Table of Contents

Homepage of hBrowseFramework **1**
General Information 1
Main Features 1
Latest Version (v1.2.3) 1
Quick start manual 1
Basics 1
Setting up the application 1
Application 2
Users 2
Mains 2
Subs 4
Setting up charts 6
Files setup 9
For developers 9
How does it work? 9
Model overview 10
Brief modules overview 10
lkfw.searchable.list plugin 11
lkfw.datatable plugin 11

Homepage of hBrowseFramework

General Information

The hBrowseFramework (aka Tasks Monitoring Web UI) is a generic monitoring tool designed to meet the needs of various communities. It is strongly configurable and easy to adjust and implement accordingly to a specific community needs. Each part of this software (dynamic tables, user selection etc.) is in fact a separate plugin which can be used separately from the main application. It was especially designed to meet the requirements of Atlas and CMS users as well as to use it as a bulked Ganga monitoring tool.

Main Features

- **Highly configurable environment** - Things like table columns, charts, data sources and many more can be set up in only one file
- **Bookmarking support** - every view has a proper representation in URL
- **History support** - not so common in ajax applications
- **Easy search** - Application provides easy way to search the tables data (part of jQuery datatables plugin)
- **Google charts support** - if you are familiar with google charts service you can easily add or modify charts
- **Independent View** - Application gets the data from ajax requests so using of different data sources is possible
- **3 main views** - user selection list, table of main content and table of sub content.

Latest Version (v1.2.3)

```
svn co http://svn.cern.ch/reps/ganga/tags/packages/hbrowse-1-2-3
```

Quick start manual

Basics

First thing to consider when setting up TMWebUI application is your data server. Main and in fact only requirement is your server to be able to serve data in a json format for 3 main views:

- User selection list
- Main content table (like jobs in Ganga)
- Sub-content table (like subjobs in Ganga)

Json data format is not important because special translation functions can be used inside the application. Of course, data that is in a right format is processed much quicker.

Setting up the application

To setup the application you need to modify *media/scripts/settings.js_example* file. Settings are split onto 3 categories:

- Application
- Users
- Mains
- Subs

Each category is represented by a javascript object (dictionary). Properties can be either string or number, boolean value or a function that returns strictly defined content. Below you can find properties reference for each category.

Application

Property	Description
userSelection	**/bool/** Display user selection page? If false, a placeholder user name has to be setup (eg. default, see below) (true/false)
jsonp	**/bool/** allow requests to other hosts (default: false)
pageTitle	**/string/** page title (example: 'Task Monitoring')
footerTxt	**/string/** footer text (example: 'jTaskMonitoring')
supportLnk	**/string/** link to support page (example: '#TaskMonitoringWebUI')
logoLnk	**/string/** link to page logo (example: 'media/images/atlaslogo.png')
usersListLbl	**/string/** Label of user list search field (example: 'Users List')
mainsLbl	**/string/** Name of mains content (example: 'Tasks')
subsLbL	**/string/** Name of subs content (example: 'Jobs')
modelDefaults	**/function/** Here You can set up model (data.js) default values, in most cases doesn't need to be changed. Function has to return the following values:
	user - **/string/** default user name (default: '')
	from - **/int/** timestamp (default: 0)
	till - **/int/** timestamp (default: 0)
	timeRange - **/string/** lastDay|last2Days|last3Days|lastWeek|last2Weeks|lastMonth (default: 'lastDay')
	refresh - **/int/** seconds (default: 0)
	tid - **/string/** eg. Job id in Gange nomenclature (default: '')
	p - **/int/** page number (default: 0)
	sorting - **/array/** (default: [])
	or - **/array/** opened table rows (default: [])
	uparam - **/array/** user defined params (for params that cannot be shared between use cases) (default: [])

Users

Property	Description
dataURL	**/string/** Users list URL for ajax request
dataURL_params	**/function/** (optional) Ajax request parameters Input: controller.Data object Output: {'<parameter_name>':<parameter_value>,...} (default: {})
searchLabel	**/string/** Label of the search field
translateData	**/function/** Translates ajax response onto searchable list plugin data format Input: json responce Output: [user1, user2, ...]

Mains

Property	Description
dataURL	**/string/** Mains URL for ajax request
dataURL_params	**/function/** (optional) Ajax request parameters Input: controller.Data object Output: {'<parameter_name>':<parameter_value>,...} (default: {})
expandableRows	**/bool/** If TRUE, rows will expand after clicking '+'

multipleER	**/bool/** If TRUE, multiple rows can be expanded
expandData	**/object/** Definition of data that will be displayed after row expansion, expandData object has the following structure:
	dataURL - **/string/** URL for ajax request
	dataURL_params - **/function/** Ajax request parameters Input: controller.Data object and current row data Output: {'<parameter_name>':<parameter_value>,...} (default: {})
	dataFunction - **/Function/** Function, definition of data that will be displayed after row expansion Input: - rowDataSet - clicked row data (from ajax datatable response) - jsonDataSet - data extracted from ajax response Output: [['properties',[[(property_name),'(property_value)'],...]], ['table',{ 'tblLabels':[(label1),(label2),...], 'tblData':[[(row1value1),(row1value2),...],[(row2value1),(row2value2),...],...] }], ['html','(custom_html)'], ['charts',[<chart_obj1>,<chart_obj2>]]] You can skip each of the output parameters ('properties', 'html', 'table' or 'charts' output sections), expanded row will keep to the order set up in this output array
sorting	**/array/** sets up column sorting, [,], sorting_direction='desc'|'asc' (default: [1,'desc'])
iDisplayLength	**/int/** Number of rows to display on single page (default: 25)
tblLabels	**/array/** column labels (example: ['monitorTaskId','Num of Jobs','Pending','Running','Successful','Failed','Unknown','Graphically'])
aoColumns	**/array/** (optional) columns options, see http://www.datatables.net/usage/columns
getDataArray	**/function/** function: extracting array of data (for dataTable) form Ajax response Example: - Ajax response: {'user_taskstable':[{col_val1, col_val2, ...}, ...]} - Required function: function(data) { return data.user_taskstable; }
translateData	**/function/** function, translates ajax response onto dataTables plugin data format, here you can setup you table content and a links to the subcontent, to do this just add an html "a" tag with class "drilldown" to the cell. Output: [[col_val1, col_val2, ...], ...]

drillDownHandler	**/function/** **Replaced setupUserParams**. It is executed every time someone clicks a table cell with a.drilldown html tag in it. Main purpose of the function is to indicate tid and (optionary) uparam parameters from the Data object (uparam allows to setup additional parameters to tid that would define a sub table ajax request). This allows to properly display subs table.
	Input:
	- Data - application Data object
	- el - clicked jQuery element
	- rowIndex - index of the clicked row
	Output: {
	'uparam':[],
	'tid':
	}

filters	**/array/** It's an array of objects, each object represents different filter and all of the filters values are stored in a Controller.Data.filters object and you can use them (eg in dataURL_params functions) by doing Data.filters.{filter_urlVariable}. The filter object consists of the following elements:

	label - **/string/** filter label /displayed above the input field/ (default: '')
	urlVariable - **/string/** lower cased, no spaces, no special characters /used in url and inside Controller.Data object/ (default: 'filter')
	fieldType - **/string/** accept 3 kinds of input types ('text' | 'select' | 'date') (default: 'text')
	value - **/string/** Initial value for the filter (default: '')

	options - **/obejct/** there are several options that you can setup depending on *fieldType* and needs, here is the example:
	{
	'dataURL':'/dashboard/request.py/inittaskprod?data=sites', // (optional) String, works only for select fieldType
	'dataURL_params':function(Data){return {};}, // (optional) Here you can setup request parameters
	// Function translates model or ajax data onto simple elements array
	// Input: data - data represents Data.mem object or ajax response depending on whether dataURL exists or not
	// Output: [['el1','el1 label'],['el2','el2 label'], ...] - Can also be defined as a static list (when you don't want to
	// load the data from url nor using Data.mem object)
	'translateData': function(data) {
	/* Example useage:
	var sitesArr = data.basicData;
	var output = [['','Off']];
	
	for (var i=0;i<sitesArr.length;i++) {
	output.push([sitesArr[i].SITENAME,sitesArr[i].SITENAME]);
	}
	
	return output;
	*/
	},
	// On and Off optional functions are executed when filters submit is clicked
	// On is executed when field has value other then empty string ("")
	// Otherwise Off is executed
	'On':function(Data) { // Data is a Controller.Data object
	$('#from,#till,#timeRange').attr('disabled',true);
	},
	'Off':function(Data) { // Data is a Controller.Data object
	$('#from,#till,#timeRange').attr('disabled',false);
	}
	}

charts	**/array/** (optional) allow to insert charts into charts tab
topTableCharts	**/array/** (optional) allow to insert charts on top of the dataTable

Subs

Property	Description
dataURL	**/string/** Mains URL for ajax request
dataURL_params	**/function/** (optional) Ajax request parameters Input: controller.Data Output: {'<parameter_name>':<parameter_value>,...} (default: {})
expandableRows	**/bool/** If TRUE, rows will expand after clicking '+'
multipleER	**/bool/** If TRUE, multiple rows can be expanded
expandData	**/object/** Definition of data that will be displayed after row expansion, expandData object

	has the following structure:
	dataURL - **/string/** URL for ajax request
	dataURL_params - **/function/** (optional) Ajax request parameters Input: controller.Data object and current row data Output: {'<parameter_name>':<parameter_value>,...} (default: {})
	dataFunction - **/Function/** Function, definition of data that will be displayed after row expansion Input: - rowDataSet - clicked row data (from ajax datatable response) - jsonDataSet - data extracted from ajax response Output: [['properties',[[(property_name),'(property_value)'],...]], ['table',{ 'tblLabels':[(label1),(label2),...], 'tblData':[[(row1value1),(row1value2),...],[(row2value1),(row2value2),...],...] }], ['html','(custom_html)'], ['charts',[<chart_obj1>,<chart_obj2>]]] You can skip each of the output parameters ('properties', 'html', 'table' or 'charts' output sections), expanded row will keep to the order set up in this output array
sorting	**/array/** sets up column sorting, [,], sorting_direction='desc'|'asc' (default: [1,'desc'])
iDisplayLength	**/int/** Number of rows to display on single page (default: 25)
tblLabels	**/array/** column labels (example: ['monitorTaskId','Num of Jobs','Pending','Running','Successful','Failed','Unknown','Graphically'])
aoColumns	**/array/** (optional) columns options, see http://www.datatables.net/usage/columns
getDataArray	**/function/** function: extracting array of data (for dataTable) form Ajax response Example: - Ajax response: {'user_taskstable':[{col_val1, col_val2, ...}, ...]} - Required function: function(data) { return data.user_taskstable; }
translateData	**/function/** function, translates ajax response onto dataTables plugin data format Output: [[col_val1, col_val2, ...], ...]
filters	**/array/** It's an array of objects, each object represents different filter and all of the filters values are stored in a Controller.Data.filters object and you can use them (eg in dataURL_params functions) by doing Data.filters.{filter_urlVariable}. The filter object consists of the following elements:
	label - **/string/** filter label /displayed above the input field/ (default: '')
	urlVariable - **/string/** lower cased, no spaces, no special characters /used in url and inside Controller.Data object/ (default: 'filter')
	fieldType - **/string/** accept 3 kinds of input types ('text' | 'select' | 'date') (default: 'text')
	value - **/string/** Initial value for the filter (default: '')

	options - **/obejct/** there are several options that you can setup depending on *fieldType* and needs, here is the example:
	{
	'dataURL':'/dashboard/request.py/inittaskprod?data=sites', // (optional) String, works only for select fieldType
	'dataURL_params':function(Data){return {};}, // (optional) Here you can setup request parameters
	// Function translates model or ajax data onto simple elements array
	// Input: data - data represents Data.mem object or ajax response depending on whether dataURL exists or not
	// Output: [['el1','el1 label'],['el2','el2 label'], ...] - Can also be defined as a static list (when you don't want to

	// load the data from url nor using Data.mem object)
	'translateData': function(data) {
	/* Example useage:
	var sitesArr = data.basicData;
	var output = [['','Off']];
	
	for (var i=0;i<sitesArr.length;i++) {
	output.push([sitesArr[i].SITENAME,sitesArr[i].SITENAME]);
	}
	
	return output;
	*/
	},
	// On and Off optional functions are executed when filters submit is clicked
	// On is executed when field has value other then empty string ("")
	// Otherwise Off is executed
	'On':function(Data) { // Data is a Controller.Data object
	$('#from,#till,#timeRange').attr('disabled',true);
	},
	'Off':function(Data) { // Data is a Controller.Data object
	$('#from,#till,#timeRange').attr('disabled',false);
	}
	}

charts	**/array/** (optional) allow to insert charts into charts tab
topTableCharts	**/array/** (optional) allow to insert charts on top of the dataTable

Setting up charts

Charts are based on google charts service and highcharts jQuery plotting library. The types of charts to draw can be toggled using *type* property (gchart for google charts and hchart for highcharts). In case when one want to use google charts to display charts on the page function *translateData* will have to return object with complete google charts parameter set (see the first example and google charts web page: http://code.google.com/apis/chart/). Google charts can be created using Chart Wizard (http://code.google.com/apis/chart/docs/chart_wizard.html), this will give you the necessary parameters to fill. In case when using highcharts the *translateData* function would have to return highcharts options object (instructions for using highcharts can be found here: http://www.highcharts.com/). When composing highcharts options object you don't have to specify options.chart.renderTo option, it will be replaced anyway. Below you can find example charts settings:

```
'charts': [
    {
        'name':'Status Overview',
        'type':'gchart', // (gchart|hchart),
        'onDemand':true,
        'dataURL': 'http://pcadc01.cern.ch/client/chartdata',
        'dataURL_params': function(Data) { return {}; },
        // translates data onto requires format:
        // {"chd":"t:60,40","chl":"Hello|World"} or highcharts options object (http://www.highcha
        'translateData':function(dataJSON) {
            output = {
                'chtt':'Example Chart',
                'cht':'p3',
                'chs':'600x350',
                'chd':dataJSON.chd,
                'chl':dataJSON.chl
            };
            return output;
        }
```

```
    }
]

'charts': [
    {
        'name':'Status Overview',
        'type':'gchart', // (gchart|hchart)
        'onDemand':false,
        // translates data onto requires format:
        // {"chd":"t:60,40","chl":"Hello|World"} or highcharts options object (http://www.highcha
        'translateData':function(dataMem) {
            var data = dataMem.subs.data;
            var dataLen = data.length;

            var obj = {
                'statuses':['S','F','R','PR','P','U'],
                'statusesCnt':[0,0,0,0,0,0],
                'statusesLbl':['Succesfull','Failed','Running','NotCompleted','Pending','Unknown'
            };

            for (var i=0; i<dataLen; i++) {
                var row = data[i];
                for (var j=0; j<obj.statuses.length; j++) {
                    if (row.STATUS == obj.statuses[j]) obj.statusesCnt[j]++;
                }
            }

            for (var j=0; j<obj.statusesLbl.length; j++) {
                if (obj.statusesCnt[j] > 0) obj.statusesLbl[j] += ' ('+obj.statusesCnt[j]+')';
                else obj.statusesLbl[j] = '';
            }

            var output = {
                "chd":"t:"+obj.statusesCnt.join(','),
                "chl":obj.statusesLbl.join('|'),
                'chtt':'Status Overview',
                'cht':'p3',
                'chs':'600x250',
                'chco':'59D118|C50000|3072F3|BB72F3|FF9900|C2BDDD'
            };
            return output;
        }
    },
    {
        'name':'Graphical representation',
        'type':'hchart', // (gchart|hchart)
        'onDemand':false,
        // translates data onto requires format:
        // {"chd":"t:60,40","chl":"Hello|World"} or highcharts options object (http://www.highcha
        'translateData':function(dataMem) {
            var data = dataMem.mains.data;
            var dataLen = data.length;
            if (dataLen > 25) dataLen = 25;

            var obj = {
                'statuses':['SUCCESS','FAILED','RUNNING','PENDING','UNKNOWN'],
                'statusesCnt':[[],[],[],[],[]],
                'taskIDs':[],
                'statusesLbl':['Succesfull','Failed','Running','Pending','Unknown'],
                'statusesColors':['#59D118','#C50000','#3072F3','#FF9900','#C2BDDD']
            };

            for (var i=0; i<dataLen; i++) {
                var row = data[i];
                for (var j=0; j<obj.statuses.length; j++) {
                    obj.statusesCnt[j].push(row[obj.statuses[j]]);
                }
```

```
            obj.taskIDs.push(row.TASKMONID.slice(0,10)+'*'+row.TASKMONID.substr(-8));
        }

        var series = [];

        for (var i=0;i<obj.statuses.length;i++) {
            series.push({
                name:obj.statusesLbl[i],
                data:obj.statusesCnt[i],
                color:obj.statusesColors[i]
            });
        }

        output = {
            chart: {
                height:400,
                width:500,
                renderTo: 'container',
                defaultSeriesType: 'bar',
                backgroundColor:'#eeeeee',
                borderColor:'#aaaaaa',
                borderWidth:2
            },
            title: {
                text: 'Graphical representation'
            },
            xAxis: {
                categories: obj.taskIDs,
                title: { text: 'Taks IDs' }
            },
            yAxis: {
                min: 0,
                title: { text: 'Jobs number' }
            },
            legend: {
                backgroundColor: '#FFFFFF',
                reversed: true
            },
            tooltip: {
                formatter: function() { return ''+this.series.name +': '+ this.y +''; }
            },
            plotOptions: {
                series: {
                    stacking: 'normal'
                }
            },
            series: series
        }

        return output;
    }
}
]
```

Chart object description:

Property	Description
name	**/string/** Obligatory chart name
type	**/string/** (gchart|hchart)
onDemand	**/bool/** If true chart will load data and draw after clicking a button
dataURL	**/string/** (optional) URL for ajax charts data request
dataURL_params	**/function/** (optional) Ajax request parameters Input: data object Output: {":,...}

translateData	**/function/** defines google charts dynamic parameters input: , Data.mem object or AJAX response output example: {"chd":"t:60,40","chl":"Hello|World"}

As a data source for charts both local stored data (data used to generate a tables) and data from ajax request can be used. When you want to use ajax data you will have to specify *dataURL* and *dataURL_params* parameters, in this case input for *translateData* function will be object extracted from ajax response, otherwise Data.mem (see data.js file for Data.mem object structure) object will serve as an input for this function.

Files setup

To enable easy update from svn few files and directories was set up as an example files:

- index.html_example
- media/scripts/settings.js_example
- media/css_example

_example string should be removed from file/directory name for each of above elements to make application working.

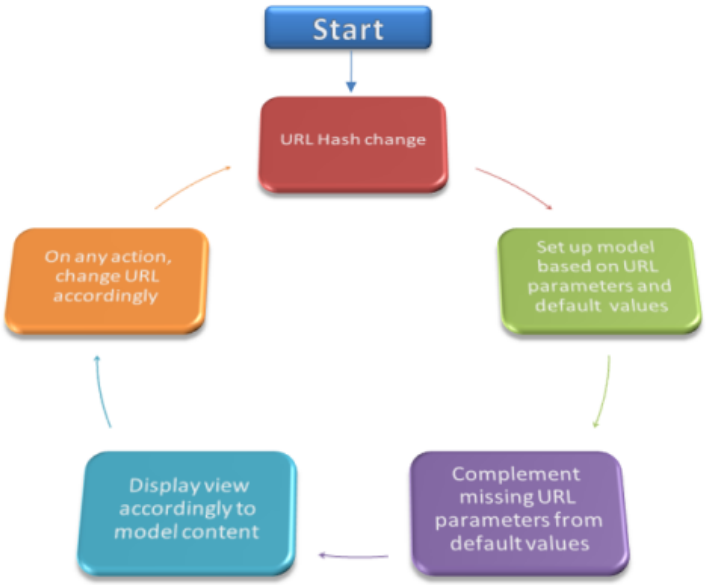
For developers

How does it work?

Application works in a loop of mutual dependency between 3 main aspects:

- Bookmarking URLs
- Data model
- View

Basically when we want to change the content of the page we will first modify the URL. The change like this will be noticed by an application and, based on URL, internal data model will be updated. At the end, view will be generated based on data stored in a model:



Model overview

All the application data are stored inside Data model which is defined in data.js file. Settings lets you access the data model in various places so it's important to understand it's structure. Data stored inside data model are split onto 2 parts, operating data and cached data. Operating data are used to display proper informations on the screen and they consists informations like user name, time periods, interface settings etc. Cached data are used in various situations like charts drawing or additional requests etc. Here is the structure with default data:

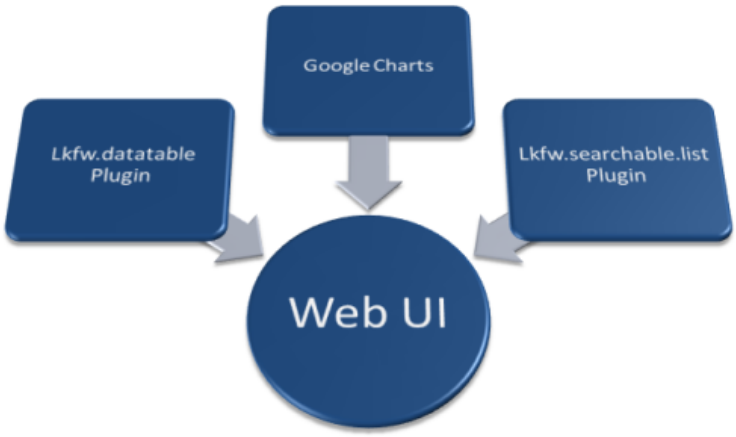
```
var Data = {
    user: '',
    from: 0,
    till: 0,
    timeRange: 'lastDay',
    refresh: 0,
    tid: '',
    p: 1,
    sorting: [],
    or: [],
    uparam: [],

    noreload: false,

    mem: {
        users: Array(),
        mains: {
            user: '',
            timestamp: 0,
            data: Array()
        },
        subs: {
            user: '',
            tid: '',
            timestamp: 0,
            data: Array()
        }
    }
}
```

Brief modules overview

Application consists of 3 main modules that are used in presentation layer.



- **lkfw.datatable plugin** - jQuery datatables plugin overlay that expands it's original functionality.
- **lkfw.searchable.list plugin** - jQuery livesearch plugin overlay fitted to the needs of the monitoring application.

- **Google Charts** - Task Monitoring Web UI uses google's chart service to draw data plots.

First two modules can be used separately in other application as a simple jQuery plugin. All you need to is to copy a component file and associated libraries.

lkfw.searchable.list plugin

This plugin can be used outside of the parent application as a separate jQuery plugin.

Required libraries:

- jquery-1.4.2.min.js
- quicksilver.js
- jquery.livesearch.js

Options:

Option	Description
listId	**/string/** Id prefix of the searchable list html element
srchFieldId	**/string/** Id prefix of the html input search field
items	**/array/** Simple array of searchable items
srchFldLbl	**/string/** Label of the input search field

Initialization example:

```
$('#someDiv').lkfw_searchableList({
    listId: 'users',
    srchFieldId: 'usersSrch',
    items: ['user1','user2','user3'],
    srchFldLbl: 'Search'
});
```

lkfw.datatable plugin

This plugin can be used outside of the parent application as a separate jQuery plugin.

Required libraries:

- jquery-1.4.2.min.js
- jquery.dataTables.min.js

Options:

Option	Description
dTable	**/array/** Array that stores all dataTable objects (tables), if you want to create more that one you can r anyone of them by index number.
tableId	**/string/** Data Table html id
items	**/array/** Array of table content. In practice it's an array of arrays, each sub-array represents a table ro Sub-arrays lengths have to be equal to column labels (headers) number.
tblLabels	**/array/** Simple array of column labels (headers).
dataTable	**/dictionary/** Dictionary of dataTables jQuery plugin specific properties (see: http://www.datatables.net/usage/options)
expandableRows	**/bool/** True or false depending on either you want rows to expand or not.
multipleER	**/bool/** True or false depending on either you want multiple rows to expand or only one.
rowsToExpand	**/array/** Array of rows ids. All of the rows from this list will be expanded on initialization.

sorting	**/array/** aaSorting dataTables option (see: http://www.datatables.net/usage/options)

fnERContent	**/function/** This function provides data for expanded rows. As an input, it takes table data row id (id "as provided" in *items* property). It should return the following structure:
	{
	'properties':[['property1','value1'],['property2','value2'],...],
	'table':{'tblLabels':['colLabel1','colLabel2','colLabel3',...],'tblData':[['rowData1','rowData2','rowDat
	'html':'some custom html...'
	}
	The content of expandable row can differ according to what is needed. Data Tables and property-val supported but also, thru custom html, such elements like google charts and many more.

fnERClose	**/function/** This method is called on expanded row close event and it's used to do some post processi operations. As an input, it takes table data row id (id are sorted "as provided" in *items* property)
fnContentChange	**/function/** This method is called on any data table content change caused by, for example, sorting ch change etc.
fnTableSorting	**/function/** Additional method to handle sorting changes events. In the jTaskMonitoring application accordingly setup URL hash.

Initialization example:

```
$('#someDiv').lkfw_dataTable({
    dTable: someVariable,
    tableId: 'mains',
    expandableRows: true,
    multipleER: false,
    items: [['row0val1','row0val2','row0val3'],['row1val1','row1val2','row1val3']],
    tblLabels: ['colname1','colname2','colname3'],
    rowsToExpand: [],
    sorting: [1,'desc'],
    fnERContent:function(dataID){ return thisRef.expand_click(dataID) },
    fnContentChange: function(el) { thisRef.mainsTableContent_change(el) },
    fnERClose: function(dataID) { thisRef.erClose_click(dataID) },
    fnTableSorting: function(el) { thisRef.tableSorting_click(el,thisRef.mainsTable[0]) },
    dataTable: {
        iDisplayLength: 25,
        sPaginationType: "input",
        bLengthChange: false
    }
});
```

-- LukaszKokoszkiewicz - 03-Dec-2010

This topic: ArdaGrid > TaskMonitoringWebUI
Topic revision: r26 - 2011-05-02 - unknown