Draft - Advanced Reorderer Tutorial

This tutorial is currently being updated to the latest API. Until this warning is removed, please do not use this information.

This page will walk the reader through an example of adding the Reorderer to an HTML file.
For more general information about the Reorderer API, see Reorderer API.

Tutorial: Adding the Reorderer to a List

You are organizing a small conference, and there is lots to do. Being the organized person you are, you create a to-do list. You'd like to use the Reorderer to allow the order of items to be changed.

On This Page

Still need help?

Join the infusion-users mailing list and ask your questions there.

Here's the HTML file for your to-do list before adding the Reorderer - it uses a simple ordered list:

```
<!DOCTYPE html      PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Conference Plannaing Tasks</title>
</head>

<body>
  <h3>Conference Planning To-do List</h3>

  <ol>
    <li>
      Select the date
    </li>
    <li>
      Create promotional plan
      <ul>
        <li>email campaign</li>
        <li>web site</li>
        <li>print materials</li>
      </ul>
    </li>
    <li>
      Book facilities
      <ul>
              <li>Select building</li>
              <li>Book rooms</li>
      </ul>
    </li>
    <li>
      Book caterer
    </li>
    <li>
      Book block of hotel rooms
    </li>
    <li>
      Develop a conference daily schedule of events
    </li>
    <li>
      Book speakers
      <ul>
              <li>Keynote</li>
              <li>Internal/employee</li>
      </ul>
    </li>
    <li>
    Plan the registration process
      <ul>
        <li>Deadline for early bird and final registrations</li>
        <li>Create/enter data into database</li>
        <li>Create/upload information for web registration</li>
      </ul>
    </li>
    <li>
      Book audio visual equipment and technicians
    </li>
    <li>
      Create name tags and welcome packages
      <ul>
        <li>schedule of events</li>
        <li>maps, tickets</li>
        <li>paper, pen, trinkets</li>
        <li>instructions</li>
      </ul>
    </li>
  </ol>
</body>
</html>
```

You can download a copy of this file from here: simple-todo-list.html

The Mark-up

The Reorderer requires that all orderable items (in our case, the ordered list items) be contained within an element (the "container") with a known ID. We'll consider the ordererd list element itself to be our container. Let's put an `id` attribute on the `<ol>` element:

```
<ol id="todo-list">
```

The Javascript

We'll need to include the necessary Javascript files in `sortable-todo-list.html`. Let's assume that you've downloaded the Fluid component library, and that it lives in a folder called `fluid-components`. In the header of the HTML file, we'll link to the require files:

```
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/jquery/jquery-1.3.2.js"><
/script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/jquery/jquery.keyboard-a11y.
js"></script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/jquery/ui.core.js"></script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/jquery/ui.draggable.js"><
/script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/jquery/jARIA.js"></script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/fluid/Fluid.js"></script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/fluid/GeometricManager.js"><
/script>
<script type="text/javascript" src="fluid-components/src/webapp/fluid-components/js/fluid/Reorderer.js"><
/script>
<!--  Ensure jQuery is in no conflicts mode -->
<script language="JavaScript" type="text/javascript">
  jQuery.noConflict();
</script>
```

(If you've saved the demonstration file elsewhere, you'll have to adjust the paths.)

Next, we'll need some Javascript to initialize the Reorderer. We'll create an initialization function in a separate file and include that file in our dependencies in `sortable-todo-list.html`:

```
<script type="text/javascript" src="todo-list.js"></script>
```

and we'll add a line of script at the end of the mark-up, in `sortable-todo-list.html`, to invoke the initialization:

```
<script type="text/javascript">
  demo.initTodoList ();
</script>
```

The initialization function in `todo-list.js`, `demo.initTodoList()`, will instantiate the Reorderer and configure it. For that, it will need the three parameters for the Reorderer constructor.

First, we need the container element for the Reorderer. We identified this element by attaching an ID to it, so let's use that:

```
var todoListContainer = fluid.utils.jById ("todo-list");
```

Next, we need a Javascript function to identify the orderable elements. In our example, all of the top level `li` elements in the list are orderable, so let's use jQuery to select all of the `li` elements that are direct children of the container:

```
var myOrderableFinder  = function () {
  return jQuery (">li", todoListContainer);
};
```

Another way to identify the orderable elements would be using IDs. We could add an ID to each orderable list item, one with a unique prefix that we can identify with a function, like this:

```
<li id="myUniquePrefix.orderable1">
      Select the date
    </li>
    <li id="myUniquePrefix.orderable2">
      Create promotional plan
      <ul>
        <li>email campaign</li>
        <li>web site</li>
        <li>print materials</li>
      </ul>
    </li>
    <li id="myUniquePrefix.orderable4">
      Book caterer
    </li>
    ...
```

Then we can use jQuery to find all the elements whose ID starts with that prefix, like this:

```
var myOrderableFinder = function () {
  return jQuery ("[id^=myUniquePrefix]", todoListContainer);
};
```

These are just example. You can use any method you like, so long as the function returns all of the elements you wish to reorder.

Finally, we need to select a LayoutHandler for use with our Reorderer. The LayoutHandler is what interprets keystrokes and understands what the arrow keys mean. The Reorderer includes a `ListLayoutHandler` that will suffice quite nicely for our ordered list, so we'll just use that. It is instantiated with the orderable finder function we wrote:

```
var layoutHandler = new fluid.ListLayoutHandler (myOrderableFinder)
```

(In the next release of the Reorderer, you will be able to easily write your own LayoutHandler. In the meantime, the out-of-the-box handlers should cover most types of supported markup.)

So now that we have all the ducks lined up, we just need to add the Reorderer to our list by instantiating it and returning it:

```
demo.initTodoList = function () {
  var todoListContainer = fluid.utils.jById ("todo-list");

  var myOrderableFinder = function () {
    return jQuery ("[id^=myUniquePrefix]", todoListContainer);
  };

  var layoutHandler = new fluid.ListLayoutHandler (myOrderableFinder);

  return new fluid.Reorderer (todoListContainer, myOrderableFinder, layoutHandler);
};
```

That's it! Our to-do items are now orderable using the mouse or the keyboard.

You can see the final version of the Javascript file at: todo-list.js.

You can see a working version of the finished product at: sortable-todo-list.html.

You can download a copy of the HTML file from here: sortable-todo-list.html.

The Styles

You've probably noticed, looking at sortable-todo-list.html, that it's very hard to tell what's orderable, and when you can or can't move things around. There are a number of "interesting moments" that happen while items are being reordered. These include, for example, the moment an item is "picked up" using the mouse, or the moment an item is selected using the keyboard. The Reorderer automatically adds CSS classes to items identifying all the interesting moments, so we can define styles for these classes to control the appearance of these moments.

Let's create a stylesheet. The first style we'll add is `orderable-default`, which is attached to all orderable elements in their default state. Let's give them a grey background, so that we can see them clearly:

```
.orderable-default{
  background-color: #eee;
}
```

When the mouse cursor hovers over a draggable item, the `orderable-hover` class is applied. Let's make the background light yellow when that happens:

```
.orderable-hover{
  background-color: lightyellow;
  cursor: move;
}
```

While an item is being dragged using the mouse, an avatar is created and is given the class `orderable-avatar` Let's make the avatar a bit see-through:

```
.orderable-avatar {
  opacity: 0.55;
  width: 300px;
  border: 0;
}
```

While an avatar is being dragged, an indicator is created to show where the item will be dropped. The avatar is given a class of `orderable-drop-marker`. Let's make it red:

```
.orderable-drop-marker{
  background-color: red;
  height: 10px;
}
```

The keyboard can also be used to navigate to and move items. Different styles are used for that. The `orderable-selected` style is used to identify which item currently has focus:

```
.orderable-selected{
  background-color: #ddd;
}
```

The `orderable-dragging` style is applied when the Control key is pressed, to indicate that the item is being moved:

```
.orderable-dragging {
  background-color: lightyellow;
}
```

That's it. If we link to this stylesheet in our file, all of the interesting moments will now be visually apparent:

```
<link href="todo-list.css" type="text/css" rel="stylesheet"/>
```

You can see the final stylesheet here: todo-list.css.

You can see a working version of the final styled version at: sortable-styled-todo-list.html.

You can download a copy of this file from here: sortable-styled-todo-list.html.