TsTable Tutorial

Table of Contents

Introduction

This tutorial is a must-read if you plan to use TsTable() in any but the simplest modes. I've split it off from the main doc page because that page was just too long and crowded, and too slow to load. So now it is here. Enjoy!

"Apply" mode: making an existing table sortable

Basics

When using "apply" mode, you're taking an existing table on your page and making it sortable. The only argument that you can supply (if desired) is **options**, which make apply mode pretty simple.

The easiest way to go is to just put the TSTable() call right before the table in question, as in the first example above. Note that it is best to use a DekiScript block (select style="DekiScript" in the editor) rather than enclosing the template call in double braces {{ }} because this will prevent an extra chunk of whitespace from appearing before the table. So, to repeat the first example above, we have this:

```
TSTable()
```

text	numbers	dates
this	9	July 22, 2009
is	-1	August 6, 2009
text	11	April 1, 2010

which then becomes:

text	numbers	dates
this	9	July 22, 2009
is	-1	August 6, 2009
text	11	April 1, 2010

Although the template is pretty flexible, there is a basic requirement that the table have a single header row up top, although the actual HTML structure doesn't matter (i.e., you don't need the header row to be in a <thead> element; the script will handle all that stuff on its own).

Note that initially, the table is unsorted; you need to click one of the column headers to activate the sort (and can shift-click multiple headers to enable multiple levels of sort).

The tablesorter code is generally pretty smart about figuring out the format of the data in each column, so it appropriately sorts text, numbers, and dates in this particular example. There's one important caveat, though: to sort correctly, *all entries* in the column must have the same general format. This is especially important with dates. The actual formats are a bit flexible, and are described under "Supported Data Types" towards the bottom of this page. Just be aware that if you see your date columns being sorted incorrectly, it can be caused by just a single badly formatted entry. I hope to improve this behavior in the future.

Some useful options

The documentation for **options** above shows that there are three options available to us in apply mode:

- **id** lets us specify a particular table to apply to, not just the next one we find on the page. This is occasionally useful in apply mode, but not very often.
- **zebra** applies zebra-striping to the table rows. This is very useful in larger tables, provided the table doesn't already use a lot of different colors.

- **initial** lets us specify which columns will be used to sort the table initially. The format is a list of two-element lists, each of which specifies (1) the column (counting from 0), and (2) the order (0 = ascending, 1 = descending). Because it is a *list* of these two-element lists, that means I can sort on multiple columns. So, for example:
  ```
      initial:[ [2,0] , [3,1] ]
  ```
  says that the table will initially be sorted ascending on column 2, and then descending on column 3.

Here's a working example:

```
TSTable{ options:{ zebra:true, initial:[ [2,0] , [3,1] ] } }
```

Column 0	Column 1	Column 2	Column 3
This	is	3	1
our	first	3	2
interesting	example	5	2

Column 0	Column 1	Column 2	Column 3
This	is	3	1
our	first	3	2
interesting	example	5	2

"Generate" mode: creating a customized sortable table

Basics

Apply mode is nice, but the real purpose of this template is to *generate* tables, using what we shall creatively refer to as "generate" mode. To generate a table, we must specify the **columns** argument, which will define the table format, and the **data** argument, which will be used to generate the actual table contents.

Let's go backwards and first discuss the **data** argument. As the docs show, this must be either a *list* of maps or a *map*of maps, where each interior map contains data for one row of the table. If you're constructing **data** in DekiScript, you'll usually produce a list of maps; the map of maps option is for some wiki data structures. For instance, page.subpages is a map of maps; TSTable() will automatically convert this to a list of maps so you can use it directly. Our examples will intermix the two options as necessary. Each row map in **data** may have many more fields than are actually displayed in the table, which is fine; the **columns** argument will determine which data gets displayed, and how.

Columns is also a list, where each element may be either a map or a string. A string is just a simple shorthand notation, and will be converted internally into this map: `{key:`*`string`*`}`. We'll initially focus on this simple format, before exploring the dizzying array of options in the map format.

In the simple **columns** format, each element specifies both the title of the column, and the key of the map element in the row data that will be used for the cell data. So here's a super-easy way to produce a table showing a summary of pages:

```
TSTable{ columns:[ "Title", "ID", "Viewcount" ], data: wiki.getpage("DekiScript/
Reference").subpages }
```

Title	ID	Viewcount
DekiScript Syntax	9748	209
DekiScript Operators	9268	146
Wiki Functions and Variables	10432	584
DekiScript Remote Function Calls	11206	143
DekiScript Statements	11333	187
DekiScript Literals	9431	244
HTML Page Composition Model	7358	193
DekiScript HTML/XML Statements	8847	279
DekiScript Functions and Variables	6479	656
JavaScript Events & Messages (JEM)	25589	738
XPath	33097	108

Note that the columns fields are case-insensitive, so the capitalization can be set for how you want the column header to look.

We have a few additional **options** available to us now that we're in generate mode. We can now specify the table width and a style for each row, and I can enable the **pager**. So, I could do this if I really wanted to:

```
TSTable{
    options: { pager:true, width:"100%", zebra:true, rowstyle:"font-weight:bold",
initial:[ [0,0] ] },
    columns:[ "Title", "ID", "Viewcount" ],
    data: wiki.getpage("en/docs/DekiScript/Reference").subpages
}
```

Title	ID	Viewcount
DekiScript Syntax	**9748**	**209**
DekiScript Operators	**9268**	**146**
Wiki Functions and Variables	**10432**	**584**
DekiScript Remote Function Calls	**11206**	**143**
DekiScript Statements	**11333**	**187**
DekiScript Literals	**9431**	**244**
HTML Page Composition Model	**7358**	**193**
DekiScript HTML/XML Statements	**8847**	**279**
DekiScript Functions and Variables	**6479**	**656**
JavaScript Events & Messages (JEM)	**25589**	**738**
XPath	**33097**	**108**

10

The "rowstyle" option is generally more useful as an evaluated DekiScript expression, but we'll get to that later.

Displaying a CSV file

Now let's display a CSV file in a sortable table. Never mind that in this case it would be easier to just use the csv2Table template; this example is designed to show how much you can do with a few lines of code when you take full advantage of DekiScript and TSTable, even using the simple columns format (you also have the filters available if you want.)
 Here's the code:

```
<div style="max-height:400px;overflow:auto">
var uri = 'http://developer.mindtouch.com/@api/deki/files/4826/
=Export_Sales_Info-demo.csv';
var csv = string.splitcsv(web.text(uri)).values;
tstable { columns:csv[0], data:list.splice(csv,0,1) };
</div>
```

How does it work? First, we have wrapped the entire table in a <div>, and set it to limit the height of the thing and scroll if the table is too big.

Next, we fetch a CSV file, use DekiScript's string.splitcsv() function to parse it, and then just take the values portion of the return.

Now we call TSTable. For columns, we'll simply use the first row of the csv, which is a list of strings. For data, we use the rest of the rows of data, which are also lists of strings. We use list.splice to pick off the required rows.

That's it!
TSTABLE ERRORS:

- **must specify COLUMNS when DATA is also specified ('generate' mode)**

(no data)

Advanced Columns format

We've already seen enough to get some good use out of TSTable, but we can do lots more by using the advanced "map" format of **columns**.

"Key" and "Title"

For any column specifier that is a map, you *must* provide a **key**, which points to the cell value in each row of the data (as in the "simple" form.) You may optionally also provide a title, which will be the title of the column in the header row, for example:

```
TSTable{
    options: { zebra:true },
    columns: [
        { key:"title", title:"Page Title" },
        { key:"id", title:"Page ID" },
        { key:"viewcount", title:"# of page views" }
      ],
    data: wiki.getpage("DekiScript/Reference").subpages
}
```

Page Title	Page ID	# of page views
DekiScript Syntax	9748	209
DekiScript Operators	9268	146

Page Title	Page ID	# of page views
Wiki Functions and Variables	10432	584
DekiScript Remote Function Calls	11206	143
DekiScript Statements	11333	187
DekiScript Literals	9431	244
HTML Page Composition Model	7358	193
DekiScript HTML/XML Statements	8847	279
DekiScript Functions and Variables	6479	656
JavaScript Events & Messages (JEM)	25589	738
XPath	33097	108

If **title** is omitted then the **key** value will be used as the title of the column. If the **key** is surrounded in parentheses, then it will be evaluated as a DekiScript expression for each row of **data**. In this case, you definitely want to provide a separate title, unless you want a really ugly column header. Evaluated DekiScript expression are described more in the next section, but here's a small example to illustrate:

```
TSTable{
    options: { zebra:true },
    columns: [
        { key:"title", title:"Page Title" },
        { key:"id", title:"Page ID" },
        { key:"(#$.comments)", title:"Number of comments" }
      ],
    data: wiki.getpage("DekiScript/Reference").subpages
}
```

Page Title	Page ID	Number of comments
DekiScript Syntax	9748	0
DekiScript Operators	9268	2
Wiki Functions and Variables	10432	2
DekiScript Remote Function Calls	11206	0
DekiScript Statements	11333	0
DekiScript Literals	9431	0

Page Title	Page ID	Number of comments
HTML Page Composition Model	7358	0
DekiScript HTML/XML Statements	8847	3
DekiScript Functions and Variables	6479	0
JavaScript Events & Messages (JEM)	25589	2
XPath	33097	1

"Width" and "Style"

You can control the styling of the column with the **width** and **style** elements, which specify the values of the **width** and**style** attributes of the <td> tag:

```
TSTable{
    options: { width:"100%", zebra:true },
    columns: [
        { key:"Title", width:"50%" },
        { key:"Path", width:"50%", style:"font-size:x-small" }
      ],
    data: wiki.getpage("DekiScript/Reference").subpages
}
```

Title	Path
DekiScript Syntax	en/docs/DekiScript/Reference/DekiScript_Syntax
DekiScript Operators	en/docs/DekiScript/Reference/DekiScript_Operators
Wiki Functions and Variables	en/docs/DekiScript/Reference/Wiki_Functions_and_Variables
DekiScript Remote Function Calls	en/docs/DekiScript/Reference/ DekiScript_Remote_Function_Calls
DekiScript Statements	en/docs/DekiScript/Reference/DekiScript_Statements
DekiScript Literals	en/docs/DekiScript/Reference/DekiScript_Literals
HTML Page Composition Model	en/docs/DekiScript/Reference/ HTML_Page_Composition_Model
DekiScript HTML/XML Statements	en/docs/DekiScript/Reference/ DekiScript_HTML//XML_Statements
DekiScript Functions and Variables	en/docs/DekiScript/Reference/ DekiScript_Functions_and_Variables

Title	Path
JavaScript Events & Messages (JEM)	en/docs/DekiScript/Reference/JEM
XPath	en/docs/DekiScript/Reference/XPath

Style, if enclosed in parentheses, will be evaluated as a DekiScript expression, which makes it possible to style each cell according to the actual values of the row. So, while again deferring detailed discussion of evaluated expressions, here's a quick example that puts the title in bold if the page has comments:

```
TSTable{
    options: { zebra:true },
    columns: [
        { key:"Title", style:"(#$.comments ? 'font-weight:bold' : '')" },
        { key:"Path" }
      ],
    data: wiki.getpage("DekiScript/Reference").subpages
};
```

Title	Path
DekiScript Syntax	en/docs/DekiScript/Reference/DekiScript_Syntax
DekiScript Operators	en/docs/DekiScript/Reference/DekiScript_Operators
Wiki Functions and Variables	en/docs/DekiScript/Reference/Wiki_Functions_and_Variables
DekiScript Remote Function Calls	en/docs/DekiScript/Reference/DekiScript_Remote_Function_Calls
DekiScript Statements	en/docs/DekiScript/Reference/DekiScript_Statements
DekiScript Literals	en/docs/DekiScript/Reference/DekiScript_Literals
HTML Page Composition Model	en/docs/DekiScript/Reference/HTML_Page_Composition_Model
DekiScript HTML/XML Statements	en/docs/DekiScript/Reference/DekiScript_HTML//XML_Statements
DekiScript Functions and Variables	en/docs/DekiScript/Reference/DekiScript_Functions_and_Variables
JavaScript Events & Messages (JEM)	en/docs/DekiScript/Reference/JEM
XPath	en/docs/DekiScript/Reference/XPath

"Initial" and "Sorter"

Initial lets you select a column for use as the sort column when the table is first displayed. Set to "0" for ascending order and "1" for descending. Ultimately, this accomplishes the same thing as the **initial** key in the **options** argument, but is a little easier to use. If the **initial** key in **options** is also specified, that argument will override anything set in**columns**. So use one or the other, but not both.

Setting sorter to false lets you disable sorting on that column. In the future, sorter will provide access to additional functionality, but for now it's just for disabling a column.

Here's a simple example using both **initial** and **sorter**:

```
TSTable{
    options: { zebra:true },
    columns: [
        { key:"Title", initial:0 },  // ascending sort on this column
        { key:"Path", sorter:false } // no sort on this column
      ],
    data: wiki.getpage("DekiScript/Reference").subpages
};
```

Title	Path
DekiScript Syntax	en/docs/DekiScript/Reference/DekiScript_Syntax
DekiScript Operators	en/docs/DekiScript/Reference/DekiScript_Operators
Wiki Functions and Variables	en/docs/DekiScript/Reference/Wiki_Functions_and_Variables
DekiScript Remote Function Calls	en/docs/DekiScript/Reference/DekiScript_Remote_Function_Calls
DekiScript Statements	en/docs/DekiScript/Reference/DekiScript_Statements
DekiScript Literals	en/docs/DekiScript/Reference/DekiScript_Literals
HTML Page Composition Model	en/docs/DekiScript/Reference/HTML_Page_Composition_Model
DekiScript HTML/XML Statements	en/docs/DekiScript/Reference/DekiScript_HTML//XML_Statements
DekiScript Functions and Variables	en/docs/DekiScript/Reference/DekiScript_Functions_and_Variables
JavaScript Events & Messages (JEM)	en/docs/DekiScript/Reference/JEM
XPath	en/docs/DekiScript/Reference/XPath

Using Evaluated DekiScript Expressions

In several places, TSTable allows you to give it DekiScript expressions to evaluate; a bit of this functionality has been demonstrated in the examples above. Fields that allow this include the **rowstyle** element in **options**, the **key** and **style**elements in each **columns** element, and the second element of each **filter**. Let's discuss this a bit further.

Basics

For each of the above-mentioned elements, surrounding the string in parentheses will cause it to be evaluated as a DekiScript expression (**filter** expressions will *always* be evaluated, so the parentheses are optional). The parentheses go inside the string. Inside the expression the symbol **$** refers to "this", which is the map representing the data for the row, so you can accessing anything in the row.

Assume for all these examples that a list of wiki pages is provided as data (perhaps the results of a wiki.getsearch() call):

Argument	Example	Effect
options	rowstyle:"font-weight:bold"	used as-is, not evaluated
	rowstyle:"(#$.comments > 0 ? 'background-color:blue' : ''))"	For each row, if the page has comments then the entire row will have blue background
columns	key:"uri"	cell contents will equal the page uri (data[n].uri)
	key:"(#$.comments)"	cell contents will evaluate to #(data[n].comments), or number of comments on the page
columns	style:"background-color:blue"	all elements in this column will have blue background
	style:"(#$.comments > 0 ? 'background-color:blue' : ''))"	for each row, if the page has comments then this column will have blue background
filter	["all items", "1"]	"1" evaluates to "true" always
	["items with comments", "#$.comments > 0"]	evaluates to true for all items with comments (no parentheses necessary here.)

How each row is processed

Understanding how each row is processed can help you take better advantage of this capability:

1. If present in **options**, **rowstyle** is processed first. A new element named **rowstyle** is added to the row data map containing the style for the row, which is DekiScript-evaluated if necessary.
2. All the **key** elements are processed. For each column *n* (starting at 0), an element **n** is added to the row data map containing the cell value. The cell value is DekiScript-evaluated if necessary. So, after this is done, there will be elements **0** .. **(#columns-1)** in the map containing the final values for each cell in the row.
3. All **style** elements are processed. For each column *n*, if a **style** element is specified, an element **style*n*** is added to the row data map containing the style for the cell. The **style** element is DekiScript-evaluated if necessary. Note that **style*n*** elements may have been pre-populated in the row data. They will be overwritten if the **style** element was included in the columns specification.
4. Finally, the filters are all DekiScript-evaluated.

In general, each stage has access to the row data added by the previous stages. So, the **filter** and **style** elements could, if you wanted, reference elements **0** .. **(#columns-1)**. Usually this is not desirable, but the capability is there if you want it.

Rules for safe "eval"ing

- You can't access any data other than the row data. So, if there's anything extra that you'll need to evaluate something, you have two options:
  - Add the extra data to the row. Then you can access it normally. Doesn't work well if you're using pre-existing wiki data structures for your data.
  - sub the values into your DekiScript expression. For example, let's say for column 0 you want to highlight any cell containing the value "0" to have a user-defined background color, stored in the DekiScript variable "highlight_color". You can provide the following evaluated style expression:
    `style:"($[0] == '0' ? 'background-color:" .. highlight_color .. "':'')"`
    This way, the highlight_color variable is evaluated in the main DekiScript context, and a constant is pasted into the eval script.
- You *can* access other DekiScript functions. You *cannot* access wiki functions; this is a DekiScript limitation.
- Remember that there are two ways to access a map item: dot notation (e.g.: `key: "(#$.comments)"`) and bracket notation (e.g.: `key: "(#$['comments'])"`). In theory, both are equivalent, and dot notation is cleaner when the key is a constant. However, I've encountered cases in evaluated DekiScript expressions where bracket notation worked but dot notation didn't. I haven't been able to isolate exactly when there's a problem, so be careful. If an eval expression is not working correctly, try using only bracket notation and see if it fixes it. If you want to be extremely conservative, always use bracket notation. Note that this only applies to the "this" variable ($).
- Add your evaluated elements in one at a time, and make sure it works. It's very hard to debug problems here if you're chasing a bunch at the same time.

Filters

The filters concept is pretty simple. If you want, you can have a pull-down control to select which rows of the table will be visible. The filter behavior is completely user-definable.

The argument is specified as a list of two-element lists, each of which specifies an option that will be available in the pull-down. The first element is the text that will appear in the pull-down, and the second is DekiScript that will be evaluated for each row to determine if that row should be displayed when this option is selected. The first option is always selected by default when the page is first rendered.

As mentioned above, a simple way to have a "show all" option is to specify a DekiScript expression of "1":

```
filter: [ [ "show all items", "1" ] ]
```

Other filters are simply evaluated DekiScript expressions that are applied to the row data, exactly the same way that **key**,**style**, and **rowstyle** elements are. Here, if the expression evaluates to "true", then that row will be displayed when the option is selected.

If we go back to the quick example given at the top, we had this filter:

```
filter: [
    [ "show all items", "1" ],
    [ "items with comments", "#$.comments>0" ],
    [ "items without comments", "#$.comments]==0" ]
]
```

The specifies three options. The first is the basic "show all". The next expression looks for the size of the "comments" map to be greater than 0. The last looks for the size of the 'comments' map to be equal to zero. And that's basically it.

If you're generating the data array, rather than passing a wiki data structure, it may sometimes pay to put in some "helper" fields into each row to make filter calculation easier, but that's up to you.

Technical Details

Customization

A number of styles in the script are customizable. My expectation is that you'll want to set the colors to coordinate with your theme. Once set in the template script, most of the styles are not customizable on a per-instance basis; I have been considering this functionality but haven't gotten around to it yet. If that's something you'd really want, let me know. Here are the colors you can customize:

- **thead_unsel_color**: this is the background color of the table headers that are *not* currently selected as sort columns.
- **thead_sel_color**: this is the background color of the table headers of columns that are currently selected as sort columns
- **zebra_color**: this is the background color of odd-numbered rows when the "zebra" option is enabled

Supported Data Types

TBD