16. DATATABLES Responsive Extension.

16.1. Responsive Extension.

In the modern world of **responsive web design** tables can often cause a particular problem for designers due to their row based layout. ***Responsive*** is an extension for DataTables that resolves that problem by optimising the table's layout for different screen sizes through the dynamic insertion and removal of columns from the table.

16.2. Initialization

Responsive can be used on the DataTables in a number of different ways. The simplest of these options is just to add the responsive option to your DataTables options with a boolean value (it is also possible to use an object for fine grained control - see the reference documentation for full details):

Example:

```
$('#myTable').DataTable( {
    responsive: true
} );
```

16.3. Features

Key features include:

- Extend your responsive design to HTML tables
- Full control over column visibility at breakpoints, or automatic visibility
- Collapsed information from the table shown in a child row
- Seamlessly integrates with DataTables
- Works with Bootstrap, Foundation and other responsive CSS frameworks

16.4. Initialisation

Responsive for DataTables can be enabled in three different ways:

- By adding the class **responsive** to the HTML table
- Using the **responsive** DataTable option
- Use the **$.fn.dataTable.Responsive** constructor

Class

To enable *Responsive* on a DataTable through a **class name**, add either:

- **responsive**, or
- **dt-responsive**

to the HTML table's class list. For example:

Example:

```
<table class="display responsive no-wrap" width="100%">
    <thead>
        <tr>
            <th>Name</th>
            <th>Position</th>
            <th>Office</th>
        </tr>
    </thead>
    ...
</table>
```

Responsive will automatically detect a DataTable being created on the page with either of the classes assigned to the table. If the classes are present, Responsive will initialise itself with the default configuration options.

Two class names are provided as options for this automatic initialisation, as the class responsive is relatively common and might already have some defined meaning in your CSS, or the framework if one is being used. For example, in Bootstrap the responsive class is used to perform scrolling of a table (also available in DataTables using the **scrollX** option).

Note that in the example above, the DataTables no-wrap class has also been added to the table. This can be particularly useful for responsive layouts as it disables the wrapping of text content in a table, keeping the text all on one line. Without this class (or use of the white-space CSS option in your own CSS for the table) the browser would attempt to collapse the text in the table onto multiple lines to reduce horizontal space. The result is that the browser window needs to be much smaller before *Responsive* can remove columns.

Option

Responsive can also be enabled for a DataTable using the responsive option. As well as being easy to use in your Javascript, this option also provides the ability to customise the configuration options for Responsive.

In its simplest form, the responsive option can be given as a boolean value to indicate if ***Responsive*** should be enabled or not for that table:

Example:

```
$('#example').DataTable( {
    responsive: true
} );
```

It can also be given as an object, with the properties defining the configuration options to use for the instance. When given as an option *Responsive* is assumed to be enabled for the table (use false if you need to disable it). The full list of options available are documented in the Responsive reference.

In the following example we use responsive as an object in combination with the **responsive.details** option to disable the default behaviour of showing the collapsed details in a child row:

Example:

```
$('#example').DataTable( {
    responsive: {
        details: false
    }
} );
```

Constructor

The final method of initialising *Responsive* for a DataTable is to directly initialise the instance yourself using the **$.fn.dataTable.Responsive** constructor. This is useful if you need to add *Responsive* after a DataTable has been initialised, and you have no ability to add the responsive class before that point.

The constructor takes two parameters:

1. The DataTable to add *Responsive* to. This can be a DataTables API instance, a jQuery selector for the table, or a jQuery object containing the table as the sole node in its result set.
2. Optionally - **configuration parameter**. These are the same options as for responsive. If not given the default configuration options will be used.

In the following example we initialise a DataTable and then add Responsive to it, through use of the constructor method. Please note that you must include the keyword new for the constructor as a new instance must be made for each table.

Example:

```
var table = $('#example').DataTable();
new $.fn.dataTable.Responsive( table, {
    details: false
} );
```

16.5. Column priority

Responsive will automatically hide columns in a table so that the table fits horizontally into the space given to it. This means that a single HTML table can be written and will be displayed nicely for desktop, tablet and mobile web-pages, all without modification of the table.

When Responsive removes columns from the display, it does so, by default, from right to left (i.e. the rightmost column will be removed first). Although this is the default behaviour, you may wish to assign your own column visibility priority to the columns, telling Responsive which columns it should remove before others. This can be particularly useful if, for example, the rightmost column includes actions buttons (edit / delete) or contains particularly important information.

The visibility priority of columns in a Responsive table can be assigned through:

- Use of the **columns.responsivePriority** initialisation option
- A **data-priority option** in the HTML tag for the column header cell.

Priority order

Priority order in *Responsive* is a numerical value, where a lower value equates to a higher priority. For example a column with a priority of 2 will be removed from the display before a column with a priority of 1.

Columns are automatically assigned a priority value of 10,000, which is used unless configured otherwise. If multiple columns share the same priority value, the rightmost will be removed first.

Configuration option

Priority order can be set using the columns.responsivePriority option, which can be defined in the columns or columnDefs arrays of the DataTables configuration. Consider the following:

Example:

```
$('#myTable').DataTable( {
    responsive: true,
    columnDefs: [
        { responsivePriority: 1, targets: 0 },
        { responsivePriority: 2, targets: -1 }
    ]
} );
```

Here the leftmost column (target:0) is assigned a priority of 1, while the rightmost column (target:-1) is assigned a priority of 2. All other columns in the table will use the default priority of 10,000 and thus be removed before either of these two columns. As the table is made narrower, eventually only the first column will be shown.

Priority data attribute

It is also possible to assign the priority **using data-* attributes** in the HTML. This is a particularly convenient way of passing information between the HTML and Javascript since no additional configuration options need be specified. Simple add a data-priority attribute to the column header cells, with the value denoting the priority to assign for the column.

Consider the following table:

Example:

```
<table id="example" class="display nowrap" width="100%">
    <thead>
        <tr>
            <th data-priority="1">First name</th>
            <th>Last name</th>
            <th>Position</th>
            <th>Office</th>
            <th>Age</th>
            <th>Start date</th>
            <th>Salary</th>
            <th>E-mail</th>
            <th data-priority="2">Extn.</th>
        </tr>
    </thead>
    <tbody>
        ...
    </tbody>
</table>
```

This exactly matches the Javascript example given above - the leftmost column has priority 1, the rightmost priority 2 and all the others a default priority.

16.6. Class logic

Responsive has three modes of operation for controlling the visibility of columns in a table:

- **Automatic** - whereby Responsive will automatically determine if a column should be shown or not
- **Manual** - where you tell Responsive what columns to show on what screen sizes.
- **Priority** - Using columns.responsivePriority to tell Responsive which columns should be given visibility priority - see priority documentation - (2.0.0).

These three modes are not mutually exclusive in a table, columns of all three types can be used in a single table.

Manual mode is triggered by assigning a specific class to a column - one which matches a set of logic rules for the breakpoints defined in Responsive. This is fully detailed below. The class can be added

either to the plain HTML table before the DataTable is initialised, or by using the DataTables **columns.className** option.

If Responsive does not detect a class name for a column that matches a breakpoint, or one of the special cases noted below (all, none and control), the column visibility will be controlled automatically with the column priority defining the hiding order.

Breakpoints

To define which columns are displayed on screen for different screen sizes, *Responsive* has the concept of **breakpoints**. A breakpoint is a width, in pixels, of the browser viewport at which a breakpoint is activated. For example, a tablet breakpoint might be activated at 1024 pixels while a mobile phone breakpoint would be 480 pixels. By naming these breakpoints we can add classes to the table to control if a column is visible at those breakpoints. As an example, if a column has the class tablet it will be visible on tablet devices, but not mobile or desktop browsers.

Responsive has five breakpoints built in:

Name	Width (x) range
`desktop`	x > 1024
`tablet-l` (landscape)	768 < x <= 1024
`tablet-p` (portrait)	480 < x <= 768
`mobile-l` (landscape)	320 < x <= 768
`mobile-p` (portrait)	x <= 320

Breakpoints can be customised using the responsive.breakpoints initialisation option, or by modifying the ***$.fn.dataTable.Responsive.breakpoints*** array which contains the default breakpoints.

Special classes

In addition to detecting classes based upon class name, Responsive will also detect the following special classes:

- **all** - Column will always be visible, regardless of the browser width
- **none** - Column will never be visible, regardless of the browser width, but the data will be shown in a child row
- **never** - Column will never be visible, regardless of the browser width, and the data will not be shown in a child row (1.0.2). Note as of Responsive 2.0.0 this option is no longer required as v2 fully supports DataTables' native column visibility (i.e. use columns.visible or column().visible() to hide a column).

- **control** - This is a special class which is used by the column option for the responsive.details.type option to designate which column is the control column in the table. This allows the Responsive stylesheet to add the required styling information for the column.

These classes can be used on any column and will take precedent over a breakpoint class.

Breakpoint logic operations

Using the full breakpoint names you can specify exactly which columns are to be shown at each breakpoint, but this can be quite verbose. For example, if you wanted to specify a column should be visible at all breakpoints but the desktop breakpoint, four classes would be needed. You can do that, but to make complex combinations more succinct Responsive provides the ability to add logic operations to a class name for a breakpoint:

Prefix:

- **not-** - Visible in all but the named breakpoint. For example not-desktop would be visible at mobile and tablet widths only.
- **min-** - Visible in breakpoints greater or equal in width to the named breakpoint. For example min-tablet-l would be visible at tablet-l and desktop widths only.
- **max-** - Visible in breakpoints less than or equal in width to the named breakpoint. For example min-tablet-p is visible at the tablet-p and mobile breakpoints.

Postfix:
The ***-l and -p options*** to designate the landscape and portrait breakpoints for tablets and mobiles are optional. This makes it easier to specify a column's visiblity for a device group. For example tablet will be visible at the **tablet-l** and **tablet-p** breakpoints.
The ***postfix*** (or rather, the lack of a postfix) and also be used in combination with the prefix logic operations. For example min-tablet will be visible in the tablet and mobile breakpoints.

Logic reference

The following table lists the class names that *Responsive* will automatically detect.

Class name	Breakpoints included in
`desktop`	`desktop`
`not-desktop`	`tablet-l`, `tablet-p`, `mobile-l`, `mobile-p`
`min-desktop`	`desktop`
`max-desktop`	`desktop`, `tablet-l`, `tablet-p`, `mobile-l`, `mobile-p`
`tablet`	`tablet-l`, `tablet-p`
`not-tablet`	`desktop`, `mobile-l`, `mobile-p`
`min-tablet`	`desktop`, `tablet-l`, `tablet-p`

Class name	Breakpoints included in
`max-tablet`	`tablet-l`, `tablet-p`, `mobile-l`, `mobile-p`
`tablet-l`	`tablet-l`
`not-tablet-l`	`desktop`, `tablet-p`, `mobile-l`, `mobile-p`
`min-tablet-l`	`desktop`, `tablet-l`
`max-tablet-l`	`tablet-l`, `tablet-p`, `mobile-l`, `mobile-p`
`tablet-p`	`tablet-p`
`not-tablet-p`	`desktop`, `tablet-l`, `mobile-l`, `mobile-p`
`min-tablet-p`	`desktop`, `tablet-l`, `tablet-p`
`max-tablet-p`	`tablet-p`, `mobile-l`, `mobile-p`
`mobile`	`mobile-l`, `mobile-p`
`not-mobile`	`desktop`, `tablet-l`, `tablet-p`
`min-mobile`	`desktop`, `tablet-l`, `tablet-p`, `mobile-l`, `mobile-p`
`max-mobile`	`mobile-l`, `mobile-p`
`mobile-l`	`mobile-l`
`not-mobile-l`	`desktop`, `tablet-p`, `tablet-l`, `mobile-p`
`min-mobile-l`	`desktop`, `tablet-l`, `tablet-p`, `mobile-l`
`max-mobile-l`	`mobile-l`, `mobile-p`
`mobile-p`	`mobile-p`
`not-mobile-p`	`desktop`, `tablet-l`, `tablet-p`, `mobile-l`
`min-mobile-p`	`desktop`, `tablet-l`, `tablet-p`, `mobile-l`, `mobile-p`
`max-mobile-p`	`mobile-p`

16.7. Details views

On narrow screens when *Responsive* removes columns from the display, the data in the hidden columns might still be useful and relevant to the end user. For this, Responsive has the ability to display the information from a row in a variety of ways including using DataTables' child row feature (***row().child()***) or a modal pop-up. This is also extensible and custom methods can be defined if required.

Display methods

The method by which a row's information can be displayed is controlled by the ***responsive.details.display*** option. This should be set to a function that will display the row's information (based on the rendered data - see below). Responsive's built in display options are available in the ***$.fn.dataTable.Responsive.display*** object - they are:

- **childRow** - Show hidden information in a child row, the visibility of which can be toggled by an end user.
- **childRowImmediate** - Immediately show information in a child row, without waiting for a user request ***modal***() - Show information in a modal pop-up - please note that this is a function and should be executed when being assigned - this is to allow options to be passed into the modal library being used (see the **responsive.details.display** reference documentation for details). When Responsive is used with Bootstrap, Foundation or jQuery UI it will automatically use their native modal dialogue libraries.

For example, you could use the child row with immediate show control using the following initialisation:

Example:

```
$('#myTable').DataTable( {
    responsive: {
        details: {
            display:
$.fn.dataTable.Responsive.display.childRowImmediate
        }
    }
} );
```

The **childRow display method** is the default option that *Responsive* will use. It uses a child row to display the data hidden by *Responsive*. To keep the screen uncluttered, by default the child row is not visible but the end user can select to view the data if they so wish.

For information on how to create a custom renderer, please see the **responsive.details.display** reference documentation. If you create one, let us know!

Rendering options

Responsive has a built in **renderer** that will display the data for the hidden columns in the details rows for the table. This is an ul/li list that contains the title of the column for the data and the data itself. The renderer is re-executed every time the column visibility changes (a desktop browser resize for example), ensuring that the full data for the table is retained.

A **custom renderer** can be defined using the **responsive.details.renderer** option if you wish to control the layout yourself, or potentially add additional information into the details row.

The **display** methods discussed above will utilise the data that is returned by the renderer, allowing the rendered information to be entirely independent of how it is displayed (i.e. it will work for both child rows and modals).

User interaction controls

Responsive has three options for how to display the controls allowing the end user to access the details row which are set through the **responsive.details.type** option:

1. **Disabled** (*false*) - no ability to show the details row
2. **Inline** (*inline* - default) - the control will be shown in the first column of the table if there are hidden columns
3. **Column based** (*column*) - the control will be shown in the designated column, with that column being used only for the purpose of showing the details row access control.

The ***inline*** type is useful for cases when you wish to add *Responsive* to an existing table without changing its structure. The column type is useful for cases when you don't wish to have the layout of the first column altered by the control icon being shown in the first column, as it is for inline.