DataTable

- One of the most complex components, slightly smaller than the RTE, but if you add the DataSource (almost mandatory) and Paginator (optional), it grows larger. Largest number of members in its API. Most tickets
- Beloved of old-style data-processing guys (such as myself): DataTable gets very well along database tables. It is not a DataGrid (like a spreadsheet)
- It originally (2.2.0) had only eight classes, it has grown to 18, 8 of them cell editors. It had 143k, now 519k (-debug, not including dependencies)

DataTable 2.3.0

- It inherited from Element
- Moved many of its properties to configuration attributes
- It also got skins.

DataTable 2.5.0

- Paginator was split into a separate class.
- Rendering was offloaded to an async queue handled by `YAHOO.util.Chain`, one more step in the fight against IE pathetic rendering speed, along with `renderLoopSize`.
- First major standardizing of names

DataTable 2.6.0: GA

- General Availablity,
- Final name changes.
- Integrated support for server-side paging and sorting (it used to be a trick).
- Cell editors became separate classes.
- Data saved from cell editor can be checked against the server (`asyncSubmitter`)
- `ScrollingDataTable` (big change, headers are a separate table that has to be kept in sync)

Common errors/misconceptions

- The data is in the RecordSet, the HTML is a reflection of that: no data should be stored in the HTML. Checkboxes are the most frequent case. (remember: the HTML is volatile)
- The data is not in the DataSource
- The largest delay is usually in rendering (in IE) not in data transfer.
  - IE can be 10 times slower than others.
  - Changing even to client-side pagination makes everything faster for the same amount of data transferred.

Common errors/misconceptions

- The data should be stored in the native data format, parsed in the DataSource

```
myDataSource.responseSchema = {
    resultsList: "ResultSet.Result",
    fields: [
         "Title", "Phone",  "City",
         {key:"Rating.AverageRating",parser:"number"},
         "ClickUrl"
    ]
};
```

If you don't, you get AverageRating sorted like this:

```
1, 11, 12, 2, 22, 23, 3
```

- Dates should be stored as Date objects. DataSource date parser uses `Date.parse()`, it doesn't parse any date.

Common errors/misconceptions

- Usual question: “Why list them twice?”
- Fields in the DataSource and columns in the Column Definitions need not match in number or position.
  - Fields not listed in the ColDefs are available in the RecordSet for other purposes
  - Non-existing fields listed in the ColDefs can be generated via formatters: buttons, checkboxes

Common errors/misconceptions

- People encode info into links via custom formatters, including href="javascript:"-type links such as (bad:)

```
formatSomeField = function(elCell, oRecord, oColumn, sData) {
          elCell.innerHTML = '<a href="javascript:dosomething('
          + sData + ', \\"' + oRecord.getData('someFieldName') +
          '\\");">' + sData + '</a>';
   };
```

- Instead, listen to click or other events and then decide what to do (good):

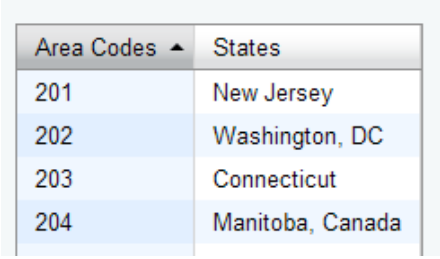
```
myDataTable.on('cellClickEvent', function (oArgs) {
        var target = oArgs.target,
            record = this.getRecord(target),
            column = this.getColumn(target);
        switch (column.key) {
                // ....
        }
});
```

Common errors/misconceptions

- Do not attach events to DataTable's HTML elements.
- The HTML gets redrawn quite often and events will be left behind: big memory leak!
- Use DataTable's own events
  - They use Event Delegation (all events are listened to at the <table> level)
  - They give you the target already resolved

Common errors/misconceptions

- `sortedBy` signals how it is sorted, not how to sort it. It signals state, it is not a command.
- `sortColumn` is the command to sort. It will update `sortedBy` when finished
- `sortedBy` simply produces the distinctive look for sorted columns:



Area Codes ▲	States
201	New Jersey
202	Washington, DC
203	Connecticut
204	Manitoba, Canada

Wishlist

- There should be an `InputField` class and let the cell editors, Forms and whatever use those fields.
- Make a row editor
- Support looping through records via some sort of each() or some() methods

```
for (var rs = this.getRecordset(),
         l = rs.getLength(), i=0;
       i < l;
         i++
    ) {
    var r = rs.getRecord(i);
    // ...
}
```

- Grouping – nesting, people keep asking for it.

Wishlist

- Variable page size (page by initials, or other ranges such as price or dates)

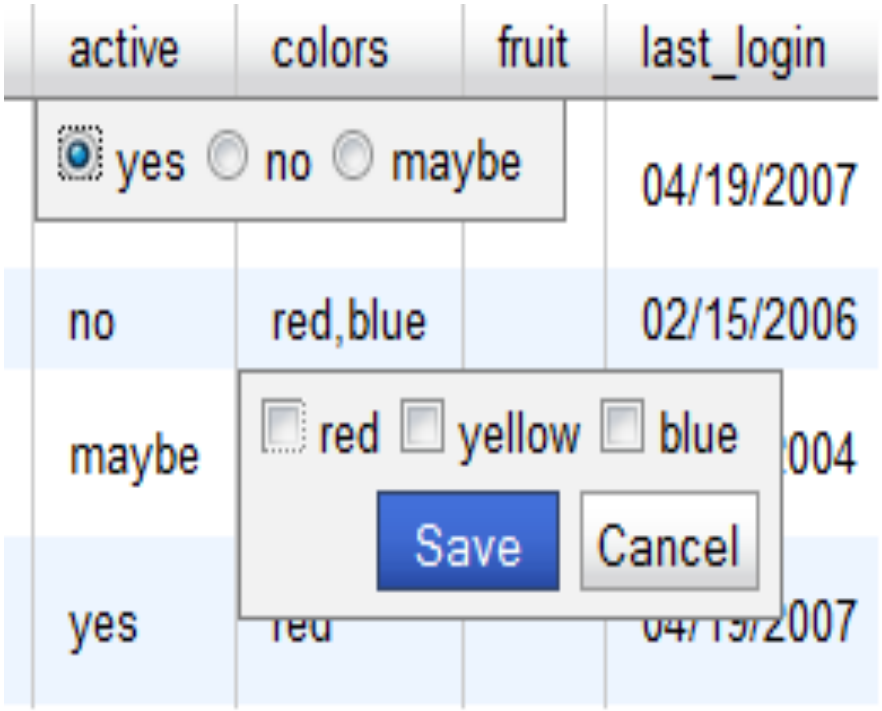
```
<< first < prev [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] next > last >>


<< first < prev [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] next > last >>
<< first < prev [0-100] [101-200] [201-300] [301+] next > last >>
```

- Filtering on show renders page sizes of different sizes

Wishlist

- Mismatch in between similarly named editors and formatters
  - DropdownCellEditor and formatDropdown
  - CheckboxCellEditor and formatCheckbox
  - RadioCellEditor and formatRadio
- There is also no built-in formatter to show data selected from DropdownCellEditor or RadioCellEditor when the keys and values of options don't match.

Radio and Checkbox Cell Edtors



Several options on the same cell

Checkbox and Radio formatters

Single option per cell. Radios are mutually exclusive only within a page.

Wishlist

- Better dropdown options handling: load asynchronously on request. Same for radios and checkboxes.

```
{key:"state", editor: new YAHOO.widget.DropdownCellEditor({
        dropdownOptions:YAHOO.example.Data.stateAbbrs,
        disableBtns:true
})},

YAHOO.example.Data.stateAbbrs = [
    "AL","AK","AZ","AR","CA","CO","CT","DE","DC","FL","GA","HI",
    "ID","IL","IN","IA","KS","KY","LA","ME","MD","MA","MI","MN","MS",
    "MO","MT","NE","NV","NH","NJ","NM","NY","NC","ND","OH","OK","OR",
    "PA","RI","SC","SD","TN","TX","UT","VT","VA","WA","WV","WI","WY"
];
```

Not easy to handle:

```
YAHOO.example.Data.states =   [
        {label:"Alabama",  value:"AL"}, {label:"Alaska",   value:"AK"},
        {label:"Arizona",  value:"AZ"}, {label:"Arkansas", value:"AR"},
… ];
```

Wishlist

- Improve button, link and email formatters (anchor's href and innerHTML should come from separate fields, button text fixed)

```
// existing simplified :
formatLink : function(el, oRecord, oColumn, oData) {
   el.innerHTML =
      "<a href=\"" + oData + "\">" + oData + "</a>";
}

 // desirable:
{key:'title', label:'Title', sortable:true,
  formatter:'link',linkOptions:{
     urlSource:'url',
     target:'Album Page'
  }
},
```

Wishlist

- Extra cell editors
  - Numeric only
  - Validated via RegExp
  - Autocompleted

Wishlist

- The best would be to declare in the Column Definition the `type` of the field, which would set the formatter, editor and parser (specially if there is no separate DataSource)
  - Shortnames for *type* would be the keys to a table much as `DT.Formatter` and `DT.Editors`

```
DT.Types = {
   number:{
      parser: DS.parseNumber,
      formatter:DT.formatNumber,
      editor:widget.TextboxCellEditor
   },
```

  - The table could be completed with a formatter to send the information back to the server, the complement to the parser.

Wishlist

- Row selection via checkboxes, plus SelectAll functionality and
  - Make it easier to execute actions on those selected (usually, DeleteRows)
  - Looping on all records help, there might be a "loop on selected" method
- Aliases on DataSource: The DataSource should translate the fields names to internal aliases.
  - That would aleviate having `key` and `field` properties in the Column Defs

Wishlist

- DataTable assumes that Record #5 corresponds to Row #5, preventing the insertion of extra custom rows because then the ordinal positions would no longer match. This makes all row-expansion, master/detail, subtotals messy and/or fragile.
  - Why can't the Record have a reference to the TR element it is used to display it?
- Aggregates: totals, averages, counts, subtotals
  - table aggregates can be added in the tfoot section
  - group aggregates (subtotals) require adding extra rows