14. DATATABLES Buttons Extension.

14.1. Buttons Extension.

A common UI paradigm to use with interactive tables is to present **buttons that will trigger some action** - that may be to alter the table's state, modify the data in the table, gather the data from the table or even to activate some external process. Showing such buttons is an interface that end users are comfortable with, making them feel at home with the table.

The ***Buttons*** library for DataTables provides a framework with common options and API that can be used with DataTables, but is also very extensible, recognising that you will likely want to use buttons which perform an action unique to your applications.

Buttons has four sets of plug-ins that are part of the core software - they are not built into the core, but rather than be included as and when you need them, selecting only the software you require. Other extensions such as Editor and Select also provide buttons for use with this library, with actions unique to their own behaviours. This ensures a consistent interface for the interactions performed with your tables.

14.2. Initialization

The ***Buttons*** library can be initialised and used in two different ways:

1. As part of the ***DataTables constructor*** with the buttons **configuration** option

2. A **new constructor**

It is important to note that **multiple instances of Buttons can be created for use with a DataTable**. This can be particularly useful if you want to present different sets of buttons to the end user - for example above and below the table.

In DataTables

As part of the *DataTables* constructor, the buttons option can be given as an **array of the buttons** you wish to show - this is typically just the button name, although you can provide options to customise the button's actions:

Example:

```
$('#myTable').DataTable( {
    buttons: [
        'copy', 'excel', 'pdf'
    ]
} );
```

When using this method of initialisation, you may also wish to use the **dom** option to tell DataTables where to display the buttons - see below. The **buttons** option can also be given as an object to provide more control over the behaviour of *Buttons*.

Constructor

Button instances can also be created using the Javascript new keyword with the ***$.fn.dataTable.Buttons*** function. This function takes two parameters:

1. The ***DataTable instance*** to apply the buttons to
2. The ***button options*** (this is the same as the options available for the buttons option).

Example:

```
var table = $('#myTable').DataTable();

new $.fn.dataTable.Buttons( table, {
    buttons: [
        'copy', 'excel', 'pdf'
    ]
} );
```

This method of initialisation is particularly useful for cases when you wish to present multiple button instances, since only a single instance can be created using the buttons option.

14.3. Displaying the buttons

With the *Buttons* instance created we still need to display the buttons somewhere on the page so the end user can interact with them! Once again, there are two ways of doing this:

1. ***dom*** - *DataTables' DOM control parameter* - this option is only available if you also use buttons. Additionally, if you use anything other than the DataTables default styling, you probably don't want to use this option!
2. **Direct insertion using the API**

dom parameter

DataTables has a number of table control elements available and where they are placed in the DOM (i.e. the order) is defined by the **dom** parameter. This parameter can be a little confusing at first, but simply put, each letter in it is a DataTables feature. For Buttons the **B** character is the letter to use:

Example:

```
$('#myTable').DataTable( {
    dom: 'Bfrtip',
    buttons: [
        'copy', 'excel', 'pdf'
    ]
} );
```

Direct insertion

If you are using one of the styling integration options, such as for **Bootstrap**, or you wish to have multiple button instances available, the ***buttons().container()*** method can be used to obtain a jQuery object that holds the container element of the button set. Thus you can then insert the element anywhere you want:

With **buttons**:

Example:

```
var table = $('#example').DataTable( {
    buttons: [
        'copy', 'excel', 'pdf'
    ]
} );

table.buttons().container()
    .appendTo( $('.col-sm-6:eq(0)', table.table().container() ) );
```

And with a **new constructor** (note how it also uses the DataTables API to obtain the buttons container):

Example:

```
var table = $('#myTable').DataTable();

new $.fn.dataTable.Buttons( table, {
    buttons: [
        'copy', 'excel', 'pdf'
    ]
} );

table.buttons().container()
    .appendTo( $('.col-sm-6:eq(0)', table.table().container() ) );
```

14.4. Features

Buttons provides the following features:

- Common interface and framework for DataTables related buttons
- Buttons can be activated with assignable key combinations
- Comprehensive API
- Fully internationalisable
- HTML5 export options for modern browsers
- Flash export options for legacy browsers
- Column visibility control
- Print view

14.5. Configuration

The DataTables extensions provide a number of **pre-configured buttons** which can be used without any customisation. However, you will inevitably wish to alter the buttons properties to suit your needs - this can be as simple as changing the displayed text, or as complex as providing a custom callback to modify the layout of a PDF document!

Customising buttons

The initialisation guide shows how buttons can be used by referencing the button type with a simple string matching the button's name:

Example:

```
    $('#myTable').DataTable( {
        buttons: [
            'copy', 'excel', 'pdf'
        ]
    } );
```

This tells Buttons simply to use the default options for the button of that name. The string is expanded automatically by Buttons to a button definition object - this object contains the individual properties of the buttons.

We can also pass an object in that extends from the available buttons:

Example:

```
    $('#myTable').DataTable( {
        buttons: [
            {
                extend: 'copy',
                text: 'Copy to clipboard'
            },
            'excel',
            'pdf'
        ]
    } );
```

Notice that we still have three buttons as before, but the first button, originally represented by a string has been replaced by an object. The ***buttons.buttons.extend*** option is used to tell *Buttons* what button type to use as a base of the button (this is optional if you wish to define a custom button!).

In the case above the ***buttons.buttons.text*** option has been used to set the text for the button. This option is common to all buttons - there are a number of common options (see below), and each button type also has the ability to define options which are specific to itself - the reference documentation for each button should be referred to for the full options available.

Extending upon this concept - using ***{ extend: 'excel' }*** (i.e. no other parameters) is functionality equivalent as simply using 'excel'!

Built in options

The ***Buttons*** library provides a number of core options that are common to all button types. The most commonly used of these are:

- **buttons.buttons.action** - Action to take when the button is activated
- **buttons.buttons.className** - Set the class name for the button
- **buttons.buttons.extend** - Define which button type the button should be based on
- **buttons.buttons.key** - Define an activation key for a button
- **buttons.buttons.text** - The text to show in the button

The full list of options are available in the initialisation options reference.

Key access

One of the more interesting built-in options for Buttons is the **buttons.buttons.key** option. This provides the ability to assign a custom keystroke to trigger the button's action. For example you could set it up so that a key press of e (with or without any modifier keys) would trigger editing of the selected rows.

Keystrokes are only acted upon when there is no other focused element on the page. So if, for example, you did bind a button to the e key, and the user were entering text into the *DataTables* search box, it would not trigger when they press the e key!

The ***buttons.buttons.key*** option can be given as either a string (simple character binding), or as an object which allows modifier keys to also be required.

For example consider:

Example:

```
$('#myTable').DataTable( {
    buttons: [
        {
            extend: 'copy',
            text: '<u>C</u>opy',
            key: {
                key: 'c',
                altKey: true
            }
        }
    ]
} );
```

In this case the copy action is triggered when the **alt key** + **c** key combination is pressed. Note also that the text property has been used to highlight to the end user that the **c** key will do something.

14.6. Built-in buttons

The *Buttons* framework is useful if you wish to define custom buttons, or use those included in another extension such as Editor, but its utility is greatly enhanced by the button types that are included as part of the library.

These buttons are not part of the core library, but rather are individual files that can be included as you require. For example you may wish to provide HTML5 file export buttons but not Flash export buttons. This simply ensures that only the code you require is delivered to the end user.

There are three button types that are part of *Buttons*:

1. File export
2. Print
3. Column visibility

File export

When displaying data in a DataTable, it can often be useful to your end users for them to have the ability to obtain the data from the DataTable, extracting it into a file for them to use locally. This can be done with either HTML5 based buttons or Flash buttons.

Buttons has four built-in buttons types which will automatically detect what the browser abilities are and what software is available - they will automatically use the HTML5 buttons if possible, falling back to Flash and finally not displaying at all if none of the requirements are met:

- **copy** - Copy to clipboard
- **csv** - Save to CSV file
- **excel** - Save to Excel XLSX file
- **pdf** - Save to a PDF document

HTML5

The latest browsers (IE10+ and evergreen browsers) have made great strides in their abilities, and creating local files is something that most can do very well now. As such, there are four buttons types available (it is recommended you use the generic buttons documented above in preference to these buttons):

- **copyHtml5** - Displays a dialogue asking the user to use their browser's copy command (HTML5 does not have a copy to clipboard API)
- **csvHtml5** - Create and save an CSV file
- **excelHtml5** - Create and save an Excel XLSX file - this **requires JSZip**. Note - this will not work in Safari.
- **pdfHtml5** - Create and save a PDF document - this **requires PDFMake** and a suitable font file.

Flash

Older browsers don't have the luxury of being able to create files locally - in order to provide support for these browsers a Flash version of the file export options is available. The advantage of these buttons is that they have no external dependency other than Flash. The disadvantage of these buttons is that they require Flash.

- **copyFlash** - Immediately copies the data to clipboard
- **csvFlash** - Create and save an CSV file
- **excelFlash** - Create and save an Excel XLSX file
- **pdfFlash** - Create and save a PDF document. Note - this does not support UTF8 characters.

Print

Another common way to extract data from a table is to print it. The print button type provides this ability by opening a new window in the user's browser, drawing a table with a copy of the data from the original table.

It then, by default, will automatically trigger the browser's print function allowing the end user to print the table. The window will be closed once the print is complete, or has been cancelled.

Column visibility

While the buttons described above focus on extracting data from the table, the column visibility buttons instead control the display of the table. They provide buttons that can be used to toggle the visibility of individual columns, groups of columns or explicitly set the visibility of columns.

The most commonly used of the available buttons is the **colvis** type. This displays a collection button with a list of all of the columns in the table (this can be customised) with the ability of the end user to toggle the visibility of the columns with a simple click.

14.7. Custom buttons

Buttons own table manipulation modules can be exceptionally useful, but the true flexibility of Buttons can only be unlocked by providing custom buttons which address problems which are unique to the domain you are working in.

For example, consider a table of pupils in class registration. To mark absent pupils a single button could be used along with the Select extension to firstly select missing pupils and then update a database at a single click.

Another example is a button that will reload the data of a DataTable - which we will explore below.

The options of what a button can do are endless; it simply executes a Javascript function that you can define!

Required parameters

When you construct a *Buttons* instance, we've seen already how you can use an object to define each button. While you will typically extend a built in button this is not required.

In fact none of the parameters are actually required - however, you will almost always wish to provide text for the button and an action. After all, without either of these, the button's functionality is rather limited!

- **buttons.buttons.text** - Define the button label
- **buttons.buttons.action** - The action to take when activated

Define during initialisation

Let's expand upon the reload example given above. With an Ajax sourced DataTable (*ajax*) we can use the ***ajax.reload()*** method to reload the data from the server. A button that calls this method provides the end user with the ability to refresh the table's data and can be constructed like this:

Example:

```
$('#myTable').DataTable( {
    ajax: '/api/data'
    buttons: [
        {
            text: 'Reload',
            action: function ( e, dt, node, config ) {
                dt.ajax.reload();
            }
        }
    ]
} );
```

Note that the **buttons.buttons.action** function is passed four parameters. These parameters describe the event that triggered the action, the DataTable hosting the button and the button itself. A full description can be found in the reference documentation.

Custom button type

Defining a custom button, as above, can be very useful for single one-off buttons, but it is also possible to define a reusable button that is extendable in exactly the same way as the pre-defined buttons. This can be done by attaching your button definition object to the **$.fn.dataTable.ext.buttons** object - the parameter name used is the name that will reference the button in the initialisation.

Example: For the reload button above we might use:

```
$.fn.dataTable.ext.buttons.reload = {
    text: 'Reload',
    action: function ( e, dt, node, config ) {
        dt.ajax.reload();
    }
};

$('#myTable').DataTable( {
    ajax: '/api/data',
    buttons: [
        'reload'
    ]
} );
```