SCRIPT.ACULO.US - SORTING ELEMENTS

Many times, you need to provide the user with the ability to reorder elements *suchasitemsinalist* 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 –

```
<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

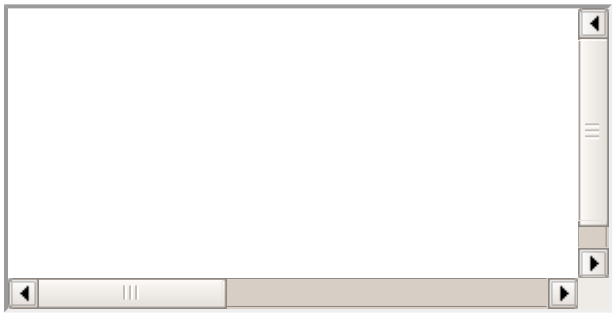
In this example, the output of the serialization will only give the numbers after the underscore in the list item IDs.

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.

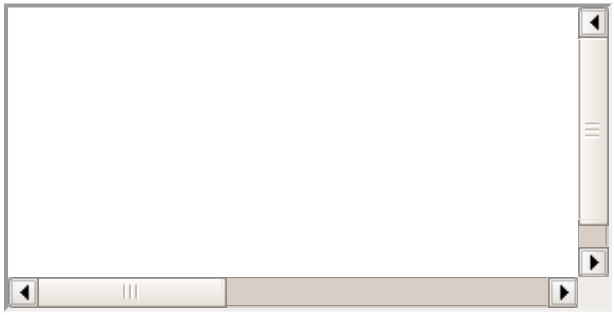
```
<html>
   <head>
      <title>Sorting Example</title>

      <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.