or more of the following options while creating your draggable object.

Option	Description	Examples
Hoverclass	The name of a CSS class that will be added to the element while the droppable is active (has an acceptable draggable hovering over it). Defaults to null.	Example
Accept	A string or an array of strings describing CSS classes. The droppable will only accept draggables that have one or more of these CSS classes.	Example
Containment	Specifies an element, or array of elements, that must be a parent of a draggable item in order for it to be accepted by the drop target. By default, no containment constraints are applied.	Example
Overlap	If set to 'horizontal' or 'vertical', the droppable will only react to a Draggable if its overlapping by more than 50% in the given direction. Used by Sortables, discussed in the next chapter.	
greedy	If true (default), it stops hovering other droppables, under the draggable won't be searched.	Example

Callback Options

Additionally, you can use any of the following callback functions in the options parameter :

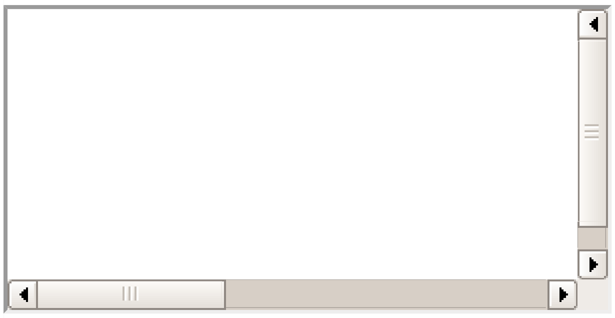
Function	Description	Examples
onHover	Specifies a callback function that is activated when a suitable draggable item hovers over the drop target. Used by Sortables, discussed in the next chapter.	
onDrop	Specifies a callback function that is called when a suitable draggable element is dropped onto the drop target.	Example

Example

Here, the first part of this example is similar to our previous example, except that we have used Prototype's handy $A() function to convert a node list of all the <img> elements in the element with the id of draggables to an array.

```
<html>
   <head>
      <title>Drag and Drop Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>

      <script type="text/javascript">

         window.onload = function() {

            // Make all the images draggables from draggables division.

            $A($('draggables').getElementsByTagName('img')).each(function(item) {
               new Draggable(item, {revert: true, ghosting: true});
            });

            Droppables.add('droparea', {hoverclass: 'hoverActive', onDrop: moveItem});

            // Set drop area by default  non cleared.
            $('droparea').cleared = false;
         }

         // The target drop area contains a snippet of instructional
         // text that we want to remove when the first item
         // is dropped into it.

         function moveItem( draggable,droparea){

            if (!droparea.cleared) {
               droparea.innerHTML = '';
               droparea.cleared = true;
            }

            draggable.parentNode.removeChild(draggable);
            droparea.appendChild(draggable);
         }
      </script>

      <style type="text/css">

         #draggables {
            width: 172px;
            border: 3px ridge blue;
            float: left;
            padding: 9px;
         }

         #droparea {
            float: left;
            margin-left: 16px;
            width: 172px;
            border: 3px ridge maroon;
            text-align: center;
            font-size: 24px;
            padding: 9px;
            float: left;
         }

         .hoverActive {
            background-color: #ffffcc;
         }

         #draggables img, #droparea img {
            margin: 4px;
            border:1px solid red;
         }

      </style>
   </head>
   <body>

      <div >
         <img src="/images/html.gif"/>
         <img src="/images/css.gif"/>
         <img src="/images/xhtml.gif"/>
         <img src="/images/wml_logo.gif"/>
         <img src="/images/javascript.gif"/>
      </div>

      <div >
         Drag and Drop Your Images in this area
      </div>

   </body>
</html>
```

This will produce following result −



SCRIPT.ACULO.US - SORTING ELEMENTS

Many times, you need to provide the user with the ability to reorder elements (such as items in a list) by dragging them.

Without drag and drop, reordering can be a nightmare, but *script.aculo.us* provides extended reordering support out of the box through the *Sortable* class. The element to become *Sortable* is passed to the *create()* method in the Sortable namespace.

A Sortable consists of item elements in a container element. When you create a new Sortable, it takes care of the creation of the corresponding *Draggables* and *Droppables*.

To use script.aculo.us's Sortable capabilities, you'll need to load the **dragdrop** module, which also requires the **effects** module. So your minimum loading for script.aculo.us will look like this −

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
<script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,dragdrop"></script>
```

Sortable Syntax

Here is the syntax of the create() method to create a sortable item. The *create()* method takes the *id* of a container element and sorts them out based on the passed options.

```
Sortable.create('id_of_container',[options]);
```

Use *Sortable.destroy* to completely remove all the event handlers and references to a Sortable created by Sortable.create.

NOTE − A call to Sortable.create, implicitly calls on Sortable.destroy if the referenced element was already a Sortable. Here is the simple syntax to call the destroy function.

```
Sortable.destroy( element );
```

Sortable Options

You can use one or more of the following options while creating your Sortable object.

Option	Description
tag	Specifies the type of the elements within the sortable container that are to be sortable via drag and drop. Defaults to 'li'.
only	Specifies a CSS class name, or array of class names, that a draggable item must posses in order to be accepted by the drop target. This is similar to the *accept* option of Draggable. By default, no class name constraints are applied.
overlap	One of false, *horizontal* or *vertical*. Controls the point at which a reordering is triggered. Defaults to *vertical*.
constraint	One of false, *horizontal* or *vertical*. Constrains the movement of dragged sortable elements. Defaults to *vertical*.
containment	Enables dragging and dropping between Sortables. Takes an array of elements or element-ids. Important note: To ensure that two way dragging between containers is possible, place all Sortable.create calls after the container elements.
handle	Same as the Draggable option of the same name, specifying an element to be used to initiate drag operations. By default, each element is its own handle.
hoverclass	Specifies a CSS class name to be applied to non-dragged sortable elements as a dragged element passes over them. By default, no CSS class name is applied.
ghosting	Similar to the Draggable option of the same name, If true, this option causes the original element of a drag operation to stay in place while a semi-transparent copy of the element is moved along with the mouse pointer. Defaults to *false*. This option does not work with IE.
dropOnEmpty	If true, it allows sortable elements to be dropped onto an empty list. Defaults to *false*.
scroll	If the sortable container possesses a scrollbar due to the setting of the CSS overflow attribute, this option enables auto-scrolling of the list beyond the visible elements. Defaults to *false*.
scrollSensitivity	When scrolling is enabled, it adjusts the point at which scrolling is triggered. Defaults to 20.
scrollSpeed	When scrolling is enabled, it adjusts the scroll speed. Defaults to 15.
tree	If true, it enables sorting with sub-elements within the sortable element. Defaults to false.
treeTag	If the tree option is enabled, it specifies the container element type of the sub-element whose children takes part in the sortable behavior. Defaults to 'ul'.

You can provide the following callbacks in the options parameter:

Option	Description
onChange	A function that will be called upon whenever the sort order changes while dragging. When dragging from one Sortable to another, the callback is called once on each

	Sortable. Gets the affected element as its parameter.
onUpdate	A function that will be called upon the termination of a drag operation that results in a change in element order.

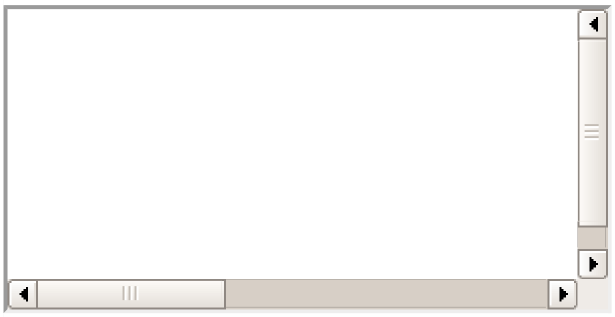
Sorting Examples

This demo has been verified to work in IE 6.0. It also works in the latest version of Firefox.

```
<html>
   <head>
      <title>Sorting Example</title>
		
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>
		
      <script type="text/javascript">
		
         window.onload = function() {
            Sortable.create('namelist',{tag:'li'});
         }
			
      </script>
		
      <style type="text/css">
         li { cursor: move; }
      </style>
		
   </head>
   <body>
	
      <p>Drag and drop list items to sort them out</p>
		
      <ul >
         <li>Physics</li>
         <li>Chemistry</li>
         <li>Maths</li>
         <li>Botany</li>
         <li>Sociology</li>
         <li>English</li>
         <li>Hindi</li>
         <li>Sanskrit</li>
      </ul>
		
   </body>
</html>
```

Use our online compiler for a better understanding of the code with different options discussed in the above table.

This will produce following result −

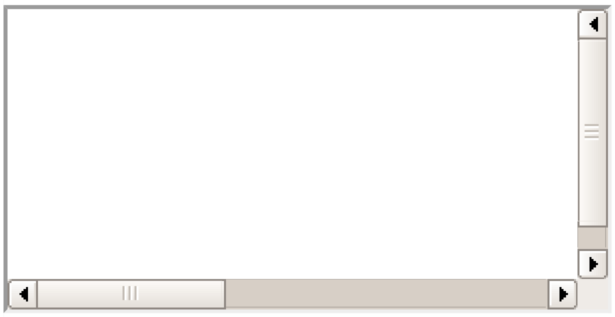


Note the usage of *tag:'li'*. Similarly, you can sort the following list of images available in <div> −

```
<html>
```

```
   <head>
      <title>Sorting Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>

      <script type="text/javascript">

         window.onload = function() {
            Sortable.create('imagelist',{tag:'div'});
         }
      </script>

      <style type="text/css">
         div { cursor: move; }
         img { border: 1px solid red; margin:5px; }
      </style>

   </head>
   <body>

      <p>Drag and drop list images to re-arrange them</p>

      <div >
         <div><img src="/images/wml_logo.gif" alt="WML Logo" /></div>
         <div><img src="/images/javascript.gif" alt="JS" /></div>
         <div><img src="/images/html.gif" alt="HTML" /></div>
         <div><img src="/images/css.gif" alt="CSS" /></div>
      </div>

   </body>
</html>
```

This will produce following result −



Serializing the Sortable Elements

The Sortable object also provides a function *Sortable.serialize()* to serialize the Sortable in a format suitable for HTTP GET or POST requests. This can be used to submit the order of the Sortable via an Ajax call.

Syntax

```
Sortable.serialize(element, options);
```

Options

You can use one or more of the following options while creating your Sortable object.

Option	Description
tag	Sets the kind of tag that will be serialized. This will be similar to what is used in *Sortable.create*.

name	Sets the name of the key that will be used to create the key/value pairs for serializing in HTTP GET/POST format. So if the *name* were to be xyz, the query string would look like − xyz[]=value1 & xyz[]=value2 & xyz[]=value3 Where the values are derived from the child elements in the order that they appear within the container.

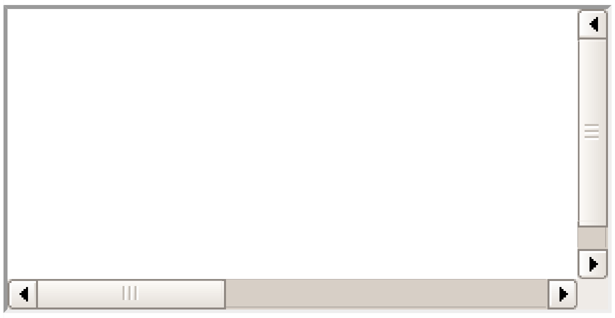
Serialize Examples

In this example, the output of the serialization will only give the numbers after the underscore in the list item IDs.

To try, leave the lists in their original order, press the button to see the serialization of the lists. Now, re-order some elements and click the button again.

```
<html>
   <head>
      <title>Sorting Example</title>
		
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>
		
      <script type="text/javascript">
		
         window.onload = function() {
            Sortable.create('namelist',{tag:'li'});
         }
			
         function serialize(container, name){
            $('display').innerHTML = 'Serialization of ' + $(container).id + ' is:
<br/><pre>' + Sortable.serialize( container,{ name:name} ) + '</pre>';
         }
      </script>
		
      <style type="text/css">
         li { cursor: move; }
      </style>
		
   </head>
   <body>
	
      <p>Drag and drop list items to sort them out properly</p>
		
      <ul >
         <li >Physics</li>
         <li >Chemistry</li>
         <li >Maths</li>
         <li >Botany</li>
         <li >Sociology</li>
         <li >English</li>
      </ul>
		
      <p>Click following button to see serialized list which can be
         passed to backend script, like PHP, AJAX or CGI</p>
			
      <button type="button" value="Click Me"
         onclick="serialize('namelist', 'list')"> Serialize
      </button>
		
      <div ></div>
		
   </body>
</html>
```

This will produce following result −



Moving Items between Sortables

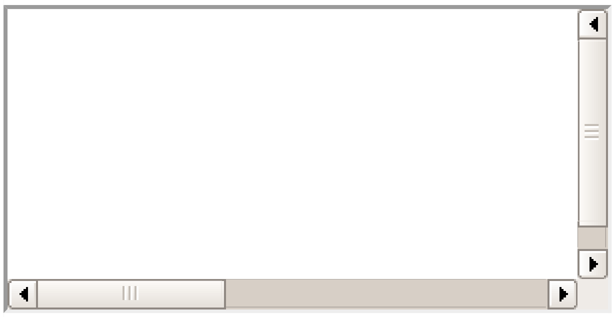
The following example shows how to move items from one list to another list.

```
<html>
   <head>
      <title>Sorting Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>

      <script type="text/javascript">

         window.onload = function() {

            Sortable.create('List1', {containment: ['List1','List2'], dropOnEmpty: true});

            Sortable.create('List2', {containment: ['List1','List2'], dropOnEmpty: true});
         }
      </script>

      <style type="text/css">

         li { cursor: move; }

         ul {
            width: 88px;
            border: 1px solid blue;
            padding: 3px 3px 3px 20px;
         }
      </style>

   </head>
   <body>

      <p>Drag and drop list items from one list to another list</p>

      <div style="float:left">
         <ul >
            <li>Physics</li>
            <li>Chemistry</li>
            <li>Botany</li>
         </ul>
      </div>

      <div style="float:left;margin-left:32px">
         <ul >
            <li>Arts</li>
            <li>Politics</li>
            <li>Economics</li>
            <li>History</li>
            <li>Sociology</li>
         </ul>
      </div>
   </body>
</html>
```

Note that the *containment* option for each container lists both the containers as containment elements. By doing so, we have enabled the child elements to be sorted within the context of their parent; It also enables them to be moved between the two containers.

We set *dropOnEmpty* to true for both the lists. To see the effect this option has on that list, move all the elements from one list into other so that one list is empty. You will find that it is allowing to drop element on empty list.

This will produce following result −



Binding to Ajax

Of course, *onUpdate* is a prime candidate for triggering Ajax notifications to the server, for instance when the user reorders a to-do list or some other data set. Combining *Ajax.Request* and *Sortable.serialize* makes live persistence simple enough −

```
<html>
   <head>
      <title>Sorting Example</title>
		
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js"></script>
      <script type="text/javascript">
		
         window.onload = function() {
            Sortable.create('List' , {onUpdate: function() {
               new Ajax.Request('file.php', {method: "post", parameters:
{data:Sortable.serialize('List')}});
               }
            });
         }
      </script>
		
      <style type="text/css">
         li { cursor: move; }
			
         ul {
            width: 88px;
            border: 1px solid blue;
            padding: 3px 3px 3px 20px;
         }
      </style>
		
   </head>
   <body>
	
      <p>When you will change the order, AJAX Request
         will be made automatically.</p>
			
      <div style="float:left">
         <ul >
            <li >Physics</li>
            <li >Chemistry</li>
            <li >Maths</li>
            <li >Botany</li>
         </ul>
      </div>

   </body>
</html>
```

Sortable.serialize creates a string like: List[]=1 & List[]=2 & List[]=3 &List[]=4, where the numbers are the identifier parts of the list element ids after the underscore.

Now we need to code *file.php*, which will parse posted data as *parse_str($_POST['data']);* and you can do whatever you want to do with this sorted data.

To learn more about AJAX, please go through our simple Ajax Tutorial.

SCRIPT.ACULO.US - CREATE SLIDERS

Sliders are thin tracks with one or more handles on them that the user can drag along the track.

The goal of a slider is to provide an alternative input method for defining a numerical value; the slider represents a range, and sliding a handle along the track defines a value within this range.

Sliders can be in either horizontal or vertical orientation. When horizontal, the left end of the track usually represents the lowest value, while in a vertical orientation, the bottom of the slide is usually the lowest value.

To use script.aculo.us's slider capabilities, you'll need to load the slider.js module along with the prototype.js module. So your minimum loading for script.aculo.us will look like this −

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
<script type="text/javascript" src="/javascript/scriptaculous.js?load=slider">< /script>
```

Creating a Slider Control

Creating a slider is, as usual, a matter of constructing a custom object over a few existing elements in your page's DOM. You'll need two elements here, one for the *handle* and one for the *track* as follows −

```
new Control.Slider(handle, track [ , options ] );
```

The track element is usually a <div>, and the handle element is a <div> or <span> within the track element. Both can be passed either by their id= or by direct DOM references, as usual.

Sliders Options

You can use one or more of the following options while creating your Slider object.

Option	**Description**
Axis	Defines the orientation of the control as *horizontal* or *vertical*. The default orientation is *horizontal*.
Range	Defines the range of the slider values as an instance of a Prototype ObjectRange instance. Defaults to 0 through 1.
Values	Defines the discrete set of values that the slider can acquire. If omitted, all values within the range can be set.
sliderValue	Sets the initial value of the slider. If omitted, the value represented by the leftmost (or top-most) edge of the slider is the initial value.
Disabled	If true, it creates a slide that is initially disabled. Obviously defaults to false.
setValue	Will update the slider's value and thus move the slider handle to the appropriate

	position.
setDisabled	Will set the slider to the disabled state (disabled = true).
setEnabled	Will set the slider to the enabled state (disabled = false).

You can provide the following callbacks in the options parameter −

Option	**Description**
onSlide	Called whenever the Slider is moved by dragging. The called function gets the slider value as its parameter.
onChange	Called whenever the Slider has finished moving or has had its value changed via the setSlider Value function. The called function gets the slider value as its parameter.

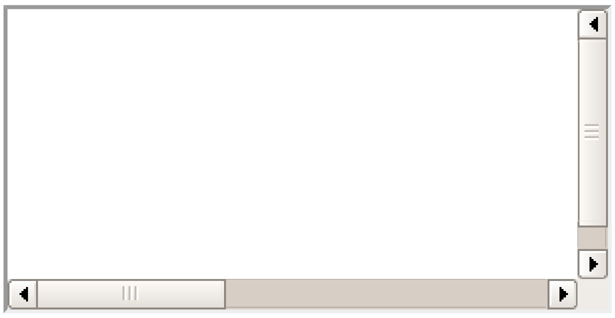
Sliders Example

```
<html>
   <head>
      <title>Sliders Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript" src="/javascript/scriptaculous.js?load=slider"
></script>

      <script type="text/javascript">

         window.onload = function() {
            new Control.Slider('handle1' , 'track1',{
            range: $R(1,100), values: [1,25,50,75,100], sliderValue: 1, onChange:
function(v){
               $('changed').innerHTML = 'Changed Value : '+v;
               },

            onSlide: function(v) {
               $('sliding').innerHTML = 'Sliding Value: '+v;
               }
            });
            new Control.Slider('handle2' , 'track2', { range: $R(1,100),
axis:'vertical', sliderValue: 1, onChange: function(v){
               $('changed').innerHTML = 'Changed Value : '+v;
               }

            onSlide: function(v) {
               $('sliding').innerHTML = 'Sliding Value: '+v;
               }
            });
         }
      </script>

      <style type="text/css">

         h1{ font-size: 1.5em; }

         .track {
            background-color: #aaa;
            position: relative;
            height: 0.5em; width: 10em;
            cursor: pointer; z-index: 0;
         }

         .handle {
            background-color: red;
            position: absolute;
            height: 1em; width: 0.25em; top: -0.25em;
            cursor: move; z-index: 2;
         }

         .track.vertical {
            width: 0.5em; height: 10em;
         }

         .track.vertical .handle {
            width: 1em; height: 0.25em; top: 0; left: -0.25em;
         }

      </style>
   </head>
   <body>

      <h1>Simple sliders</h1>

      <div   >
         <div   ></div>
      </div>

      <p  ></p>
      <p  ></p>

      <div   >
         <div   ></div>
      </div>

   </body>
</html>
```

Points to note:

- You can change the slider image of any slider using CSS. Use CSS properties *background-image: url(track.gif)* and *background-repeat: no-repeat* to set the slider image.
- The range value can be specified using $R(minValue, MaxValue). For example, $R(1, 100).
- The range value can be specified in terms of specific values. For example values: [1,25,50,75,100]. In this case, the slider would only achieve the discrete values listed as the handle was moved.
- At any time, the value of the slider can be set under program control by calling the setValue() method of the slider instance, as in: sliderInstance.setValue(50);

This will produce following result −



SCRIPT.ACULO.US - AUTO COMPLETION

Out of the box, script.aculo.us supports two sources for auto-completion −

- Remote sources (obtained through Ajax),
- Local sources (string arrays in your web page's scripts).

Depending on the source you're planning to use, you'll instantiate *Ajax.Autocompleter* or *Autocompleter.Local*, respectively. Although equipped with specific options, these two objects share a large feature set and provide a uniform user experience.

There are four things you'll always pass to these objects while building them −

- The text field you want to make autocompletable. As usual, you can pass the field itself or the value of its id = attribute.
- The container for autocompletion choices, which will end up holding a <ul></li> list of options to pick from. Again, pass the element directly or its **id =**. This element is most often a simple <div>.</p></li>
- The data source, which will be expressed, depending on the source type, as a JavaScript array of strings or as a URL to the remote source.
- Finally, the options. As always, they're provided as a hash of sorts, and both autocompletion objects can make do with no custom option; there are suitable defaults for everything.

To use script.aculo.us's autocompletion capabilities, you'll need to load the controls.js and effects.js modules along with the prototype.js module. So, your minimum loading for script.aculo.us will look like this −

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
<script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,controls"></script>
```

Creating an Ajax Auto-Completer

The construction syntax is as follows −

```
new Ajax.Autocompleter(element, container, url [ , options ] )
```

The constructor for the *Ajax.Autocompleter* accepts four parameters −

- The element name or reference to a text field that is to be populated with a data choice.
- The element name or reference to a <div> element to be used as a menu of choices by the control.
- The URL of the server-side resource that will supply the choices.
- The usual options hash.

Options

You can use one or more of the following options while creating your Ajax.Autocompleter object.

Option	Description
paramName	The name of the query parameter containing the content of the text field that is posted to the server-side resource. Defaults to the name of the text field.
minChars	Number of characters that must be entered before a server-side request for choices can be fired off. Defaults to 1.
Frequency	The interval, in seconds, between internal checks to see if a request to the server-side resource should be posted. Defaults to 0.4.
Indicator	The id or reference to an element to be displayed while a server-side request for choices is underway. If omitted, no element is revealed.
Parameters	A text string containing extra query parameters to be passed to the server-side resource.

updateElement	A callback function to be triggered when the user selects one of the choices returned from the server that replaces the internal function that updates the text field with the chosen value.
afterUpdateElement	A callback function to be triggered after the updateElement function has been executed.
Tokens	A single text string, or array of text strings that indicate tokens to be used as delimiters to allow multiple elements to be entered into the text field, each of which can be auto-completed individually.

Example

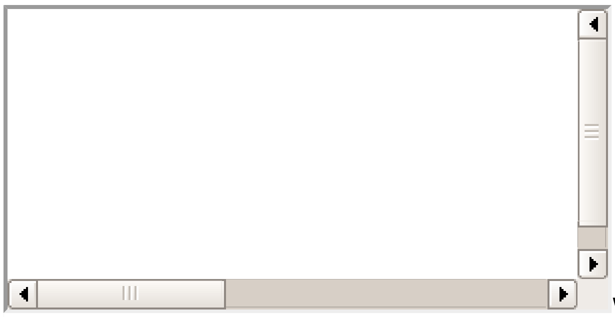
```
<html>
   <head>
      <title>Simple Ajax Auto-completer Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,controls"></script>
      <script type="text/javascript">

         window.onload = function() {

            new Ajax.Autocompleter( 'autoCompleteTextField', 'autoCompleteMenu',
'/script.aculo.us/serverSideScript.php', {});
         }
      </script>
   </head>
   <body>

      <p>Type something in this box and then select suggested option from the list </p>

      <div>
         <label>Text field:</label>
         <input type="text" />
         <div ></div>
      </div>

   </body>
</html>
```

Now, we need a server side to access this page and serve the data source URL (serverSideScript.php). You will keep a complete logic to display suggestions in this script.

Just for example, we are keeping a simple HTML text in *serverSideScript.php*. You can write your script using CGI, PHP, Ruby, or any other server side scripting to choose appropriate suggestions and format them in the form of <ul><li>...</li></ul> and pass them back to the caller program.

```
<ul>
   <li>One</li>
   <li>Two</li>
   <li>Three</li>
   <li>Four</li>
   <li>Five</li>
   <li>Six</li>
</ul>
```

This will produce following result −

with different options discussed in the above table.

Creating a Local Auto-Completer

Creating a local auto-completer is almost identical to creating an Ajax Auto-completer as we have discussed in the previous section.

The major difference lies in how the backing data set to use for auto-completion is identified to the control.

With an Ajax Auto-completer, we have supplied the URL of a server-side resource that would perform the necessary filtering, given the user input, and return only the data elements that matched. With a Local Autocompleter, we supply the full list of data element instead, as a JavaScript String array, and the control itself performs the filtering operation within its own client code.

The whole construction syntax is actually as follows −

```
new Autocompleter.Local(field, container, dataSource [ , options ] );
```

The constructor for the Autocompleter.Local accepts four parameters −

- The element name or reference to a text field that is to be populated with a data choice.
- The element name or reference to a <div> element to be used as a menu of choices by the control
- For the third parameter, instead of a URL as with the server-assisted auto-completer, we supply a small String array, which contains all of the possible values.
- The usual options hash.

Options

You can use one or more of the following options while creating your Autocompleter.Local object.

Option	Description
Choices	The number of choices to display. Defaults to 10.
partialSearch	Enables matching at the beginning of words embedded within the completion strings. Defaults to true.
fullSearch	Enables matching anywhere within the completion strings. Defaults to false.
partialChars	Defines the number of characters that must be typed before any partial matching is attempted. Defaults to 2.
ignoreCase	Ignores case when matching. Defaults to true.

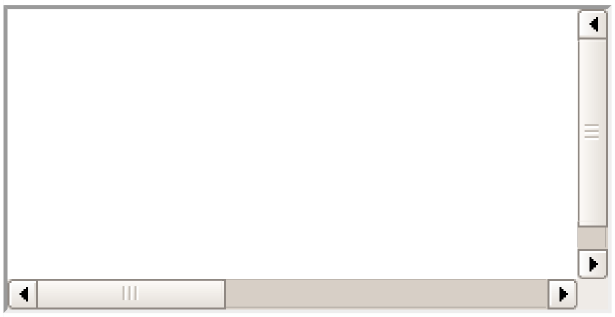
Example

```
<html>
   <head>
      <title>Simple Ajax Auto-completer Example</title>
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,controls"></script>
      <script type="text/javascript">

         window.onload = function() {

            new Autocompleter.Local('autoCompleteTextField', 'autoCompleteMenu',
['abcdef','abcdeg','xyzabcefg', 'tybabdefg','acdefg'], {ignoreCase:false});
         }
      </script>
   </head>
   <body>

      <p>Type something in this box and then select suggested option from the list </p>

      <div>
         <label>Text field:</label>
         <input type="text" />
         <div ></div>
      </div>

   </body>
</html>
```

When displayed, and after the character 'a' is typed into the text box, it displays all the matching options.

Use our online compiler for a better understanding of the code with different options discussed in the above table.

This will produce following result –



SCRIPT.ACULO.US - IN-PLACE EDITING

In-place editing is one of the hallmarks of Web 2.0.style applications.

In-place editing is about taking non-editable content, such as a <p>, <h1>, or <div>, and letting the user edit its contents by simply clicking it.

This turns the static element into an editable zone (either singleline or multiline) and pops up submit and cancel buttons (or links, depending on your options) for the user to commit or roll back the modification.

It then synchronizes the edit on the server side through Ajax and makes the element non-editable again.

To use script.aculo.us's in-place editing capabilities, you'll need to load the controls.js and effects.js modules along with the prototype.js module. So, your minimum loading for script.aculo.us will look like this –

```
<script type="text/javascript" src="/javascript/prototype.js"></script>
<script type="text/javascript" src="/javascript/scriptaculous.js?
load=effects,controls"></script>
```

Creating an In-Place Text Editor

The whole construction syntax is as follows −

```
new Ajax.InPlaceEditor(element, url [ , options ] )
```

The constructor for the Ajax.InPlaceEditor accepts three parameters −

- The target element can either be a reference to the element itself or the id of the target element.
- The second parameter to the Ajax.InPlaceEditor specifies the URL of a server-side script that is contacted when an edited value is completed.
- The usual options hash.

Options

You can use one or more of the following options while creating your Ajax.InPlaceEditor object.

Option	Description
okButton	A Boolean value indicating whether an "ok" button is to be shown or not. Defaults to true.
okText	The text to be placed on the ok button. Defaults to "ok".
cancelLink	A Boolean value indicating whether a cancel link should be displayed. Defaults to true.
cancelText	The text of the cancel link. Defaults to "cancel".
savingText	A text string displayed as the value of the control while the save operation (the request initiated by clicking the ok button) is processing. Defaults to "Saving".
clickToEditText	The text string that appears as the control "tooltip" upon mouse-over.
rows	The number of rows to appear when the edit control is active. Any number greater than 1 causes a text area element to be used rather than a text field element. Defaults to 1.
cols	The number of columns when in active mode. If omitted, no column limit is imposed.
size	Same as cols but only applies when rows is 1.
highlightcolor	The color to apply to the background of the text element upon mouse-over. Defaults to a pale yellow.
highlightendcolor	The color to which the highlight color fades to as an effect. **Note** − support seems to be spotty in some browsers.
loadingText	The text to appear within the control during a load operation. The default is "Loading".
loadTextURL	Specifies the URL of a server-side resource to be contacted in order to load the initial value of the editor when it enters active mode. By default, no backend load operation takes place and the initial value is the text of the target element.
externalControl	An element that is to serve as an "external control" that triggers placing the editor into an active mode. This is useful if you want another button or other

	element to trigger editing the control.
ajaxOptions	A hash object that will be passed to the underlying Prototype Ajax object to use as its options hash.

Callback Options

Additionally, you can use any of the following callback functions in the options parameter

Function	Description
onComplete	A JavaScript function that is called upon successful completion of the save request. The default applies a highlight effect to the editor.
onFailure	A JavaScript function that is called upon failure of the save request. The default issues an alert showing the failure message.
callback	A JavaScript function that is called just prior to submitting the save request in order to obtain the query string to be sent to the request. The default function returns a query string equating the query parameter "value" to the value in the text control.

CSS Styling and DOM id Options

You can also use one the following options to control the behavior of in place editor −

Option	Description
savingClassName	The CSS class name applied to the element while the save operation is in progress. This class is applied when the request to the saving URL is made, and is removed when the response is returned. The default value is "inplaceeditor-saving".
formClassName	The CSS class name applied to the form created to contain the editor element. Defaults to "inplaceeditor-form".
formId	The id applied to the form created to contain the editor element.

Example

```
<html>
   <head>
      <title>Simple Ajax Auto-completer Example</title>

      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,controls"></script>
      <script type="text/javascript">

         window.onload = function() {
            new Ajax.InPlaceEditor('theElement', '/script.aculo.us/transform.php',
{formId: 'whatever', okText: 'Upper me!', cancelText: 'Never mind'});
         }
      </script>
   </head>
   <body>

      <p>Click over the "Click me!" text and then change text and click OK.</p>
      <p>Try this example with different options.</p>

      <div >
```

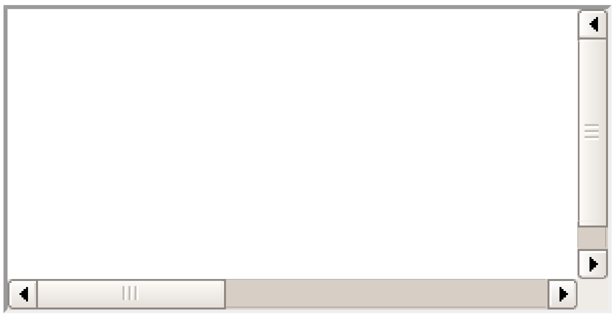
```
         Click me!
      </div>

   </body>
</html>
```

When displayed, click and edit the text. This rather trivial PHP script converts the value of a query parameter with the key "value" to its uppercase equivalent, and writes the result back to the response.

Here is the content of transform.php script.

```
<?php

   if( isset($_REQUEST["value"]) )
   {
      $str = $_REQUEST["value"];
      $str = strtoupper($str);
      echo "$str";
   }

?>
```

This will produce following result −



The In-Place Collection Editor Options

There is one more object called *Ajax.InPlaceCollectionEditor*, which supports in-place editing and gives you the option to select a value from the given options.

The whole construction syntax is as follows −

```
new Ajax.InPlaceCollectionEditor(element, url [ , options ] )
```

The constructor for the Ajax.InPlaceCollectionEditor accepts three parameters:

- The target element can either be a reference to the element itself or the id of the target element
- The second parameter to the Ajax.InPlaceEditor specifies the URL of a server-side script that is contacted when an edited value is completed.
- The usual options hash.

Options

Aside from the addition of the collection option, the list of options for the In-Place Collection Editor is a subset of the options inherited from the In-Place Text Editor.

Option	Description
okButton	A Boolean value indicating whether an "ok" button is to be shown or not. Defaults to true.

okText	The text to be placed on the ok button. Defaults to "ok".
cancelLink	A Boolean value indicating whether a cancel link should be displayed. Defaults to true.
cancelText	The text of the cancel link. Defaults to "cancel".
savingText	A text string displayed as the value of the control while the save operation (the request initiated by clicking the ok button) is processing. Defaults to "Saving".
clickToEditText	The text string that appears as the control "tooltip" upon mouse-over.
Highlightcolor	The color to apply to the background of the text element upon mouse-over. Defaults to a pale yellow.
Highlightendcolor	The color to which the highlight color fades to as an effect. **Note** − support seems to be spotty in some browsers.
Collection	An array of items that are to be used to populate the select element options.
loadTextUrl	Specifies the URL of a server-side resource to be contacted in order to load the initial value of the editor when it enters active mode. By default, no backend load operation takes place and the initial value is the text of the target element. In order for this option to be meaningful, it must return one of the items provided in the collection option to be set as the initial value of the select element.
externalControl	An element that is to serve as an "external control" that triggers placing the editor into active mode. This is useful if you want another button or other element to trigger editing the control.
ajaxOptions	A hash object that will be passed to the underlying Prototype Ajax object to use as its options hash.

Callback Options

Additionally, you can use any of the following callback functions in the options parameter −

Function	Description
onComplete	A JavaScript function that is called upon successful completion of the save request. The default applies a highlight effect to the editor.
onFailure	A JavaScript function that is called upon failure of the save request. The default issues an alert showing the failure message.

CSS Styling and DOM id Options

You can also use one the following options to control the behavior of in-place editor −

Option	Description
savingClassName	The CSS class name applied to the element while the save operation is in progress. This class is applied when the request to the saving URL is made, and is removed when the response is returned. The default value is "inplaceeditor-saving".

formClassName	The CSS class name applied to the form created to contain the editor element. Defaults to "inplaceeditor-form".
formId	The id applied to the form created to contain the editor element.

Example

```
<html>
   <head>
      <title>Simple Ajax Auto-completer Example</title>
		
      <script type="text/javascript" src="/javascript/prototype.js"></script>
      <script type="text/javascript"  src="/javascript/scriptaculous.js?
load=effects,controls"></script>
      <script type="text/javascript">
		
         window.onload = function() {
            new Ajax.InPlaceCollectionEditor('theElement',
'/script.aculo.us/transform.php', {formId: 'whatever', okText: 'Upper me!', cancelText:
'Never mind', collection: ['one','two','three','four','five']});
         }
      </script>
   </head>
   <body>
	
      <p>Click over the "Click me!" text and then change text and click OK.</p>
      <p>Try this example with different options.</p>
		
      <div >
         Click me!
      </div>
		
   </body>
</html>
```

Here is the content of the transform.php script.

```
<?php

   if( isset($_REQUEST["value"]) )
   {
      $str = $_REQUEST["value"];
      $str = strtoupper($str);
      echo "$str";
   }

?>
```

When displayed, click and select one of the displayed options. This rather trivial PHP script converts the value of the query parameter with the key "value" to its uppercase equivalent, and writes the result back to the response.

Use our online compiler for a better understanding of the code with different options discussed in the above table.

This will produce following result −