Specifications for Table Navigation and Event Management

Introduction:

The goal of this project is to produce a JavaScript component to encapsulate user interaction with an HTML table. This component will respond to DOM events within the table, and translate those DOM events into a defined set of application specific events. It will also encapsulate keyboard navigation of the HTML table. The details of these events, and interactions are part of this specification.

Implementation:

The total specification will be produced as a single JavaScript component, in one file.

The product may take advantage of the following open source libraries, it is assumed that these will be included in the page, the component is not responsible for loading them:

- jQuery
- Underscore
- Backbone

All event work will use, or extend, the Backbone.Events module.

Sample HTML table:

The following markup is an example of an HTML data table produced by our product. The requirements will reference this example

```
<div id="divOne">
<table class=”table dataTableContainer”>

<thead>
  <tr>
    <th scope="col"></th>
    <th scope="col">Name</th>
    <th scope="col">Age</th>
    <th scope="col">Gender</th>
```

```
  </tr>
</thead>
<thead>
  <tr>
    <th></th>
    <th></th>
    <th></th>
    <th></th>
  </tr>

</thead>

<tbody>
  <tr>
    <th></th>
    <td data-meta-column="ELEMENT_ATTR_C1086.NAME">Row 1, Cell 1</td>
    <td data-meta-column="ELEMENT_ATTR_C1086.AGE">Row 1, Cell 2</td>
    <td data-meta-column="ELEMENT_ATTR_C1086.GENDER">Row 1, Cell 3</td>
  </tr>
  <tr>
    <th></th>
    <td data-meta-column="ELEMENT_ATTR_C1086.NAME">Row 2, Cell 1</td>
    <td data-meta-column="ELEMENT_ATTR_C1086.AGE">Row 2, Cell 2</td>
    <td data-meta-column="ELEMENT_ATTR_C1086.GENDER">Row 2, Cell 3</td>
  </tr>
</tbody>

</table>
</div>
```

Although this table only contains four columns and the body only contains two rows, in practice the number of rows and columns is unspecified, the component must be capable of working with an HTML table of any dimension.

In general there will be two *`thead`* sections, although one of them may not always be visible to the user.

There will be only one tbody section. Within the *`tbody`* section the first child of a row (*`tr`*) may be a *`th`* element, or it may be a *`td`* element. The example shows a *`th`*, but that will not always be true.

In this example there is only one HTML5 '`data-`' attribute, but there no specification for the number these attributes. It may be zero, or many. The component will assemble these into a JavaScript object when triggering an event (see specification below).

In this example there is no *`tfoot`* element and that can be considered outside the scope of this

project.

There will be more than one such HTML table in the HTML document (web page) so each component must operate within the scope of a single table, each being a discrete object, so events, and API calls for one instance do not affect other instances.

Navigation:

Following are the requirements for the keyboard and mouse navigation of the table.
The component must support keyboard navigation, using the arrow keys, and tab key, through the table. Once a cell in the tbody has focus, the user's interaction with the arrow keys must move the focus through the table.
1) Up arrow must move the focus to the cell with the same position in the previous row, unless it is in the first row.
2) Down arrow must move the focus to the cell with the same position in the next row, unless it is in the last row.
3) Left arrow must move the focus to the previous cell in the same row, unless it is in the first cell, in which case it moves to the last cell in the previous row.
4) Right arrow must move the focus to the next cell in the same row, unless it is in the last cell, in which case it moves to the first cell in the next row.
5) The tab key will behave like the right arrow key, except that it can move out of the table to the next element in the page capable of accepting focus.
6) If the shift key is held down while the down arrow key is pressed multiple times, then multiple cells (forming a column) should be selected. Only the last cell will have focus in this case.
If the navigation is not possible – for example, the left arrow key is pressed from the first cell of the first row, then it is a "no-op" for navigation but should still fire an appropriate event.

The component must check to see that the prescribed cell is capable of receiving focus and if necessary set the "tabindex" attribute appropriately.

The navigation component shall fire events before leaving a cell, and after entering a cell. In addition it shall fire an event before leaving a row, and after entering a row.

Events:

The component will provide an API (details in the API section) to subscribe, and unsubscribe from specific events. When a user interaction (a navigation keystroke, or click in the thead, or tbody) occurs the component will raise an event to subscribed listeners, and pass the name of the event, and an event object (defined in the API section).

The following events will be provided:

‘**cellSelect’** : This will be fired when a “tbody td” is clicked, or after the focus is set on a “td” resulting from keyboard navigation.

‘**cellDeSelect’** : This will be fired before a “tbody td” loses focus.

‘**rowSelect’** : This will be fired after the cell selection (focus) moves into a different row.

‘**rowDeSelect’** : This will be fired before the selection (focus) moves from the current row.

‘**columnSelect’** : This will be fired when a ‘thead tr th’ is clicked.

API:

Methods for event subscription:
These are standard Backbone.Events methods.

1) **listenTo**
2) **listenToOnce**
3) **stopListening**

For each event fired, an “Event Object” will be passed to the registered event handler methods.

var eventObject = {
el : … // this is the DOM element. Either a TD, a TR (from tbody) or TH, depending on the event.
cellIndex : // The numeric index of the current td, or th, within the tr.
rowEl : // this is a DOM element, the TR which is the parent of el.
data : {} // this is an array of objects, see below.

};

Members of the eventObject:

- **el** : This is the DOM element (a td, th, or tr) that is the source of the event.
- **cellIndex** : This is the index, or position of the cell within it’s parent row.
- **rowEl** : This is the DOM element, a tr, which is the parent of the el.
- **data** : This is an object. Each name-value pair is based on the HTML5 “data-“ attributes of the el. The event handler will retrieve these from the DOM element and package them in an object, the name will be the HTML attribute value following the “data-“ of the attribute name.

The following programmatic navigation methods will also be provided:

1) **moveToNextCell**
2) **moveToPreviousCell**

3) **moveToNextRow**
4) **moveToPreviousRow**

The interface (or api) for the component is:
api = {

```
listenTo : function(){},
listenToOnce : function(){},
stopListening : function(){},
moveToNextCell : function(el){},
moveToPreviousCell : function(el){},
moveToNextRow : function(el){},
moveToPreviousRow : function(el){}
getRow : function(el){}
```

}
The first three methods having to do with event subscription will have the signature of the corresponding Backbone Event module methods.

Then navigation methods will pass a DOM element, normally a td element, for reference with respect to the selection/focus movement.

Finally, the getRow method will take the 'el' from an eventObject (assumed to be a 'td', in most cases) and return the 'tr' DOM node which is the parent.

Constructor:
The constructor will take a JavaScript object as an argument. That object will contain a CSS selector to identify the table. It will be of the form:

```
var controller = new TableController( { el : 'div#divOne table' } );
```

Testing

An HTML file, named tabtest.html is attached. This contains a copy of the example table, along with some samples, and possible ideas for testing. It is by no means complete, nor is it meant to represent a complete test case.

A set of basic tests to show event handling, and the use of the API functions for navigation are required. This is intended to show the basic capabilities and usage of the component.