Handsontable

Search

Getting started
Basic usage
Developer guide
Community

**Demos**

Rows and columns

Handsontable PRO / 3.0.0 / Source: Core Switch to CE

# Core

handsontable/src/core.js, line 35

After Handsontable is constructed, you can modify the grid behavior using the available public methods.

## How to call methods

These are 2 equal ways to call a Handsontable method:

```
// all following examples assume that you constructed Handsontable like this
var ht = new Handsontable(document.getElementById('example1'), options);

// now, to use setDataAtCell method, you can either:
ht.setDataAtCell(0, 0, 'new value');
```

```
$('#example1').handsontable('setDataAtCell', 0, 0, 'new value');
```

---

## Methods

handsontable/src/core.js, line 3374

### addHook(key, callback)

Adds listener to the specified hook name (only for this Handsontable instance).

## Parameters:

| Name | Type | Description |
|---|---|---|
| key | String | Hook name. |
| callback | function \| Array | Function or array of Functions. |

See:

- Hooks#add

## Example

```
hot.addHook('beforeInit', myCallback);
```

handsontable/src/core.js, line 3410

### addHookOnce(key, callback)

## Parameters:

| Name | Type | Description |
|---|---|---|
| `key` | String | Hook name. |
| `callback` | function \| Array | Function or array of Functions. |

See:

- Hooks#once

## Example

```
hot.addHookOnce('beforeInit', myCallback);
```

handsontable/src/core.js, line 1857

**alter**`(action, index, amount, source, keepEmptyRows)`

Allows altering the table structure by either inserting/removing rows or inserting/removing columns:

Insert new row(s) above the row with a given `index`. If index is `null` or `undefined`, the new row will be added after the last row.

```
var hot = new Handsontable(document.getElementById('example'));
hot.alter('insert_row', 10);
```

Insert new column(s) before the column with a given `index`. If index is `null` or `undefined`, the new column will be added after the last column.

Remove the row(s) at the given `index`.

```
var hot = new Handsontable(document.getElementById('example'));
hot.alter('remove_row', 10);
```

Remove the column(s) at the given `index`.

```
var hot = new Handsontable(document.getElementById('example'));
hot.alter('remove_col', 10);
```

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `action` | String | | See grid.alter for possible values: `"insert_row"`, `"insert_col"`, `"remove_row"`, `"remove_col"` |
| `index` | Number | | Visual index of the row/column before which the new row/column will be inserted/removed. |
| `amount` | Number | 1 | optional Amound of rows/columns to be inserted/removed. |
| `source` | String | | optional Source indicator. |
| `keepEmptyRows` | Boolean | | optional Flag for preventing deletion of empty rows. |

handsontable/src/core.js, line 1845

### clear()

Clears the data from the grid (the table settings remain intact).

Since:
- 0.11.0

Returns the visual index of the first rendered column.

### Returns: {Number} Visual index of the first visible column.

handsontable/src/core.js, line 1927

**colToProp**`(col)` {String|Number}

Returns the property name that corresponds with the given column index. `DataMap#colToProp`
If the data source is an array of arrays, it returns the columns index.

### Parameters:

| Name | Type | Description |
|---|---|---|
| `col` | Number | Visual column index. |

### Returns: {String|Number} Column property or physical column index.

handsontable/src/core.js, line 2850

**countCols**`()` {Number}

Returns the total number of columns in the grid.

### Returns: {Number} Total number of columns.

handsontable/src/core.js, line 2989

**countEmptyCols**`(ending)` {Number}

## Parameters:

| Name | Type | Description |
|---|---|---|
| `ending` | Boolean | optional If `true`, will only count empty columns at the end of the data source row. |

## Returns: {Number} Count empty cols.

---

handsontable/src/core.js, line 2965

**countEmptyRows**`(ending)` {Number}

Returns the number of empty rows. If the optional ending parameter is `true`, returns the number of empty rows at the bottom of the table.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `ending` | Boolean | optional If `true`, will only count empty rows at the end of the data source. |

## Returns: {Number} Count empty rows.

---

handsontable/src/core.js, line 2943

**countRenderedCols**`()` {Number}

Returns the number of rendered columns (including columns partially or fully rendered outside viewport).

handsontable/src/core.js, line 2921

**countRenderedRows**`()` {Number}

Returns the number of rendered rows (including rows partially or fully rendered outside viewport).

Returns: {Number} Returns -1 if table is not visible.

handsontable/src/core.js, line 2839

**countRows**`()` {Number}

Returns the total number of rows in the grid.

Returns: {Number} Total number in rows the grid.

handsontable/src/core.js, line 2817

**countSourceCols**`()` {Number}

Returns the total number of columns in the data source.

Since:

- 0.26.1

Returns: {Number} Total number in columns in data source.

handsontable/src/core.js, line 2804

**countSourceRows**`()` {Number}

Returns the total number of rows in the data source.

## Returns: {Number} Total number in rows in data source.

---

handsontable/src/core.js, line 2954

**countVisibleCols**`()` {Number}

Returns the number of visible columns. Returns -1 if table is not visible

## Returns: {Number} Number of visible columns or -1.

---

handsontable/src/core.js, line 2932

**countVisibleRows**`()` {Number}

Returns the number of visible rows (rendered rows that fully fit inside viewport).

## Returns: {Number} Number of visible rows or -1.

---

handsontable/src/core.js, line 3200

**deselectCell**`()`

Deselects the current cell selection on grid.

---

handsontable/src/core.js, line 3265

**destroy**`()`

Removes grid from the DOM.

## Fires:

**destroyEditor**`(revertOriginal)`

Destroys the current editor, renders and selects the current cell.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `revertOriginal` | Boolean | optional If != `true`, edited data is saved. Otherwise the previous value is restored. |

handsontable/src/core.js, line 1377

**emptySelectedCells**`()`

Erases content from cells that have been selected in the table.

Since:

- 0.36.0

handsontable/src/core.js, line 3339

**getActiveEditor**`()` {Object}

Returns the active editor object.

## Returns: {Object} The active editor object.

handsontable/src/core.js, line 1899

**getCell**`(row, col, topmost)` {Element}

Returns a TD element for the given `row` and `col` arguments, if it is rendered on screen.
Returns `null` if the TD is not rendered on screen (probably because that part of the table is not visible).

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `col` | Number | Visual column index. |
| `topmost` | Boolean | If set to true, it returns the TD element from the topmost overlay. For example, if the wanted cell is in the range of fixed rows, it will return a TD element from the `top` overlay. |

## Returns: {Element} The cell's TD element.

handsontable/src/core.js, line 2426

**getCellEditor**`(row, col)` {Object}

Returns the cell editor by the provided `row` and `col` arguments.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `col` | Number | Visual column index. |

## Returns: {Object} The Editor object.

handsontable/src/core.js, line 2330

**getCellMeta**`(row, col)` {Object}

Returns the cell properties object for the given `row` and `col` coordinates.

## Parameters:

## Fires:

- Hooks#event:beforeGetCellMeta
- Hooks#event:afterGetCellMeta

## Returns: {Object} The cell properties object.

handsontable/src/core.js, line 2389

**getCellMetaAtRow**`(row)` {Array}

Returns a row off the cell meta array.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Physical index of the row to return cell meta for. |

Since:
- 0.30.0

## Returns: {Array}

handsontable/src/core.js, line 2412

**getCellRenderer**`(row, col)` {function}

Returns the cell renderer function by given `row` and `col` arguments.

| Name | Type | Description |
|---|---|---|
| `row` | Number \| Object | Visual row index or cell meta object. |
| `col` | Number | optional Visual column index. |

Since:

- 0.11

## Returns: {function} The renderer function.

handsontable/src/core.js, line 2439

### **getCellValidator**`(row, col)` {function|RegExp|undefined}

Returns the cell validator by `row` and `col`, provided a validator is defined. If not - it doesn't return anything.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `col` | Number | Visual column index. |

## Returns: {function|RegExp|undefined} The validator function.

handsontable/src/core.js, line 2625

### **getColHeader**`(col)` {Array|String}

Returns an array of column headers (in string format, if they are enabled). If param `col` is given, it returns the header at the given column as a string.

| Name | Type | Description |
|---|---|---|
| `col` | Number | optional Visual column index. |

## Fires:

- Hooks#event:modifyColHeader

## Returns: {Array|String} The column header(s).

---

handsontable/src/core.js, line 2726

**getColWidth**`(col)` {Number}

Returns the width of the requested column.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `col` | Number | Visual column index. |

Since:

- 0.11

## Fires:

- Hooks#event:modifyColWidth

## Returns: {Number} Column width.

---

### Parameters:

| Name | Type | Description |
|---|---|---|
| `elem` | Element | The HTML Element representing the cell. |

### Returns: {CellCoords} Visual coordinates object.

---

handsontable/src/core.js, line 1556

**getCopyableData**`(row, column)` {String}

Returns the data's copyable value at specified row and column index (`DataMap#getCopyable`).

### Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `column` | Number | Visual column index. |

Since:

- 0.19.0

### Returns: {String}

---

handsontable/src/core.js, line 1539

**getCopyableText**`(startRow, startCol, endRow, endCol)` {String}

## Parameters:

| Name | Type | Description |
|---|---|---|
| `startRow` | Number | From visual row index. |
| `startCol` | Number | From visual column index. |
| `endRow` | Number | To visual row index. |
| `endCol` | Number | To visual column index. |

Since:

- 0.11

## Returns: {String}

---

handsontable/src/core.js, line 1515

**getData**`(r, c, r2, c2)` {Array}

Returns the current data object (the same one that was passed by `data` configuration option or `loadData` method,
unless the `modifyRow` hook was used to trim some of the rows. If that's the case - use the `Core#getSourceData` method.).
Optionally you can provide cell range by defining `row`, `col`, `row2`, `col2` to get only a fragment of grid data.

Note: getData functionality changed with the release of version 0.20. If you're looking for the previous functionality,
you should use the `Core#getSourceData` method.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `r` | Number | optional From visual row index. |

|  |  |  |
|---|---|---|
| `c2` | Number | optional To visual column index. |

## Returns: {Array} Array with the data.

---

handsontable/src/core.js, line 2000

**getDataAtCell**`(row, col)` {String|Boolean|null}

Returns the cell value at `row`, `col`. `row` and `col` are the **visible** indexes (note, that if columns were reordered or sorted,
the currently visible order will be used).

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `col` | Number | Visual column index. |

## Returns: {String|Boolean|null} Data at cell.

---

handsontable/src/core.js, line 2028

**getDataAtCol**`(col)` {Array}

Returns array of column values from the data source. `col` is the **visible** index of the column.
Note, that if columns were reordered or sorted, the currently visible order will be used.

## Parameters:

Since:

- 0.9-beta2

## Returns: {Array} Array of cell values.

handsontable/src/core.js, line 2045

**getDataAtProp**`(prop)` {Array}

Given the object property name (e.g. `'first.name'`), returns an array of column's values from the data source.
You can also provide a column index as the first argument.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `prop` | String \| Number | Property name / physical column index. |

Since:

- 0.9-beta2

## Returns: {Array} Array of cell values.

handsontable/src/core.js, line 2160

**getDataAtRow**`(row)` {Array}

Returns a single row of the data. The `row` argument is the **visible** index of the row.

## Parameters:

Since:

- 0.9-beta2

## Returns: {Array} Array of row's cell data.

handsontable/src/core.js, line 2015

### getDataAtRowProp(row, prop) {*}

Return value at `row`, `prop`. (Uses `DataMap#get`)

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `prop` | String | Property name. |

## Returns: {*} Cell value.

handsontable/src/core.js, line 2176

### getDataType(rowFrom, columnFrom, rowTo, columnTo) {String}

Returns a data type defined in the Handsontable settings under the `type` key (Options#type).
If there are cells with different types in the selected range, it returns `'mixed'`.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `rowFrom` | Number | From visual row index. |

| | | |
|---|---|---|
| `columnTo` | Number | To visual column index. |

Since:

- 0.18.1

## Returns: {String} Cell type (e.q: `'mixed'`, `'text'`, `'numeric'`, `'autocomplete'`).

---

handsontable/src/core.js, line 3363

**getInstance**`()` {Handsontable}

Returns the Handsontable instance.

## Returns: {Handsontable} The Handsontable instance.

---

handsontable/src/core.js, line 3350

**getPlugin**`(pluginName)` {*}

Returns plugin instance using the plugin name provided.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `pluginName` | String | The plugin name. |

Since:

- 0.15.0

## Returns: {*} The plugin instance.

Returns an array of row headers' values (if they are enabled). If param `row` was given, it returns the header of the given row as a string.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | optional Visual row index. |

## Fires:

- Hooks#event:modifyRowHeader

## Returns: {Array|String} Array of header values / single header value.

---

handsontable/src/core.js, line 2786

**getRowHeight**`(row)` {Number}

Returns the row height.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |

Since:

- 0.11

## Fires:

Returns: {Number} The given row's height.

handsontable/src/core.js, line 1570

**getSchema**`()` {Object}

Returns schema provided by constructor settings. If it doesn't exist then it returns the schema based on the data
structure in the first row.

Since:

- 0.13.2

Returns: {Object} Schema object.

handsontable/src/core.js, line 1302

**getSelected**`()` {Array.<Array>|undefined}

Returns indexes of the currently selected cells as an array of arrays `[[startRow, startCol, endRow, endCol], ...]`.

Start row and start col are the coordinates of the active cell (where the selection was started).

The version 0.36.0 adds a non-consecutive selection feature. Since this version, the method returns an array of arrays.
Additionally to collect the coordinates of the currently selected area (as it was previously done by the method)
you need to use `getSelectedLast` method.

Returns: {Array.|undefined} An array of arrays of the selection's coordinates.

handsontable/src/core.js, line 1321

**getSelectedLast**`()` {Array|undefined}

Since:

- 0.36.0

## Returns: {Array|undefined} An array of the selection's coordinates.

---

handsontable/src/core.js, line 1340

**getSelectedRange()** {Array.<CellRange>|undefined}

Returns the current selection as an array of CellRange objects.

The version 0.36.0 adds a non-consecutive selection feature. Since this version, the method returns an array of arrays.
Additionally to collect the coordinates of the currently selected area (as it was previously done by the method)
you need to use `getSelectedRangeLast` method.

Since:

- 0.11

## Returns: {Array.|undefined} Selected range object or undefined` if there is no selection.

---

handsontable/src/core.js, line 1358

**getSelectedRangeLast()** {CellRange|undefined}

Returns the last coordinates applied to the table as a CellRange object.

Since:

- 0.36.0

## Returns: {CellRange|undefined} Selected range object or undefined` if there is no selection.

Returns the object settings.

## Returns: {Object} Object containing the current grid settings.

handsontable/src/core.js, line 2068

**getSourceData**`(r, c, r2, c2)` {Array}

Returns the source data object (the same that was passed by `data` configuration option or `loadData` method).
Optionally you can provide a cell range by using the `row`, `col`, `row2`, `col2` arguments, to get only a fragment of grid data.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `r` | Number | optional From physical row index. |
| `c` | Number | optional From physical column index (or visual index, if data type is an array of objects). |
| `r2` | Number | optional To physical row index. |
| `c2` | Number | optional To physical column index (or visual index, if data type is an array of objects). |

Since:

- 0.20.0

## Returns: {Array} Array of grid data.

handsontable/src/core.js, line 2093

**getSourceDataArray**`(r, c, r2, c2)` {Array} of grid data.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `r` | Number | optional From physical row index. |
| `c` | Number | optional From physical column index (or visual index, if data type is an array of objects). |
| `r2` | Number | optional To physical row index. |
| `c2` | Number | optional To physical column index (or visual index, if data type is an array of objects). |

Since:

- 0.28.0

## Returns: {Array} An array of arrays.

---

handsontable/src/core.js, line 2145

**getSourceDataAtCell**`(row, column)` {*}

Returns a single value from the data source.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Physical row index. |
| `column` | Number | Visual column index. |

Since:

- 0.20.0

handsontable/src/core.js, line 2118

**getSourceDataAtCol**`(column)` {Array}

Returns an array of column values from the data source. `col` is the index of the row in the data source.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `column` | Number | Visual column index. |

Since:

- 0.11.0-beta3

## Returns: {Array} Array of the column's cell values.

handsontable/src/core.js, line 2132

**getSourceDataAtRow**`(row)` {Array|Object}

Returns a single row of the data (array or object, depending on what you have). `row` is the index of the row in the data source.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Physical row index. |

Since:

- 0.11.0-beta3

## Returns: {Array|Object} Single row of data.

Get phrase for specified dictionary key.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `dictionaryKey` | String | Constant which is dictionary key. |
| `extraArguments` | * | Arguments which will be handled by formatters. |

Since:

- 0.35.0

## Returns: {String}

---

handsontable/src/core.js, line 1787

### getValue() {*}

Get value from the selected cell.

Since:

- 0.11

## Returns: {*} Value of selected cell.

---

handsontable/src/core.js, line 2604

### hasColHeaders() {Boolean}

Returns information about if this table is configured to display column headers.

Since:

- 0.11

handsontable/src/core.js, line 3392

### hasHook(key) {Boolean}

Check if for a specified hook name there are added listeners (only for this Handsontable instance).

## Parameters:

| Name | Type | Description |
|---|---|---|
| key | String | Hook name |

See:

- Hooks#has

## Returns: {Boolean}

## Example

```
var hasBeforeInitListeners = hot.hasHook('beforeInit');
```

handsontable/src/core.js, line 2592

### hasRowHeaders() {Boolean}

Returns information about if this table is configured to display row headers.

Since:

- 0.11

handsontable/src/core.js, line 3029

### **isEmptyCol**`(col)` {Boolean}

Check if all cells in the the column declared by the `col` argument are empty.

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `col` | Number | Column index. |

#### Returns: {Boolean} `true` if the column at the given `col` is empty, `false` otherwise.

handsontable/src/core.js, line 3017

### **isEmptyRow**`(row)` {Boolean}

Check if all cells in the row declared by the `row` argument are empty.

#### Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |

#### Returns: {Boolean} `true` if the row at the given `row` is empty, `false` otherwise.

handsontable/src/core.js, line 1205

### **isListening**`()` {Boolean}

## Returns: {Boolean} `true` if the instance is listening, `false` otherwise.

handsontable/src/core.js, line 1164

**listen**`(modifyDocumentFocus)`

Listen to the keyboard input on document body.

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `modifyDocumentFocus` | Boolean | true | optional If `true`, currently focused element will be blured (which returns focus to the document.body). Otherwise the active element does not lose its focus. |

Since:
- 0.11

handsontable/src/core.js, line 1424

**loadData**`(data)`

Reset all cells in the grid to contain data from the data array.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `data` | Array | Array of arrays or array of objects containing data. |

## Fires:

handsontable/src/core.js, line 1228

**populateFromArray**
`(row, col, input, endRow, endCol, source, method, direction, deltas)`
{Object|undefined}

Populate cells at position with 2D input array (e.g. `[[1, 2], [3, 4]]`).
Use `endRow`, `endCol` when you want to cut input when a certain row is reached.
Optional `source` parameter (default value "populateFromArray") is used to identify this call in the resulting events (beforeChange, afterChange).
Optional `populateMethod` parameter (default value "overwrite", possible values "shift_down" and "shift_right")
has the same effect as pasteMode option `Options#pasteMode`

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `row` | Number | | Start visual row index. |
| `col` | Number | | Start visual column index. |
| `input` | Array | | 2d array |
| `endRow` | Number | | optional End visual row index (use when you want to cut input when certain row is reached). |
| `endCol` | Number | | optional End visual column index (use when you want to cut input when certain column is reached). |
| `source` | String | "populateFromArray" | optional Source string. |
| `method` | String | "overwrite" | optional Populate method. Possible options: `shift_down`, `shift_right`, `overwrite`. |
| `direction` | String | | Populate direction. (left\|right\|up\|down) |
| `deltas` | Array | | Deltas array. |

Since:

- 0.9.0

handsontable/src/core.js, line 1940

**propToCol**`(prop)` {Number}

Returns column index that corresponds with the given property. `DataMap#propToCol`

## Parameters:

| Name | Type | Description |
|---|---|---|
| `prop` | String \| Number | Property name or physical column index. |

## Returns: {Number} Visual column index.

handsontable/src/core.js, line 2232

**removeCellMeta**`(row, col, key)`

Remove a property defined by the `key` argument from the cell meta object for the provided `row` and `col` coordinates.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `col` | Number | Visual column index. |
| `key` | String | Property name. |

## Fires:

handsontable/src/core.js, line 3429

### removeHook(key, callback)

Removes the hook listener previously registered with `Core#addHook`.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `key` | String | Hook name. |
| `callback` | function | Function which have been registered via `Core#addHook`. |

See:

- Hooks#remove

## Example

```
hot.removeHook('beforeInit', myCallback);
```

handsontable/src/core.js, line 1408

### render()

Rerender the table.

handsontable/src/core.js, line 2899

### rowOffset() {Number}

Returns an visual index of the first rendered row.

handsontable/src/core.js, line 3447

### runHooks(key, p1, p2, p3, p4, p5, p6) {*}

Run the callbacks for the hook provided in the `key` argument using the parameters given in the other arguments.

## Parameters:

| Name | Type | Description |
|---|---|---|
| key | String | Hook name. |
| p1 | * | optional Argument passed to the callback. |
| p2 | * | optional Argument passed to the callback. |
| p3 | * | optional Argument passed to the callback. |
| p4 | * | optional Argument passed to the callback. |
| p5 | * | optional Argument passed to the callback. |
| p6 | * | optional Argument passed to the callback. |

See:

- Hooks#run

## Returns: {*}

## Example

```
hot.runHooks('beforeInit');
```

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `row` | Number | | optional Visual row index. |
| `column` | Number | | optional Visual column index. |
| `snapToBottom` | Boolean | false | optional If `true`, viewport is scrolled to show the cell on the bottom of the table. |
| `snapToRight` | Boolean | false | optional If `true`, viewport is scrolled to show the cell on the right side of the table. |

Since:

- 0.24.3

## Returns: {Boolean} `true` if scroll was successful, `false` otherwise.

handsontable/src/core.js, line 3210

**selectAll**`()`

Select the whole table. The previous selection will be overwritten.

Since:

- 0.38.2

handsontable/src/core.js, line 3041

**selectCell**`(row, column, endRow, endColumn, scrollToCell, changeListener)`
{Boolean}

Select cell specified by `row` and `col` values or a range of cells finishing at `endRow`, `endCol`. If the table was configured to support data column properties that properties can be used to making a selection.

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `row` | Number | | Visual row index. |
| `column` | Number \| String | | Visual column index or column property. |
| `endRow` | Number | | optional Visual end row index (if selecting a range). |
| `endColumn` | Number \| String | | optional Visual end column index or column property (if selecting a range). |
| `scrollToCell` | Boolean | true | optional If `true`, the viewport will be scrolled to the selection. |
| `changeListener` | Boolean | true | optional If `false`, Handsontable will not change keyboard events listener to himself. |

## Returns: {Boolean} `true` if selection was successful, `false` otherwise.

## Example

```
// select a single cell
hot.selectCell(2, 4);
// select a single cell using column property
hot.selectCell(2, 'address');
// select a range of cells
hot.selectCell(2, 4, 3, 5);
// select a range of cells using column properties
hot.selectCell(2, 'address', 3, 'phone_number');
// select a range of cells without scrolling to them
hot.selectCell(2, 'address', 3, 'phone_number', false);
```

`(row, prop, endRow, endProp, scrollToCell, changeListener)` {Boolean}

Select the cell specified by the `row` and `prop` arguments, or a range finishing at `endRow`, `endProp`. By default, viewport will be scrolled to selection.

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `row` | Number | | Visual row index. |
| `prop` | String | | Property name. |
| `endRow` | Number | | optional visual end row index (if selecting a range). |
| `endProp` | String | | optional End property name (if selecting a range). |
| `scrollToCell` | Boolean | true | optional If `true`, viewport will be scrolled to the selection. |
| `changeListener` | Boolean | true | optional If `false`, Handsontable will not change keyboard events listener to himself. |

Deprecated
- Yes

## Returns: {Boolean} `true` if selection was successful, `false` otherwise.

handsontable/src/core.js, line 3080

**selectCells**`(coords, scrollToCell, changeListener)` {Boolean}

Make multiple, non-contiguous selection specified by `row` and `column` values or a range of cells finishing at `endRow`, `endColumn`. The method supports two input formats which are the same as that produces by `getSelected` and `getSelectedRange` methods.

By default, viewport will be scrolled to selection. After the `selectCells` method had finished, the instance will be listening to keyboard input on the document.

## Parameters:

| | | | |
|---|---|---|---|
| `coords` | <Array> \| Array.<CellRange> | | the same format as `getSelected` method returns or as an CellRange objects which is the same format what `getSelectedRange` method returns. |
| `scrollToCell` | Boolean | true | optional If `true`, the viewport will be scrolled to the selection. |
| `changeListener` | Boolean | true | optional If `false`, Handsontable will not change keyboard events listener to himself. |

Since:

- 0.38.0

## Returns: {Boolean} `true` if selection was successful, `false` otherwise.

## Example

```
// using an array of arrays
hot.selectCells([[1, 1, 2, 2], [3, 3], [6, 2, 0, 2]]);
// using an array of arrays with defined columns as props
hot.selectCells([[1, 'id', 2, 'first_name'], [3, 'full_name'], [6, 'last_name', 0,
// or using an array of CellRange objects (produced by `.getSelectedRange()` metho
const selected = hot.getSelectedRange();

selected[0].from.row = 0;
selected[0].from.col = 0;

hot.selectCells(selected);
```

handsontable/src/core.js, line 3150

**selectColumns**`(startColumn, endColumn)` {Boolean}

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `startColumn` | Number | | The visual column index from which the selection starts. |
| `endColumn` | Number | startColumn | optional The visual column index to which the selection finishes. If `endColumn` is not defined the column defined by `startColumn` will be selected. |

Since:

- 0.38.0

## Returns: {Boolean} `true` if selection was successful, `false` otherwise.

## Example

```
// select column using visual index
hot.selectColumns(1);
// select column using column property
hot.selectColumns('id');
// select range of columns using visual indexes
hot.selectColumns(1, 4);
// select range of columns using column properties
hot.selectColumns('id', 'last_name');
```

handsontable/src/core.js, line 3177

**selectRows**`(startRow, endRow)` {Boolean}

## Parameters:

| Name | Type | Default | Description |
|---|---|---|---|
| `startRow` | Number | | The visual row index from which the selection starts. |
| `endRow` | Number | startRow | optional The visual row index to which the selection finishes. If `endRow` is not defined the row defined by `startRow` will be selected. |

Since:

- 0.38.0

## Returns: {Boolean} `true` if selection was successful, `false` otherwise.

## Example

```
select row using visual index
hot.selectRows(1);
select range of rows using visual indexes
hot.selectRows(1, 4);
```

handsontable/src/core.js, line 2291

**setCellMeta**`(row, col, key, val)`

Sets a property defined by the `key` object to the meta object of a cell corresponding to params `row` and `col`.

## Parameters:

| | | |
|---|---|---|
| `key` | String | Property name. |
| `val` | String | Property value. |

Since:

- 0.11

## Fires:

- Hooks#event:afterSetCellMeta

handsontable/src/core.js, line 2270

**setCellMetaObject**`(row, col, prop)`

Set cell meta data object defined by `prop` to the corresponding params `row` and `col`.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |
| `col` | Number | Visual column index. |
| `prop` | Object | Meta object. |

Since:

- 0.11

handsontable/src/core.js, line 1076

**setDataAtCell**`(row, col, value, source)`

Set new value to a cell. To change many cells at once, pass an array of `changes` in format `[[row, col, value], ...]` as
the only parameter. `col` is the index of a **visible** column (note that if columns were reordered,
the current visible order will be used). `source` is a flag for before/afterChange events. If you pass only

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number \| Array | Visual row index or array of changes in format `[[row, col, value], ...]`. |
| `col` | Number | Visual column index. |
| `value` | String | New value. |
| `source` | String | optional String that identifies how this change will be described in the changes array (useful in onAfterChange or onBeforeChange callback). |

handsontable/src/core.js, line 1125

**setDataAtRowProp**`(row, prop, value, source)`

Set new value to a cell. To change many cells at once, pass an array of `changes` in format `[[row, prop, value], ...]` as
the only parameter. `prop` is the name of the object property (e.g. `first.name`). `source` is a flag for before/afterChange events.
If you pass only array of changes then `source` could be set as second parameter.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number \| Array | Visual row index or array of changes in format `[[row, prop, value], ...]`. |
| `prop` | String | Property name or the source string. |
| `value` | String | Value to be set. |
| `source` | String | optional String that identifies how this change will be described in changes array (useful in onChange callback). |

handsontable/src/core.js, line 1260

**spliceCol**`(col, index, amount, elements)`

Parameter `index` is the row index at which to start changing the array.
If negative, will begin that many elements from the end. Parameter `amount`, is the number of the old array elements to remove.
If the amount is 0, no elements are removed. Fourth and further parameters are the `elements` to add to the array.
If you don't specify any elements, spliceCol simply removes elements from the array.
`DataMap#spliceCol`

## Parameters:

| Name | Type | Description |
|---|---|---|
| `col` | Number | Index of the column in which do you want to do splice. |
| `index` | Number | Index at which to start changing the array. If negative, will begin that many elements from the end. |
| `amount` | Number | An integer indicating the number of old array elements to remove. If amount is 0, no elements are removed. |
| `elements` | * | optional The elements to add to the array. If you don't specify any elements, spliceCol simply removes elements from the array. |

Since:

- 0.9-beta2

handsontable/src/core.js, line 1281

**spliceRow**`(row, index, amount, elements)`

Adds/removes data from the row. This function works is modelled after Array.splice.
Parameter `row` is the index of row in which do you want to do splice.
Parameter `index` is the column index at which to start changing the array.
If negative, will begin that many elements from the end. Parameter `amount`, is the number of old array elements to remove.
If the amount is 0, no elements are removed. Fourth and further parameters are the `elements` to add to the array.
If you don't specify any elements, spliceCol simply removes elements from the array.
`DataMap#spliceRow`

| | | |
|---|---|---|
| `row` | Number | Index of column in which do you want to do splice. |
| `index` | Number | Index at which to start changing the array. If negative, will begin that many elements from the end. |
| `amount` | Number | An integer indicating the number of old array elements to remove. If amount is 0, no elements are removed. |
| `elements` | * | optional The elements to add to the array. If you don't specify any elements, spliceCol simply removes elements from the array. |

Since:

- 0.11

---

handsontable/src/core.js, line 1987

**toPhysicalColumn**`(column)` {Number}

Translate visual column index into physical.
If displayed columns order is different than the order of columns stored in memory (i.e. manual column move is applied)
to retrieve valid physical column index you can use this method.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `column` | Number | Visual column index. |

Since:

- 0.29.0

## Returns: {Number} Returns physical column index.

---

handsontable/src/core.js, line 1974

**toPhysicalRow**`(row)` {Number}

## Parameters:

| Name | Type | Description |
|---|---|---|
| `row` | Number | Visual row index. |

Since:
- 0.29.0

## Returns: {Number} Returns physical row index.

handsontable/src/core.js, line 1963

**toVisualColumn**`(column)` {Number}

Translate physical column index into visual.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `column` | Number | Physical column index. |

Since:
- 0.29.0

## Returns: {Number} Returns visual column index.

handsontable/src/core.js, line 1952

**toVisualRow**`(row)` {Number}

| Name | Type | Description |
|---|---|---|
| `row` | Number | Physical row index. |

Since:

- 0.29.0

## Returns: {Number} Returns visual row index.

handsontable/src/core.js, line 1191

### **unlisten**`()`

Stop listening to keyboard input on the document body.

Since:

- 0.11

handsontable/src/core.js, line 1583

### **updateSettings**`(settings, init)`

Use it if you need to change configuration after initialization. The `settings` parameter is an object containing the new
settings, declared the same way as in the initial settings object.
Note, that although the `updateSettings` method doesn't overwrite the previously declared settings, it might reset
the settings made post-initialization. (for example - ignore changes made using the columnResize feature).

## Parameters:

| Name | Type | Description |
|---|---|---|
| `settings` | Object | New settings object. |

## Fires:

- Hooks#event:afterCellMetaReset
- Hooks#event:afterUpdateSettings

## Example

```
hot.updateSettings({
   contextMenu: true,
   colHeaders: true,
   fixedRowsTop: 2
});
```

handsontable/src/core.js, line 2458

**validateCells**(callback)

Validates all cells using their validator functions and calls callback when finished.

If one of the cells is invalid, the callback will be fired with `'valid'` arguments as `false` - otherwise it would equal `true`.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `callback` | function | optional The callback function. |

Validates columns using their validator functions and calls callback when finished.

If one of the cells is invalid, the callback will be fired with `'valid'` arguments as `false` - otherwise it would equal `true`.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `columns` | Array | optional Array of validation target visual columns indexes. |
| `callback` | function | optional The callback function. |

handsontable/src/core.js, line 2471

**validateRows**`(rows, callback)`

Validates rows using their validator functions and calls callback when finished.

If one of the cells is invalid, the callback will be fired with `'valid'` arguments as `false` - otherwise it would equal `true`.

## Parameters:

| Name | Type | Description |
|---|---|---|
| `rows` | Array | optional Array of validation target visual row indexes. |
| `callback` | function | optional The callback function. |