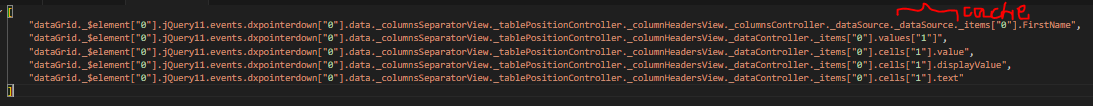
## DataGrid

* DataGrid is a UI component.
* Represent the data from local or remote source in the form of a **grid**.
* Offers basic features like, sorting, grouping, searching.
* Adv. Capabilities like, state storing, client-side exporting, master-detail interface…

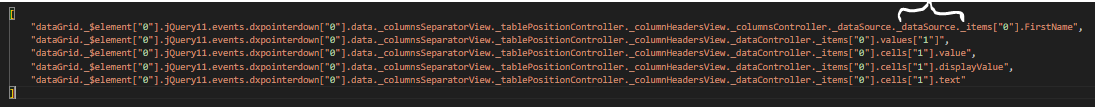
### DataGrid options

* accessKey – no access key in html found
* activeStateEnabled – no visual diff
* allowColumnReordering – user can reorder columns
* allowColumnResizing – user can resize column width
* autoNavigateToFocusedRow – when true, and when focusedRowKey is changed => Automatically scrolls to the focused row (incompatible with **infinite scrolling**).
* cacheEnabled – when performing operations like sorting, grouping, paging => the UI component takes data from this cache. consider disabling it for frequently changing data sources.

Cache enabled:



Cache disabled:



* cellHintEnabled: when cells value is overflowed then enbaling this option, when hover on cell will show complete value of that cell.
* columnAutoWidth: When this property is set to true, all columns adjust their width to the content. columnAutoWidth vs columnWidth: “auto”.
* columnChooser: used to specify which column to hide at runtime. (type – object)
  + allowSeach – seaches column to hide in column chooser box.
  + Empty panel text – text that is to be appear when no column is selected to hide.
  + Enabled – enables column chooser (default f)
  + Heigh and width
  + Mode – specify mode of selecting column to hide (“dragAndDrop”, “select”)
  + searchTimeout
  + title – title of column chooser box
* columnFixing – used to specify which column to fix while horizontal scrolling at runtime. (Type – object)
  + enabled – when true => right clicking on column header gives option in context menu for column fixing (fix and unfix).
  + Texts – object, text to appear in context menu (fix, leftPosition, rightPosition, unfix)
* columnHidingEnabled: should Ui component hide columns to adapt to the screen or container size. If allowColumnResizing is true and columnResizingMode is widget then this property is ignored.
* columnMinWidth: minimum width of the column. (type number).
* columns[]

default: a column is created for each field of DataSource object.

To sepcify which columns should be on ui, u need to use this array.

Each **grid column** is represented by an object containing column setting in this array.

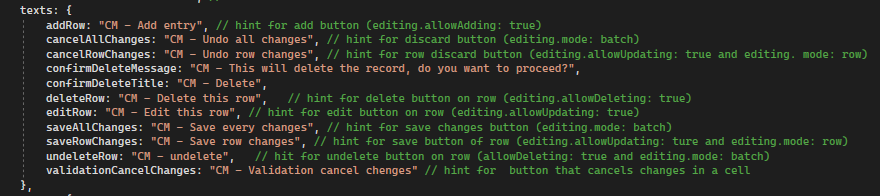
Column properties defines the behaviour and appearance of a grid column.

* alignment – aligns the content of the column. (undefined, “left”, ”right”, “center”)
* allowEditing – can user edit value at runtime. (default inherits from editing.allowEditing). If values in the column are calculated customarily using the calculateCellValue property, they cannot be edited at runtime.
* allowExporting – whether data from this column should be exported. Column should be visible. Do not disable the allowExporting property if you export the DataGrid with multi-line headers.
* allowFiltering – whether data can be filtered by this column.
* allowFixing – whether user can fix the column at runtime. columnFixing should be true.
* allowGrouping – whether the user can group data **by value of this column**. Grouping must be enabled. In a column with calculated values, this property is set to false by default.
* **editing{}**: data grid allow user to add, update, delete data. (allow adding,updating, deleting)
* **allowAdding**: User can add row? (bool)
* **allowDeleting**: user can delete a row? (fn, bool)
* **allowUpdating**: user can update row? (fn, bool)
* **refreshMode**: string

The refreshMode option in DevExtreme's DataGrid specifies the operations that are performed after saving changes. Here's a breakdown of each mode:

1. **Full**:
   * **Data Reloading:** Data is reloaded from the data source. This means that the entire dataset is fetched again, including the data that was just modified, to reflect the latest changes.
   * **Data Processing Operations:** All data processing operations, such as sorting, grouping, and filtering, are reapplied to the reloaded dataset.
   * **UI Component Repaint:** The entire UI component is redrawn to display the reloaded and processed data. This can include re-rendering all rows, headers, and other UI elements.
2. **Reshape**:
   * **Data Reloading: No data reloading occurs.** The current dataset remains the same.
   * **Data Processing Operations:** Data processing operations are reapplied to the current dataset. This means that sorting, grouping, and filtering are performed on the existing data.
   * **UI Component Repaint:** Only the UI component is redrawn to reflect the changes in the processed data. This means that rows and other UI elements may be updated without reloading the entire dataset.
3. **Repaint**:
   * **Data Reloading**: No data reloading occurs. The current dataset remains the same.
   * **Data Processing Operations**: No data processing operations are performed. The current dataset is used as is.
   * **UI Component Repaint**: Only the UI component is redrawn. This means that the UI elements are updated without any changes to the data or its processing.

* **texts**: Contains properties that specify texts for editing-related UI elements.



* **grouping**: grouping allow to **group rows** that share **common values** in one or more column, => making it more easier to analyze and understand data.
* Filtering:

The columns[].filterType property in DevExtreme's DataGrid controls how the header filter behaves when a user interacts with it. It determines whether a user changes the current filter by including (selecting) or excluding (clearing the selection of) values in the filter.

Example Scenario:

Let's say you have a column in your DataGrid that contains the following values: "A", "B", "C", "D", and "E".

Include (Default):

Initially, none of the values are selected in the header filter.

If the user selects "A" and "B", the filter includes only rows where the column value is "A" or "B".

If the user then selects "C", the filter includes rows where the column value is "A", "B", or "C".

Exclude:

Initially, all values are selected in the header filter.

If the user deselects "A" and "B", the filter excludes rows where the column value is "A" or "B".

If the user then deselects "C", the filter excludes rows where the column value is "A", "B", or "C".

**Include**: Users start with no values selected and add values to filter.

**Exclude**: Users start with all values selected and remove values to filter.

* Sorting

Columns[]:

**allowSorting** (true) – determine does this column support sorting

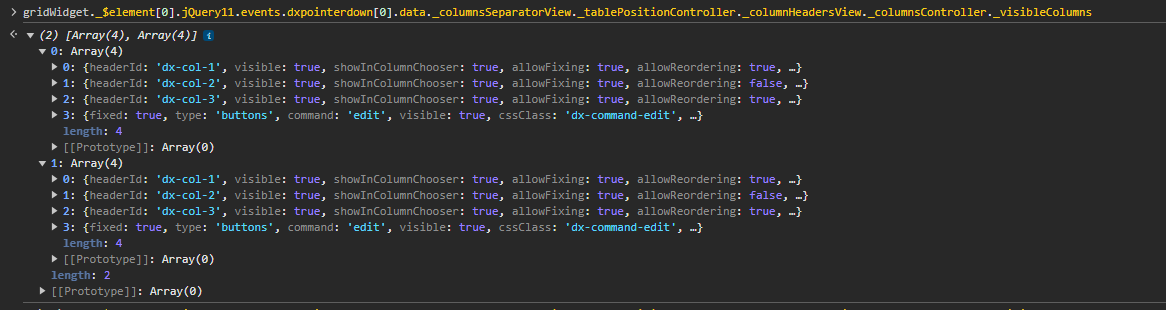
**caculateSortValue** (str, fn) - The calculateSortValue property is used when you want the DataGrid to sort a column based on a value that is not directly displayed in the column. This is useful when the displayed value needs to be sorted differently from its appearance or when sorting should be based on derived or computed values. Used to define custom sorting for individual rows within a column.

Column Customization

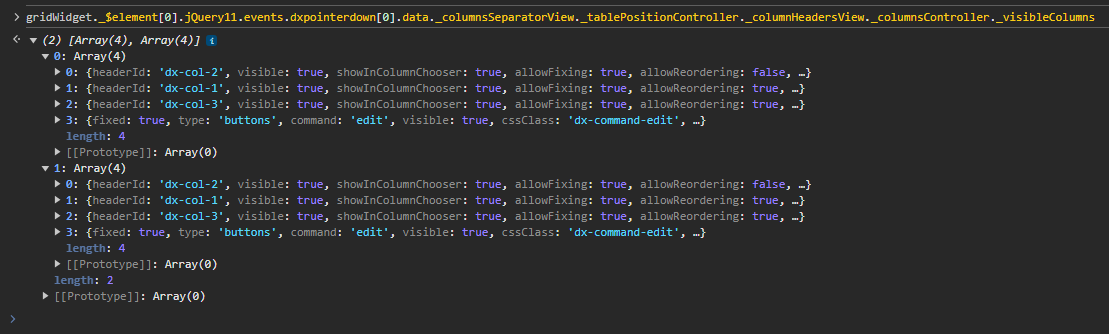
Reordering

xFindValuePathsInObj(gridWidget, 3); to find the path of number 3 in gridWidget

Before reordering –



After reordering



Grid header

xShuffleArrayBetweenIndices(gridWidget.\_$element[0].jQuery11.events.dxpointerdown[0].data.\_columnsSeparatorView.\_tablePositionController.\_columnHeadersView.\_columnsController.\_visibleColumns[0])

gridWidget.repaint()

grid body

xShuffleArrayBetweenIndices(gridWidget.\_$element[0].jQuery11.events.dxpointerdown[0].data.\_columnsSeparatorView.\_tablePositionController.\_columnHeadersView.\_columnsController.\_visibleColumns[1])

gridWidget.repaint()

**cacluateDisplayValue** – ki bhai yhe display value display krna

specifying displayExpr: "name" in the lookup configuration already handles the basic mapping of customerId to the customer name. However, calculateDisplayValue can be particularly useful in more complex scenarios where additional logic is required, or where the display value is derived from multiple fields or requires a specific format.

Why Use calculateDisplayValue When displayExpr Is Already Provided?

Complex Display Logic: calculateDisplayValue allows you to define custom logic to format or transform the display value beyond simple field mapping.

Derived Values: It can be used to derive display values that depend on multiple fields or need additional processing.

Fallback Values: It can provide fallback or default values if the lookup value is not found.

Conditional Formatting: You can apply conditional logic to display different values based on the row data.

**calculateFilterExpression** – ki bhai iss filter exp ke basis pe filter krna

The calculateFilterExpression function in DevExtreme's DataGrid allows you to specify custom rules for filtering data in a column. It provides flexibility in defining how the filter should be applied based on the user input and the selected filter operation. Here's a breakdown of its parameters and how it works:

Function Parameters:

filterValue: The user input value used for filtering. For operations like "between", "anyof", and "noneof", it may contain an array of values.

selectedFilterOperation: The selected filter operation, such as "=", "<>", ">", ">=", "<", "<=", "startswith", "endswith", "contains", or "notcontains".

target: The UI element used for filtering data, such as "filterRow", "headerFilter", "filterBuilder", or "search".

Return Value:

The function should return a filter expression that defines how the data should be filtered. The format of the filter expression depends on the selected filter operation:

For most operations, the filter expression has the format [selector, comparisonOperator, filterValue], where:

selector: Represents the data field or function that returns column values. You can use this.calculateCellValue if your column contains calculated values.

comparisonOperator: One of the comparison operators listed above.

filterValue: The user input value to compare against.

For the "between" operation, the filter expression has the format [ [selector, ">=", startValue], "and", [selector, "<=", endValue] ], where startValue and endValue define the range.

**calculateGroupValue –** ki bhai iske basis pe group krna

The calculateGroupValue property in DevExtreme's DataGrid allows you to define custom logic for grouping grid records. It can be set to either a string or a function.

**calculateSortValue –** ki bhai iss value pr sorting krna

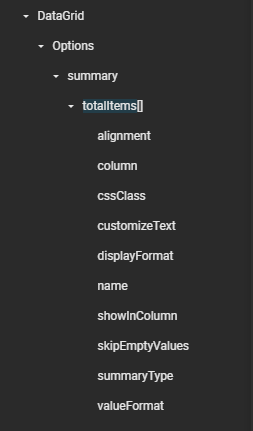
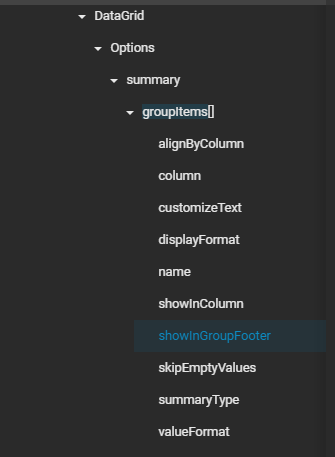
The calculateSortValue property in DevExtreme's DataGrid allows you to specify custom logic for calculating values used to sort a column. This can be useful when you need to sort data based on a value that is not directly present in the data source, or when you need to apply custom sorting logic. Here's a real-life example to illustrate this:

## Summaries in DataGrid

In devextreme datagrid u can display summaries to show aggregated data for **groups** or entire **grid**.

### Grid Summaries

Grid summaries show aggregated values for the entire grid, such as total, avg, min, max value of a col.

### Group Summaries

Group Summaries display aggregate values for each group in the grid. This allows you to show summaries like totals or averages for each group.

### Custom Summaries

Custom Summaries allow you to calculate and display custom aggregate values based on your specific requirements.

## Adaptibility

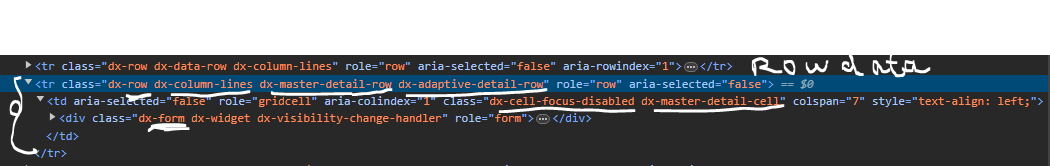
When the total **width of columns exceeds component width**, the DataGrid either **truncates** **column** **cell text** or **adds horizontal scrolling**.

As an alternative, the component **can hide one or several columns** to **prevent** **horizontal scrolling** and **display cell text in full.**

Each column has a unique default hiding priority. The rightmost column has the priority of 0. This value is incremented by 1 for columns from right to left; the column with the lowest priority is hidden first.

You can use the columns[].hidingPriority property to specify custom hiding priorities for those columns that you want to hide.

The detail area contains a dx-form.



onAdaptiveDetailRowPreparing: function (e){}; e object:

