05. DATATABLES Ajax.

5.1. Ajax

Ajax data is loaded by DataTables simply by using the ajax option to set the URL for where the Ajax request should be made. For example, the following shows a minimal configuration with Ajax sourced data:

Example:

```
$('#myTable').DataTable( {
    ajax: '/api/myData'
} );
```

5.2. JSON Data Source

When considering Ajax loaded data for DataTables we almost always are referring to a **JSON** payload - i.e. the data that is returned from the server is in a JSON data structure. This is because the JSON is derived from Javascript and it therefore naturally plays well with Javascript libraries such as DataTables. It is also a compact and easily understood data format which has proven to be very popular in the Javascript world.

It is possible to use **other data formats such as XML and YAML** with DataTables, although these formats need to be converted to Javascript object notation (i.e. JSON) before they are using - this is typically done in using **ajax.dataSrc**. The remainder of this document will consider only JSON.

With our JSON data source we need two key pieces of information:

- Where the array of data that represents the rows of data in the table is in the object
- Where the data point for each column is in the row object / array.

5.3. Data array location

DataTables requires an **array of items** to represent the table's data, where each item in the array is a row. The **item** is typically an object or an array (discussed in more detail below) - so the first thing we need to do is tell DataTables where that array is in the data source.

Consider, for example, the following three JSON data objects shown on the left below, as you will be able to see each of the three structures contain the same data for the array of data to be displayed in the

table, but the location of that array if different in each. Each is perfectly valid and can be used in different circumstances - there is no single *"correct way"*!

The **ajax.dataSrc** (i.e. data source) option is used to tell DataTables where the data array is in the JSON structure. **ajax.dataSrc** is typically given as a string indicating that location in Javascript object notation - i.e. simply set it to be the name of the property where the array is! An empty string is a special case which tells DataTables to expect an array (as in the first example above).

The three data structures are each shown with their corresponding DataTables initialisations.

1) **Simple array of data**.

2) **Object with data property** - note that the data parameter format shown here can be used with a simplified DataTables initialisation as data is the default property that DataTables looks for in the source data object.

3) **Object with staff property**.

Example: Simple array of data (1):

```
[
    {
        "name": "Tiger Nixon",
        "position": "System Architect",
        "salary": "$320,800",
        "start_date": "2011/04/25",
        "office": "Edinburgh",
        "extn": "5421"
    },
    ...
}
```

```
$('#myTable').DataTable( {
    ajax: {
        url: '/api/myData',
        dataSrc: ''
    },
    columns: [ ... ]
} );
```

Example: Object with data property (2):

```
{
```

```
    "data": [
        {
            "name": "Tiger Nixon",
            "position": "System Architect",
            "salary": "$320,800",
            "start_date": "2011/04/25",
            "office": "Edinburgh",
            "extn": "5421"
        },
        ...
    ]
}
```

```
$('#myTable').DataTable( {
    ajax: '/api/myData',
    columns: [ ... ]
} );

// or!

$('#myTable').DataTable( {
    ajax: {
        url: '/api/myData',
        dataSrc: 'data'
    },
    columns: [ ... ]
} );
```

Example: Object with staff property (3):

```
{
    "staff": [
        {
            "name": "Tiger Nixon",
            "position": "System Architect",
            "salary": "$320,800",
            "start_date": "2011/04/25",
            "office": "Edinburgh",
            "extn": "5421"
        },
        ...
    ]
}
```

```
$('#myTable').DataTable( {
    ajax: {
        url: '/api/myData',
```

```
            dataSrc: 'staff'
        },
        columns: [ ... ]
    } );
```

5.4. Column data points

Now that DataTables knows where to get the data for the rows, we need to also tell it where to get the data for each cell in that row - this is done through the **columns.data** option.

Let's consider again three different data formats, again shown on the left below - only a single row of data is shown in each case (*i.e. it is not wrapped in a structure as discussed above for brevity*).

As you will be able to see, in each of the three cases, the same data is available for the row, but the structure of the JSON data is different. We use the *columns.data* property to tell DataTables where to get the data for each column.

Like the **ajax.dataSrc** option discussed above, **columns.data** is typically given as a string that represents the location of the data required in Javascript dotted object notation. It can also be given in other forms such as an index for accessing an array.

The corresponding initialisation of DataTables for the three data structures is shown on the right.

1) Array of data - note that the array option does not require the *columns.data* option to be set. This is because the default value for columns.data is the column index (i.e. 0, 1, 2…).

2) Object of data.

3) Nested objects - in this case note that in the nested objects we use dotted object notation such as *hr.position* to access nested data. With this ability almost any JSON data structure can be used with DataTables.

Example: Array of data (1).

```
[
    "Tiger Nixon",
    "System Architect",
    "$320,800",
    "2011/04/25",
    "Edinburgh",
    "5421"
]
```

```
$('#myTable').DataTable( {
```

```
        ajax: ...
    } );

    // or!

    $('#myTable').DataTable( {
        ajax: ...,
        columns: [
            { data: 0 },
            { data: 1 },
            { data: 2 },
            { data: 3 },
            { data: 4 },
            { data: 5 }
        ]
    } );
```

Example: Object of data (2).

```
{
    "name": "Tiger Nixon",
    "position": "System Architect",
    "salary": "$320,800",
    "start_date": "2011/04/25",
    "office": "Edinburgh",
    "extn": "5421"
}
```

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

Example: Nested objects (3).

```
{
    "name": "Tiger Nixon",
    "hr": {
        "position": "System Architect",
        "salary": "$320,800",
        "start_date": "2011/04/25"
```

```
        },
        "contact": {
            "office": "Edinburgh",
            "extn": "5421"
        }
    }
```

```
    $('#myTable').DataTable( {
        ajax: ...,
        columns: [
            { data: 'name' },
            { data: 'hr.position' },
            { data: 'hr.salary' },
            { data: 'hr.state_date' },
            { data: 'contact.office' },
            { data: 'contact.extn' }
        ]
    } );
```

5.5. Ajax configuration

At this time **DataTables unfortunately does not support configuration via Ajax**. This is something that will be reviewed in future, however, JSON does not provide an option to represent Javascript functions which can be exceptionally useful in a DataTables configuration, therefore Ajax loaded configuration could not use all of the options DataTables makes available.