07. DATATABLES API.

7.1. API

DataTables and its *extensions* have an extensive API which can be used to access the data contained in the table and otherwise manipulate the table after initialisation has completed. The **DataTables API is designed to reflect the structure of the data in the table**, and the ways that you will typically want to **interact with the table**. As such the API is composed of six key areas for working with the table and its data:

- Tables
- Columns
- Rows
- Cells
- Core
- Utilities

Extensions and **plug-ins** may also add additional methods to the API exposing the features and functionality that they add to the DataTable.

- https://datatables.net/extensions
- https://datatables.net/manual/plug-ins/api

For a full list of the methods that are available in the API, please refer to the API reference section.

- https://datatables.net/reference/api

7.2. Terminology

In order to keep the documentation of the API succinct, a number of terms are used to describe, or refer to, aspects of the API. These are defined here:

- **Instance** - an *instance* of the API is a single object instance which refers to a specific set of DataTable tables (i.e. is the the result of new Api()).

- **Result set** - the data held by the API instance. DataTables API instances are "*array like*" in that they hold data in the same way as a plain Javascript array (*and thus can be accessed using array [] notation*) and have many of the same methods available (**sort**() and **push**() for example). This is much in the same way as jQuery is array like in nature.

- **Context** - the *context* of an API instance describes the DataTables that the API instance has a link to. Each API instance can refer to one or more DataTables tables, with the ability to act upon those tables.

7.3. Accessing the API

A new DataTables API instance can be obtained for one or more tables in one of three different ways:

Example:

```
$( selector ).DataTable();
$( selector ).dataTable().api();
new $.fn.dataTable.Api( selector );
```

The result from each is an instance of the DataTables API object which has the tables found by the selector in its context.

It is important to note the difference between **$(selector).DataTable()** and **$(selector).dataTable()**. The former returns a **DataTables API instance**, while the latter returns a **jQuery object**. An **api**() method is added to the jQuery object so you can easily access the API, but the jQuery object can be useful for manipulating the table node, as you would with any other jQuery instance (such as using *addClass*(), etc.).

Upgrade note:

The DataTables API in v1.9 and earlier was be accessed using **$().dataTable().method()**. This was done by extending the jQuery object with the DataTables API methods.

The old API is still available in DataTables 1.10 for backwards compatibility, but the new API is now preferred as it presents much greater flexibility and improved functionality.

7.4. Chaining

Like jQuery, the DataTables API makes extensive use of **chaining**, with many, but not all, of the DataTables methods returning the API instance itself, so you can immediately call another API method. For example:

Example:

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

table.search( 'Fiona' ).draw();
```

This makes use of two different API methods, **search**() and **draw**(), on a single line. It could equally be written as:

Example:

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

table.search( 'Fiona' );
table.draw();
```

The functionality is **identical** in this case, but chaining can allow for more succinct and expressive code.

Where the DataTables API departs from jQuery's chaining method is that **DataTables makes use of nested methods and properties**. For example the *ajax.json*() method gives you access to the latest JSON data from an Ajax call DataTables has made - in this case the *json*() method is a child of the ajax property. Likewise, the *columns*() (and other data manipulation methods) provide their own chaining child methods. For example *columns().visible()*. This allows the API to be very expressive in terms of how you access data, with methods relating to what has been called before.

All top level methods of the API will always be available, at all levels of nesting of the API. For example *draw*() is a top level method, but it can be called after a row is removed from the table:

Example:

```
table.row( selector ).remove().draw();
```

Please note that not all methods will return an API instance for chaining. In some cases, returning the API instance for chaining wouldn't be appropriate, such as *cell().node()* to get a **td / th element**. The API reference documentation contains full details for each method and what it will return.

https://datatables.net/reference/api

7.5. Multiple Tables

DataTables API instances can refer to **multiple tables** in their context. The API treats each table in a context as the same. For example:

Example:

```
var tables = $('.dataTable').DataTable();

tables.page( 'next' ).draw( false );
```

This code will select all tables in the document with the class dataTable and jump them all to their next page of data display (using *page()*).

Equally, an API instance can refer to a single table if required, simply by altering the jQuery selector used to create the API instance: ***$('#myTable').DataTable();*** will create an API instance with a single table in its context for example.

Example:

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

7.6. Plural / Singular

When working with the API you will notice that the methods make extensive use of plural and singular terminology. Although this is relatively unusual in an API, it is done to reflect the way that you access the data in the table - for example **rows().data()** will return an API instance with the data for the **selected rows** in its **result set** (*i.e. like an array*), while **row().data()** will return the data for just that **row**. This makes working with the API much more intuitive as you will always get the result you expect.

To be clear, in English (for our international users):

- Singular === 1
- Plural > 1

So if you want to use one of the selector methods to select multiple items, use the plural form of the method. If you want to select a single specific item, use the singular form.

7.7. Example - column filter

API reference documentation contains an example for each API method, along with a detailed description of what it does, what it returns and what parameters it will accept; but let's form the concepts described above into a detailed, line-by-line example of how the DataTables API can be used. In this case we create a **select** element in the footer cell of each column in the table, which contains the data from that column and will be used for searching the table.

Example:

```
 1   var table = $('#example').DataTable();
 2
 3   table.columns().flatten().each( function ( colIdx ) {
 4
 5   // Create the select list and search operation
 6   var select = $('<select />')
 7           .appendTo(
 8               table.column(colIdx).footer()
 9           )
10           .on( 'change', function () {
11               table
12                   .column( colIdx )
13                   .search( $(this).val() )
14                   .draw();
15           } );
16
17 // Get the search data for the 1st column & add to the select list
18
19     table
20          .column( colIdx )
21          .cache( 'search' )
22          .sort()
23          .unique()
24          .each( function ( d ) {
25              select.append( $('<option value="'+d+'">'+d+'</option>') );
26          } );
27   } );
```

- Line 1 - Get an **instance of the DataTables API** with a single table in its context.

- Line 3 - **Select all columns** in the table with the ***columns***() method. ***flatten***() is used to reduce the 2D array return of *columns*() to a 1D array of column indexes, and the utility method ***each***() is used to perform an action on each of the selected columns.

- Line 6 - The **select** element for the column's filter is created.

- Line 7 - Using the ***column().footer()*** method to get the **footer cell** element into which to insert the select input.

- Line 10 - Use the jQuery **on**() method to perform an action when the select element's value changes - in this case searching the table.

- Lines 11 - 14 - Use **column().search()** to queue up the search, and **draw()** chaining to the result to update the table's display.

- Lines 18 - 20 - Get the data from the column that DataTables uses to search the table using the **column().cache()** method.

- Lines 21 - 22 - The utility methods **sort()** and **unique()** are used to reduce the data to a nicely ordered list that you would want to present to your end users.

- Lines 23 - 25 - A search term option is added to our select list, ready for use.

As you can see, the DataTables API is extremely flexible, and provides a wide range of options for accessing and manipulating the table. Please see the API reference documentation for a full list of the methods that are available. Furthermore, plug-ins such as Editor can extend the API with custom methods such as **row().edit()** and **cell().edit()** among other options.