02. DATATABLES Data.

2.1. Data

Data is complex, and all data is different. Accordingly, DataTables has a wealth of options which can be used to configure how it will obtain the data to display in the table, and how it processes that data.

There are three core concepts in how DataTables handles data, which are discussed in detail on this page:

- Processing mode
- Data types
- Data sources

2.2. Processing Modes

DataTables has two different modes of processing data (ordering, searching, etc. of data):

- **Client-side processing** - the full data set is loaded up-front and data processing is done in the browser.
- **Server-side processing** - an Ajax request is made for every table redraw, with only the data required for each display returned. The data processing is performed on the server.

Each has its own advantages and disadvantages, but the key indicator for which mode you should select is based on the number of rows in your table. As a rule of thumb, if you are working with less than 10,000 rows use client-side processing, for greater than 100,000 rows use server-side processing. In-between is a grey area where you will need to make a decision based upon the nature of your app and the data you wish to display!

Please note that the two processing modes are mutually exclusive - they cannot be used at the same time, nor is it possible to dynamically change from one mode to the other.

Client-side processing

Client-side processing is the default operating mode of DataTables as it is easy to use and requires no additional code to be written. In client-side processing mode, the ordering of the data in the table, searching, paging and all other data processing operations that DataTables performs are done in the browser by DataTables itself.

- Full dataset is loaded up-front
- Data can be read from DOM, JS data source or via Ajax
- Data processing is performed in the browser
- For small to medium data sets

Points to consider:

- Time to download the full dataset
- Complex ordering on large data sets can slow the browser down

Server-side processing

Where server-side processing comes into play is when you have very large quantities of data that you wish to display in the table (millions of rows for example). At these levels, sending the data to the client, and then having Javascript process the data can involve noticeable overhead, and may result in poor performance of the end application. In server-side processing mode, all ordering, searching, etc. of the data is handed off to the server which can make use of the database engines available there, which are highly tuned for exactly these kinds of operations. Each page of data (referred to as a draw in DataTables terminology) involves making an Ajax request to the server. Although each Ajax request might take a fraction of a second to complete, this approach might be preferable to a large wait up-front as all data loads.

- Ajax request made for every redraw
- Only the data needed is loaded
- Server performs the processing
- Ideal for large data sets

Points to consider:

- Latency of Ajax requests
- Can your server handle the load of lots of requests?

Server-side processing is enabled by the **serverSide** option, and full documentation for how server-side processing operates is available in its section of this manual. https://datatables.net/manual/server-side

2.3. Data source types

The main data source used for a DataTable must always be an array (it is created automatically when using DOM sourced data). Each item in that array will define a row to be displayed and DataTables can use three basic Javascript data types as the data source for the rows:

- **Arrays** - []
- **Objects** - {}
- **Instances** - new MyClass()

DataTables can consume data from any of these options using the **columns.data** and **columns.render** options. The default mode of operation is an array, but objects and instances can be useful as they are typically more intuitive when working with complex data.

Arrays

Arrays are easy to work with in a DataTable as the mapping between array elements to the column the data appears in is performed simply by the column index reading the array element value in that position. For example, the first table column maps to the first array element for the row's data source, etc.

Due to this, when using arrays as your data source, the number of elements in each array must be equal to the number of columns in the table. For example, for a 6 column table you might have:

Example:

```
var data = [
    [
        "Tiger Nixon",
        "System Architect",
        "Edinburgh",
        "5421",
        "2011/04/25",
        "$3,120"
    ],
    [
        "Garrett Winters",
        "Director",
        "Edinburgh",
        "8422",
        "2011/07/25",
        "$5,300"
    ]
]
```

And the table initialisation:

Example:

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

Which would result in a table such as:

Name	Position	Office	Extn.	Start date	Salary
Tiger Nixon	System Architect	Edinburgh	5421	2011/04/25	$3,120
Garrett Winters	Director	Edinburgh	8422	2011/07/25	$5,300

Objects

Objects are great for intuitive use in a slightly different way from arrays. If you are actively working with the data through the API, objects can make obtaining a particular piece of data very easy as you need only use a property name, rather than remembering which array index that data is in (for example **data.name**, rather than **data[0]**).

Objects can also contain more information than is required for the DataTable display, which can be very useful for operating on the data (for example including a database primary key which is not visible to the end user).

The down side of using objects is that you need to explicitly tell DataTables which property it should use from the object for each column. This is done using the **columns.data** and / or **columns.render** options.

Object based data may look like:

Example:

```
[
    {
        "name":       "Tiger Nixon",
        "position":   "System Architect",
        "salary":     "$3,120",
        "start_date": "2011/04/25",
        "office":     "Edinburgh",
        "extn":       "5421"
    },
    {
        "name":       "Garrett Winters",
        "position":   "Director",
        "salary":     "$5,300",
        "start_date": "2011/07/25",
        "office":     "Edinburgh",
        "extn":       "8422"
    }
```

```
    ]
```

And the table initialisation:

Example:

```
$('#example').DataTable( {
    data: data,
    columns: [
        { data: 'name' },
        { data: 'position' },
        { data: 'salary' },
        { data: 'office' }
    ]
} );
```

Note that only 4 columns are defined and that data displayed in each column is easily defined by changing where each property is used, rather than needing to reorder the source objects as would be the case with an array data source. This would result in a table such as:

Name	Position	Salary	Office
Tiger Nixon	System Architect	$3,120	Edinburgh
Garrett Winters	Director	$5,300	Edinburgh

Instances

It can be quite useful to have DataTables display information from Javascript object instances, as these instances define abstract methods which can be used to update data. For example you might have an Employee class, or a Car class, etc. depending on the data you are modelling. Instances can be used in DataTables in much the same way as objects - simply pass in your object and give the method or property name to **columns.data** for the data for each column.

Example:

```
function Employee ( name, position, salary, office ) {
    this.name = name;
    this.position = position;
    this.salary = salary;
    this._office = office;
```

```
        this.office = function () {
            return this._office;
        }
    };

    $('#example').DataTable( {
        data: [
            new Employee( "Tiger Nixon", "System Architect", "$3,120",
    "Edinburgh" ),
            new Employee( "Garrett Winters", "Director", "$5,300",
    "Edinburgh" )
        ],
        columns: [
            { data: 'name' },
            { data: 'salary' },
            { data: 'office' },
            { data: 'position' }
        ]
    } );
```

Note that office is a method of the class above, while name, position and salary are properties. DataTables will automatically realise that there is a function, execute it and use the returned value for the cell (note you could also use the syntax office() to be explicit that a function is used - see **columns.data** for further information).

Like the object based data source method above, although in this case the Position column is shown at the end of the table to show that the properties can be read in any order, this would result in a table such as:

Name	Salary	Office	Position
Tiger Nixon	$3,120	Edinburgh	System Architect
Garrett Winters	$5,300	Edinburgh	Director

2.4. Data Sources

With the concepts of the processing mode and data types now defined, we can consider how DataTables actually gets the data it is to operate with. There are three basic sources for the data that DataTables will display in the table:

- **DOM** (i.e. the plain HTML of the document)
- **Javascript**
- **Ajax** sourced data

DOM

When DataTables starts up, it will automatically check the table it is operating on for data that already exists inside it and use it for the table (note that it will throw this data away if you pass in data using **data** or **ajax** to get new data!). This is the simplest method of using DataTables - working with a regular HTML table.

Note that when using a DOM sourced table, DataTables will use arrays as the data source (see above) by default, although you could use the **columns.data** option to have it construct objects for the row data instead.

HTML5

DataTables can also make use of HTML5 d**ata-*** attributes, which can provide DataTables with additional information for ordering and search data. For example you might have a column with a date formatted such as "21st November 2016". Browsers will struggle to sort that, but you could provide a **data-order** attribute as part of the HTML for the cell which contains a timestamp that can be easily sorted upon. Extending that, search data can be provided using **data-search**. For example:

Example:

```
<td data-search="21st November 2016 21/11/2016" data-
order="1479686400">
      21st November 2016
  </td>
```

DataTables will automatically detect:

- **Ordering data:** data-order and data-sort attributes
- **Search data**: data-search and data-filter attributes.

See the manual entry for orthogonal data for further information.

Javascript

You can instruct DataTables to use data given to it using the **data** initialisation option. This data can be in the form of arrays, objects or instances (see above) and can be sourced from anywhere you want! As long as Javascript can access the data, you can send it to DataTables (be it from a custom Ajax call, a

WebSocket or just a good old fashioned array of data).

This method can be particularly useful when working extensively with the DataTables API, in particular the **row.add()** and **row().remove()** methods can be used to add and remove data from the table dynamically, again from whatever source you wish to choose.

Ajax

Ajax sourced data is much like Javascript sourced data, but DataTables will make an Ajax call to get the data for you. It can often be very useful to source table data from a specific script, separating the logic for retrieving the data from the display. Ajax sourced data in DataTables is controlled by the **ajax** option. In its simplest form, you set the property value as a string, pointing at the URL you want to load data from.

Like Javascript sourced data, Ajax sourced data can be in the form of objects or arrays (see above), but not, in this case, instances (since they cannot be represented in JSON).

Server-side processing as discussed above is a special form of Ajax sourced data, whereby the data to be shown for each page in the DataTable is retrieved by an Ajax request only when that page is required for display to the user. This allows the power of the database engine on servers to be utilised for large data sets. For more information about server-side processing, and how it can be implemented, please refer to its documentation in this manual. https://datatables.net/manual/server-side

Ajax loading of data is discussed in detail in the next section of this manual.
https://datatables.net/manual/ajax

2.5. Where to go next

Data is of course at the core of DataTables and it is impossible to cover the topic fully in a single page. This page gives a general overview of how DataTables handles data, but other topics include:

- Orthogonal data
- Renderers
- Ajax sourced data
- API access to the data.