10. DATATABLES Server side Processing.

10.1. Server side Processing

There are times when reading data from the DOM is simply too slow or unwieldy, particularly when dealing with many thousands or millions of data rows. To address this DataTables' server-side processing feature provides a method to let all the "heavy lifting" be done by a database engine on the server-side, and then have that information drawn in the user's web-browser. Consequently, you can display tables consisting of millions of rows with ease.

When using server-side processing, DataTables will make an Ajax request to the server for each draw of the information on the page (*i.e. when paging, ordering, searching, etc.*). DataTables will send a number of variables to the server to allow it to perform the required processing and then return the data in the format required by DataTables.

Server-side processing is enabled by use of the **serverSide** option, and configured using **ajax**.

10.2. Sent parameters

When making a request to the server using server-side processing, DataTables will send the following data in order to let the server know what data is required:

Parameter name	Type	Description
`draw`	`integer`	Draw counter. This is used by DataTables to ensure that the Ajax returns from server-side processing requests are drawn in sequence by DataTables (Ajax requests are asynchronous and thus can return out of sequence). This is used as part of the `draw` return parameter (see below).
`start`	`integer`	Paging first record indicator. This is the start point in the current data set (0 index based - i.e. 0 is the first record).
`length`	`integer`	Number of records that the table can display in the current draw. It is expected that the number of records returned will be equal to this number, unless the server has fewer records to return. Note that this can be -1 to indicate that all records should be returned (although that negates any benefits of server-side processing!)
`search[value]`	`string`	Global search value. To be applied to all columns which have `searchable` as `true`.
`search[regex]`	`boolean`	`true` if the global filter should be treated as a regular expression for advanced searching, `false` otherwise. Note that normally server-side processing scripts will not perform regular expression searching for performance reasons on large data sets, but it is technically possible and at the discretion of your script.

Parameter name	Type	Description
`order[i][column]`	`integer`	Column to which ordering should be applied. This is an index reference to the `columns` array of information that is also submitted to the server.
`order[i][dir]`	`string`	Ordering direction for this column. It will be `asc` or `desc` to indicate ascending ordering or descending ordering, respectively.
`columns[i][data]`	`string`	Column's data source, as defined by `columns.data`.
`columns[i][name]`	`string`	Column's name, as defined by `columns.name`.
`columns[i][searchable]`	`boolean`	Flag to indicate if this column is searchable (`true`) or not (`false`). This is controlled by `columns.searchable`.
`columns[i][orderable]`	`boolean`	Flag to indicate if this column is orderable (`true`) or not (`false`). This is controlled by `columns.orderable`.
`columns[i][search][value]`	`string`	Search value to apply to this specific column.
`columns[i][search][regex]`	`boolean`	Flag to indicate if the search term for this column should be treated as regular expression (`true`) or not (`false`). As with global search, normally server-side processing scripts will not perform regular expression searching for performance reasons on large data sets, but it is technically possible and at the discretion of your script.

The **order[i]** and **columns[i]** parameters that are sent to the server are arrays of information:

- **order[i]** - is an array defining how many columns are being ordered upon - i.e. if the array length is 1, then a single column sort is being performed, otherwise a multi-column sort is being performed.

- **columns[i]** - an array defining all columns in the table.

In both cases, i is an integer which will change to indicate the array value. In most modern server-side scripting environments this data will automatically be available to you as an array.

10.3. Returned data

Once DataTables has made a request for data, with the above parameters sent to the server, it expects **JSON data** to be returned to it, with the following parameters set:

Parameter name	Type	Description
`draw`	`integer`	The draw counter that this object is a response to - from the `draw` parameter sent as part of the data request. Note that it is **strongly recommended for security reasons** that you *cast* this parameter to an integer, rather than simply echoing back to the client what it sent in the `draw` parameter, in order to prevent Cross Site Scripting (XSS) attacks.
`recordsTotal`	`integer`	Total records, before filtering (i.e. the total number of records in the database)
`recordsFiltered`	`integer`	Total records, after filtering (i.e. the total number of records after filtering has been applied - not just the number of records being returned for this page of data).
`data`	`array`	The data to be displayed in the table. This is an array of data source objects, one for each row, which will be used by DataTables. Note that this parameter's name can be changed using the `ajax` option's `dataSrc` property.
`error`	`string`	*Optional*: If an error occurs during the running of the server-side processing script, you can inform the user of this error by passing back the error message to be displayed using this parameter. Do not include if there is no error.

In addition to the above parameters which control the overall table, DataTables can use the following optional parameters on each individual row's data source object to perform automatic actions for you:

Parameter name	Type	Description
`DT_RowId`	`string`	Set the ID property of the `tr` node to this value
`DT_RowClass`	`string`	Add this class to the `tr` node
`DT_RowData`	`object`	Add the data contained in the object to the row using the jQuery `data()` method to set the data, which can also then be used for later retrieval (for example on a click event).
`DT_RowAttr`	`object`	Add the data contained in the object to the row `tr` node as attributes. The object keys are used as the attribute keys and the values as the corresponding attribute values. This is performed using using the jQuery `param()` method.

10.4. Configuration

Server-side processing in DataTables is enabled through use of the **serverSide** option. Simply set it to true and DataTables will operate in server-side processing mode. You will also want to use the **ajax** option to specify the URL where DataTables should get its Ajax data from. As such, the simplest server-side processing initialisation is:

Example:

```
$('#example').DataTable( {
    serverSide: true,
    ajax: '/data-source'
} );
```

Configuration of how DataTables makes the Ajax request is configured through the **ajax** option. In the above example we used it as a string, which instructs DataTables to use its default settings for making the Ajax request. However, you can customise these settings by passing ajax in as an object. As an object, ajax maps directly onto the jQuery a**jax configuration object**, so any options you can use in a jQuery request, you can also use with DataTables! For example, to make a **POST request**:

Example:

```
$('#example').DataTable( {
    serverSide: true,
    ajax: {
        url: '/data-source',
        type: 'POST'
    }
} );
```

For further information about the Ajax options available in DataTables, refer to the ajax documentation. https://datatables.net/reference/option/ajax

10.5. Example

Example of server-side processing return using arrays as the data source for the table:

Example:

```
{
    "draw": 1,
    "recordsTotal": 57,
    "recordsFiltered": 57,
    "data": [
        [
            "Angelica",
            "Ramos",
            "System Architect",
            "London",
            "9th Oct 09",
            "$2,875"
```

```
            ],
            [
                "Ashton",
                "Cox",
                "Technical Author",
                "San Francisco",
                "12th Jan 09",
                "$4,800"
            ],
            ...
        ]
    }
```

Example of server-side processing return using objects, with **DT_RowId** and **DT_RowData** also included, as the data source for the table:

Example:

```
    {
        "draw": 1,
        "recordsTotal": 57,
        "recordsFiltered": 57,
        "data": [
            {
                "DT_RowId": "row_3",
                "DT_RowData": {
                    "pkey": 3
                },
                "first_name": "Angelica",
                "last_name": "Ramos",
                "position": "System Architect",
                "office": "London",
                "start_date": "9th Oct 09",
                "salary": "$2,875"
            },
            {
                "DT_RowId": "row_17",
                "DT_RowData": {
                    "pkey": 17
                },
                "first_name": "Ashton",
                "last_name": "Cox",
                "position": "Technical Author",
                "office": "San Francisco",
                "start_date": "12th Jan 09",
                "salary": "$4,800"
            },
            ...
        ]
    }
```