jqGrid

a jQuery Plugin

by

Tony Tomov

Version 3.2
Released July 2008

Contents

jqGrid 3
Acknowledgements 4
What's New? 5
Version 3.2 5
Version 3.1 7
Introduction 9
Requirements 9
Do I need to Pay for jqGrid? 10
Installation 10
How it Works 12
Tutorial: Creating Your First Grid 14
The Data 14
The HTML 14
The Server-side File 18
PHP and MySQL 19
COOP Example 20
Data Types 23
XML Data 24
XML String 28
JSON Data 28
JSON String 33
Array Data 33
User Data 36
Basic Grid 38
Properties 38
colModel Properties 42
Height and Width 44
Obsolete Properties 46
Events 46
Methods 47
Available methods (as of version 3.2) 48
Advanced Methods 51
Obsolete Methods 52
Pager (Navigation bar) 53
Properties 53
Events 54
Methods 54
Custom Buttons 54
Form Editing 57
Properties 57
Searching 57
Add Row 59
Edit Row 59
Delete Row 62
Inline Editing 65
Properties 65
Methods 67
Example 69
Multi-Selecting Items 71
Subgrids 73
A Subgrid 73
Defining a SubGrid 73
A Grid as Subgrid 77

Defining a Grid as a subGrid 77
Master/Detail Grids 79
Defining Master/Details Grids 79
User Modules 81
Posting Data 81

jqGrid

jqGrid is an Ajax-enabled JavaScript control that provides solutions for representing and manipulating tabular data on the web. Since the grid is a client-side solution loading data dynamically through Ajax callbacks, it can be integrated with any server-side technology, including PHP, ASP, Java Servlets, JSP, ColdFusion, and Perl.

jqGrid uses a jQuery Java Script Library and is written as plugin for that package. For more information on jQuery, please refer to the jQuery web site.

jqGrid's Home page can be found here.

Working examples of jqGrid, with explanations, can be found here.

I started the idea when I needed an easy way to represent database information in my project. The first requirement was speed and the second, independence from server-side technology and the database backend.

-- Tony

Acknowledgements

Author

The author of jqGrid and its accompanying documentation is Tony Tomov. Suggestions for enhancements, reports of bugs, and requests for help can be made on jqGrid's community forum.

Editor

This documentation is edited and maintained by Reg Brehaut. Reports of errors or confusing expressions, and suggestions for inclusion or enhancement can be made on the forum or directly to Reg.

Special Thanks

Special thanks to Brice Burgess for the invaluable advice, writing the shrinkToFit feature and, of course, providing the excellent jqModal plugin used by jqGrid.

Contributors

Paul Tiseo contributed grid.postext.js.

Participants in the jqGrid forum have contributed significantly by asking questions for clarification, by pointing out problems, and by suggesting ideas for enhancements.

Software

This documentation has been produced using West Wind's HTML Help Builder, quite possibly the best product of its kind on the market. Check out how it can help you document your system, today.

What's New?

As new versions are released, Release Notes will be posted here.

Version 3.2

Released 2008-07-15

IMPORTANT: Required Actions

1. Some methods, parameters or options have been removed in this release; please refer to Obsolete Properties and Obsolete Methods to see what has been removed and what to use instead.
2. the distribution of code across modules has been enhanced to allow for future expansion while still keeping the basic package as small as possible. Please review the contents of jquery.jqGrid.js to see what you may need to add or change. See Installation.
3. additions have been made to the CSS files (in themes); either replace the previous versions with what comes with this release or, if you have incorporated the jqGrid CSS settings into your own CSS files, review and revise the sections noted here:

```
/* Pager *//
...
/* End Pager *//
```

and

```
/*Subgrid  text mode*//
...
/* End Subgrid *//
```

4. Previous versions of jqModal are not compatible with jQuery 1.2.6; please replace your copy of jqModal.js with the updated version included in this release.

Bug Fixes

- corrected bug in *delRowData* and *setRowData* methods when using two or more grids and trying to delete/update a row with the same id
- corrected bug with url parameter in *editGridRow* (now the url can be changed dynamically)
- corrected bug in modal window
- corrected bug in formedit with class names
- corrected bug when the [Enter] key is pressed in *editGridRow*
- corrected bug in Safari 3 when resizing columns
- corrected bug with header columns and data columns position
- corrected misaligment between table header and table rows bug in IE (this reduces the header width by 1px and previously tight columns may now cut off the header text slightly)
- corrected bug in *hideCol* and *showCol* methods in IE
- corrected bug in IE when inline edit of element of type select
- corrected bug in safari when try to use pager in grid as subgrid
- corrected bug in navigator when try to attach buttons in grid as subgrid
- corrected mouseover bug when using grid as subgrid
- corrected bug when *shrinkToFit* parameter is set to false and the grid is resized

- corrected jquery.jqGrid.js loader bug
- corrected bug in *delGridRow* method when using modal for first time
- corrected bug in getRowData method to handle properly empty fields
- corrected bug in inline edit when restoring non-editable cells
- corrected bug that occured when starting to resize a column but the mouse is not moved
- corrected bug in subgrid when the data in a cell is wider than the width of the cell.

Additions & Changes

grid base

- added *afterInsertRow*(rowid, rowdata) event - fires after every inserted row. Rowid is the id of inserted row. Rowdata is array of the inserted values. The array is of type name:value, where the name is the name from colModel.
- added *setLabel*(colname,newlabel, sattr) method set a new label to the header. We can set attributes and classes (sattr). If sattr is a string, we add a class using the using the jQuery addClass. If sattr is array we set css properties via jQuery css
- added *gridComplete* event - fires after all the data is loaded into the grid and all other processes are complete
- added *onSelectAll*(array of the selected ids) fires (if defined) when multiselect is true and you click on the header checkbox. Parameter passed to this event is an array of selected rows. If the rows are unselected, the array is empty.
- added *clearGridData* - clear the currently loaded data from grid
- added *loadError* - fires when a error in ajax request
- added *loadBeforeSend* - fires before sending the request
- added method *setCell*. This very useful method can change the content of particular cell and can set class or style properties.
- added property *hiddengrid*. If set to true the grid initially is hidden. The data is not loaded and only the caption layer is shown. When click to show grid the data is loaded and grid is shown. From this point we have a regular grid.
- added property *loadui* - "disable", "enable", "block"
- *onPaging*(which button) fires when a pager button is clicked and before populating the data; accepts which button is clicked: first, last, prev, next
- *resetSelection* method now resets the header checkbox, if mutiselect is true
- hidden fields are no longer included in the calculation of the grid width
- the grid should be set only with table element and class. The cellSpacing, cellPadding and border attributes are added automatically.
- hideCol and showCol can accept an array of data as parameter. Example: hideCol(["name1","name2"]) will hide the name1 and name2 columns. Also the method can accept a single string as parameter - i.e. hideCol("name1").The same applies to the showCol method
- the appropriate sort image now appears in the column heading when the grid is initially loaded and the sortname is set.

formedit

- added method *navButtonAdd* - add a custom button to pager
- added method *GridToForm*
- added method *FormToGrid*
- added property *editrules*: {edithidden:true, required:true(false), number:true(false), minValue:val, maxValue:val}
- *editGridRow* can now accept default values in input text field, when action is add
- *searchGrid* now searches not by name but by the index name, if any

inlineedit

- added *onerrorfunc* as the 6th parameter in saveRow to handle errors returned from the server; also can be passed from editRow (where it is the new 8th parameter) when using the [Enter] key to trigger the save.

Miscellaneous

- improved performance in json and xml data reading when using zebra-striping
- improved performance (by 50%) in addRowData and setRowData methods
- improved performance of reading data when browser is IE or Mozilla (related to corrected misaligment bug)
- formedit (modal windows) are now compatible with jQuery 1.2.6
- formedit is now compatible with other JS libraries, like Prototype

Version 3.1

Released 2008-04-05

Bug Fixes

- grid width bug when hidden fields are set and width of grid too.
- fixed bug in setSortName method.
- fixed bug in addRowData method when add at top of the grid and grid has no data.
- CSS bug when editing input type=text field in inline edit module
- added missed functions in editGridRow method
- fixed bugs with events names in formedit module

IMPORTANT: Some methods are to be removed in the next release; please refer to Replaced Methods to see which will be removed and what to use instead.

Additions

- added getGridParam method - this method requested parameters from option array of the grid. If the parameter is empty - the entry options are get
- added setGridParam method - set a particular parameter. Note - for some parameters to take effect a trigger("reloadGrid") should be executed. Note that with this method we can override events like onSelectRow and etc.
- added onPaging event - this event fires after click on page button and before data population
- added resetSelection method - this method reset (unselect) the selected row(s)
- added option toolbar add at bottom or at top of the gridbody div element where we can put custom html content.
- added option userData in options array. Whit this we get user defined data from the response and use it later. To use this an additional option userdata in xmlReader and jsonReader is added. (thanks to Paul Tiseo). For more information refer to xml file and JSON Data
- added option postData. This array is passed directly to the url options - ajax request (thanks to Paul Tiseo). Refer to API Methods for manipulating postData array.
- added scope for easy translation when the grid is used multiple times in the application. That mean that the translation strings can be called only once and not every time when the grid is constructed. The scope is $.jgrid.defaults To use this you need to simply do

$.extend($.jgrid.defaults,{ recordtext: "My record text", loadtext: "Process text",...}) only once. Note that we can override all other parameters. There are other additions for form manipulation module - $.jgrid.search - for the search method, $.jgrid.edit - for editing and adding method, $.jgrid.del - for deleting method, $.jgrid.nav - for the navigator .

Introduction

This documentation is also available as a pdf file.

This documentation assumes that you are familiar with concepts like web server, database server and scripting programming language. Using this documentation will be easier if you have already installed this software.

Conventions

- Names of things -- modules, options, parameters or settings, etc. -- are in *italics*
- Values are shown in a different font: e.g., "change the value of the associated *include* from true to false."
- Names of keys (e.g., Enter) appear in square brackets in the text, e.g., press [Enter]
- for readability, code samples are shown with spaces separating elements e.g., *name: value*; in practice however, spaces are not significant and *name : value*, *name:value* and *name :value* are treated all the same

A reminder: javascript is case-sensitive, so *subGrid: true* is not the same as *subgrid: true*

Errors and Omissions

It is almost inevitable that some errors will creep into this documentation; please report any you find to the editor.

Suggestions for expanded or new topics, or contributions of examples are also very welcome.

Requirements

At the very least, you will need:

- jqGrid plugin,
- jQuery library, version 1.1.4 or later, and
- a web browser

To manipulate and represent local (static) data – i.e. array data, data stored in an xml file, or data stored in a JSON file – that's all you need.

But the primary purpose of jqGrid is to manipulate and represent dynamic data over the web, and for this you will also need

- a web server (e.g., IIS, Apache, Tomcat),
- a database backend (e.g., Postgre SQL, Oracle, MSSQL, MySQL), and
- a server-side scripting language (e.g., PHP, ASP)

Do I need to Pay for jqGrid?

No. This package is free, distributed under GPL and MIT, so you don't need to pay. Just follow the GPL rules and everybody will be happy. I really was needing some way to contribute to the open source community, and I hope this is just the beginning.

If you really love jqGrid and wish to make a donation, you can contact me (Tony) at tony@trirand.com.

Installation

First you need to download the jQuery JavaScript library. This library can be downloaded from www.jquery.com. Please download the latest stable version of jQuery library and not a development version.

You need to download the jqGrid plugin.

Create a directory on your web server, so that you can access it: http://myserver/mydir/, where mydir is the name that you have created.

Place the jQuery library in that directory; unpack the jqGrid.zip in the same directory. You should have this directory structure:

- jquery.js
- jquery.jqGrid.js
- js
  - grid.basic.js
  - grid.custom.js
  - grid.formedit.js
  - grid.inlinedit.js
  - grid.postext.js
  - grid.subgrid.js
  - jqDnR.js
  - jqModal.js
- min
  - grid.basic-min.js
  - grid.custom-min.js
  - grid.formedit-min.js
  - grid.inlinedit-min.js
  - grid.postext-min.js
  - grid.subgrid-min.js
- themes
  - basic (a folder containing several files related to this theme)
  - coffee (another folder with theme files)
  - green (jqGrid comes with the four themes shown here)
  - sand (you can easily add your own)
  - jqModal.css

where:

- *jquery.js* is the jQuery library,
- *jquery.jqGrid.js* is the main module for including different plugins depending on your needs.

- *grid.basic.js* is the main plugin. Without this plugin, all other plugins are unusable.
- *grid.custom.js* a plugin used if we want to use advanced grid methods
- *grid.formedit.js* a plugin used for form editing, including adding and deleting data.
- *grid.inlinedit.js* a plugin used if we want to have inline editing
- *grid.subgrid.js* a plugin used if we want to use subgrids
- *grid.postext.js* a plugin (available separately) used to manipulate the post data array
- *jqModal.js* a plugin used for form editing (modal dialogs)
- *jqDnR.js* a plugin used for form editing (drag and resize)
- *themes* the directory containing the different styles for the grid.

If you want to use all of the features of jqGrid you do not need to do anything more.

If you want to use only some of the features or only the basic functions of jqGrid, you may want to edit the jquery.jqGrid.js file and remove the files you will not be using. This file must also be edited if you place the javascript files in other locations than those specified above. This file is simple and can be easily configured.

```
// we make it simple as possible
function jqGridInclude()
{
    var pathtojsfiles = "js/"; // need to be adjusted
    // if you do not want some module to be included
    // set include to false.
    // by default all modules are included.
    var minver = true;
    var modules = [
        { include: true, incfile:'grid.base.js',minfile: 'min/grid.base-min.js'}, // jqGrid base
        { include: true, incfile:'grid.custom.js',minfile: 'min/grid.custom-min.js'}, //jqGrid
custom
        { include: true, incfile:'grid.formedit.js',minfile: 'min/grid.formedit-min.js' }, //
jqGrid Form editing
        { include: true, incfile:'grid.inlinedit.js',minfile: 'min/grid.inlinedit-min.js' }, //
jqGrid inline editing
        { include: true, incfile:'grid.subgrid.js',minfile: 'min/grid.subgrid-min.js'}, //jqGrid
subgrid
        { include: true, incfile:'grid.postext.js',minfile: 'min/grid.postext-min.js'} //jqGrid
postext
    ];
    for(var i=0;i<modules.length; i++)
    {
        if(modules[i].include == true) {
           if (minver != true) IncludeJavaScript(pathtojsfiles+modules[i].incfile);
           else IncludeJavaScript(pathtojsfiles+modules[i].minfile);
        }
    }
    function IncludeJavaScript(jsFile)
    {
      var fileref=document.createElement('script');
      fileref.setAttribute("type","text/javascript");
      fileref.setAttribute("src", jsFile);
      document.getElementsByTagName("head")[0].appendChild(fileref);
    }
}

jqGridInclude();
```

If you have a different path to javascript files you must change the value of the variable *pathtojsfiles* appropriately.

If you want to exclude some modules you simply change the value of the associated *include* from true to false, in the *modules* array.

If you plan to use the form editing module you should include *jqModal.js*, *jqDnR.js* and *jqModal.css* files in your html page.

Now you are ready to use jqGrid.

How it Works

Understanding this will help you to work better with jqGrid and use the full capabilities of the plugin.

The first thing we must understand is that we have two major divisions:

- Server-side manipulation, and
- Client-side representation.

In other words, jqGrid is a component that helps you, in an easy way, to represent database information on the client side using a server-side technology. Moreover it helps you to manipulate that data back into the database.

What is server-side manipulation (SSM)? There are many definitions possible, but I try to explain it in terms of jqGrid.

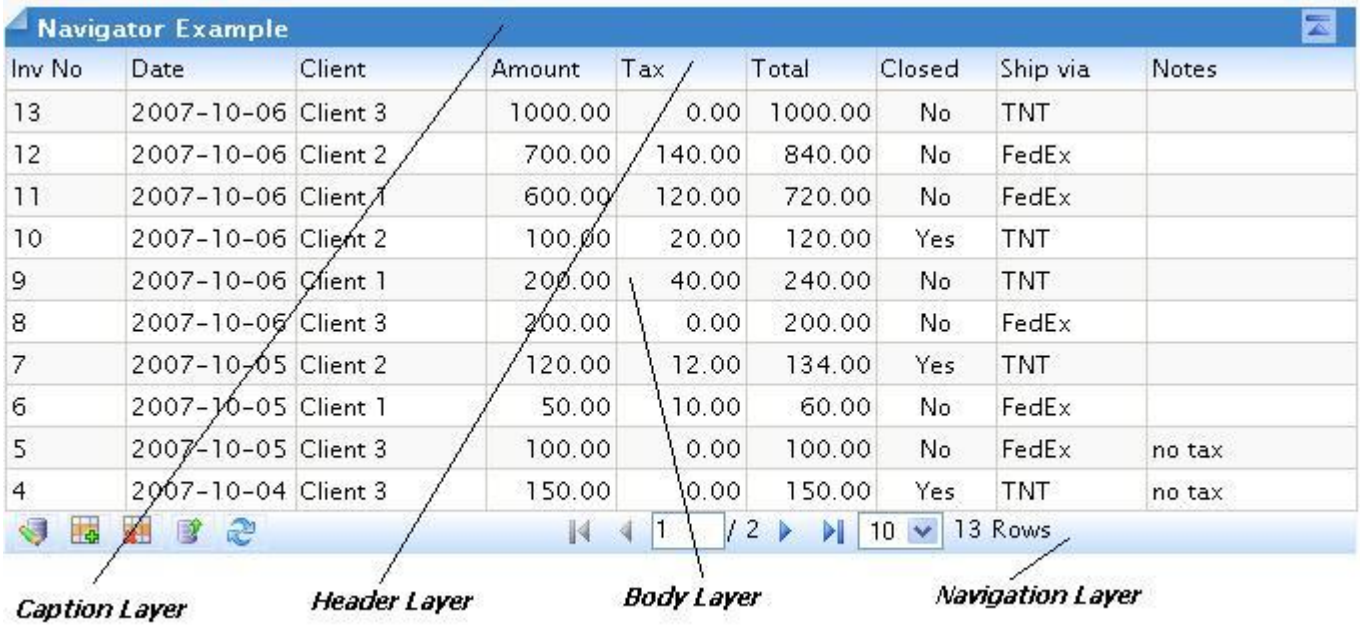
Basically SSM means the server handles the editing and not the user's browser. SSM isn't something that is visible within a web page. Everything is done on the server side using any common programming language. Basically it's a server-side command that tells the server to place a file or text within the page once it is called from a user.

In terms of jqGrid this means that you should care about this: you must have a piece of code that deals with information stored in the database using some scripting language and web server. Using this code you should be able to return requested information back to the client (web browser). jqGrid uses Ajax calls to retrieve the requested information and represent it to client using Java Script language.

Having the needed (requested) information, jqGrid constructs the representation (tabular data) described by you in what is called the Column Model (colModel).

The constructed tabular data at the client side has:

- Caption layer
- Header layer
- Body layer
- Navigation layer



Caption layer contains common information for the represented data.

Header layer contains information about the columns: labels, width, etc.

Body layer is the data requested from the server and displayed according to the settings in the column model.

Navigation layer contains additional information from the requested data and actions for requesting little pieces of information – in the literature called paging. Note that the navigation layer can be placed not only at bottom of the grid, but anywhere on the page. The Navigation layer is also the place for adding buttons or links for editing, deleting, adding to and searching your grid data.

The minimum for the representing the data are Header layer and Body layer.

To allow freedom and flexibility, and often a better impression, jqGrid relies on CCS (Cascading Style Sheets) to govern its appearance.

Tutorial: Creating Your First Grid

For this tutorial, and as an example to refer to throughout this documentation, we'll create a grid with Invoice information.

First of all we need to decide what data we want to represent at the client. Let's have the following:

- Invid – the invoice number,
- invdate – the date of the invoice,
- amount,
- tax,
- total (including tax), and
- note – additional information about the invoice.

The Data

We'll need a table with the following format. This example is based on MySQL; please create yours however you would normally do it.

```
CREATE TABLE invheader (
  invid int(11) NOT NULL auto_increment,
  invdate date NOT NULL,
  client_id int(11) NOT NULL,
  amount decimal(10,2) NOT NULL default '0.00',
  tax decimal(10,2) NOT NULL default '0.00',
  total decimal(10,2) NOT NULL default '0.00',
  note char(100) default NULL,
  PRIMARY KEY  (id)
  );
```

Then, put some values into it.

The HTML

The html file looks like this:

```
<html>
<head>
<title>jqGrid Demo</title>
<link rel="stylesheet" type="text/css" media="screen" href="themes/basic/grid.css" />
<link rel="stylesheet" type="text/css" media="screen" href="themes/jqModal.css" />
<script src="jquery.js" type="text/javascript"></script>
<script src="jquery.jqGrid.js" type="text/javascript"></script>
<script src="js/jquery.jqModal.js" type="text/javascript"></script>
<script src="js/jquery.jqDnR.js" type="text/javascript"></script>
<script type="text/javascript">
jQuery(document).ready(function(){
  jQuery("#list").jqGrid({
    url:'example.php',
    datatype: 'xml',
    mtype: 'GET',
    colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
    colModel :[
      {name:'invid', index:'invid', width:55},
      {name:'invdate', index:'invdate', width:90},
```

```
      {name:'amount', index:'amount', width:80, align:'right'},
      {name:'tax', index:'tax', width:80, align:'right'},
      {name:'total', index:'total', width:80, align:'right'},
      {name:'note', index:'note', width:150, sortable:false} ],
    pager: jQuery('#pager'),
    rowNum:10,
    rowList:[10,20,30],
    sortname: 'id',
    sortorder: "desc",
    viewrecords: true,
    imgpath: 'themes/basic/images',
    caption: 'My first grid'
  });
});
</script>
</head>
<body>
<table id="list" class="scroll"></table>
<div id="pager" class="scroll" style="text-align:center;"></div>
</body>
</html>
```

The assumption that makes the above work is that the saved file is in the directory where you placed the jqGrid files. If it is not, you will need to change the pathing appropriately.

First, we need to include the files required to construct the grid. This is done between the <head> tags in the html document.

```
<head>
<link rel="stylesheet" type="text/css" media="screen" href="themes/basic/grid.css" />
<script src="jquery.js" type="text/javascript"></script>
<script src="jquery.jqGrid.js" type="text/javascript"></script>
…
</head>
```

- The <link…/> tag loads the style sheet for jqGrid,
- The first <script ../> tag loads the jquery library,
- The second <script ../> tag loads the required jqGrid plugins,
- The third and fourth <script ../> tags load the additional modules required for some functions, and
- The last script tag is where we write the commands needed to construct the grid. A detailed description of this area appears below.

Between the <body> tags you define where you want to place the grid.

```
<body>
…
<table id="list" class="scroll"></table>
<div id="pager" class="scroll" style="text-align:center;"></div>
…
</body>
```

The definition of the grid is done via the html tag <table>. To make our life easy it is good idea to give the table an id that is unique in this html document. The second step is to assign a class "scroll" so that we can use the style definitions in the CSS provided with jqGrid.
Cellspacing, cellpadding and border attributes are added by jqGrid and shoudl not be included in the definition of your table.

We want to use a paging mechanism too, so we define the navigation layer. This can be done with the commonly-used tag. Giving the class "scroll" of the navigator specifies that we want to use the CSS provided with jqGrid. It is important to note that the navigation layer can be placed arbitrarily any place in the html document. Normally, and in this case, it is under the grid.

We use the jQuery document.ready function to run our script at the appropriate time. For more information on this, refer to the jQuery documentation.

The syntax for constructing the grid is:

```
jQuery('#grid_selector').jqGrid( options )
```

where:

- *grid_selector* is the unique id of the grid table (*list* using our example above)
- *jqGrid* is the plugin, and
- *options* is an array, in our example several lines, of the information needed to construct the grid.

Let's begin with the options array, which looks like this: (These options can appear in any order)

```
{
  url:'example.php',
  datatype: 'xml',
  mtype: 'GET',
  colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
  colModel :[
    {name:'invid', index:'invid', width:55},
    {name:'invdate', index:'invdate', width:90},
    {name:'amount', index:'amount', width:80, align:'right'},
    {name:'tax', index:'tax', width:80, align:'right'},
    {name:'total', index:'total', width:80,align:'right'},
    {name:'note', index:'note', width:150, sortable:false} ],
  pager: jQuery('#pager'),
  rowNum:10,
  rowList:[10,20,30],
  sortname: 'id',
  sortorder: 'desc',
  viewrecords: true,
  imgpath: 'themes/basic/images',
  caption: 'My first grid'
}
```

The settings and options used above are described here; listings of all settings and options can be found in API Methods and colModel API.

Property	Description
url	tells us where to get the data. Typically this is a server-side function with a connection to a database which returns the appropriate information to be filled into the Body layer in the grid
datatype	this tells jqGrid the type of information being returned so it can construct the grid. In this case we tell the grid that we expect xml data to be returned from the server, but other formats are possible. For a list of all available datatypes refer to API Methods
mtype	tells us how to make the ajax call: either 'GET' or 'POST'. In this case we will use the GET method to retrieve data from the server
colNames	an array in which we place the names of the columns. This is the text that appears in the head of the grid (Header layer). The names are separated with commas
colModel	an array that describes the model of the columns. This is the most important part of the grid. Here I explain only the options used above. For the complete list of options see colModel API: name the name of the column. This name does not have to be the name from database table, but later we will see how we can use this when we have different data formats index the name passed to the server on which to sort the data (note that we could pass column numbers instead). Typically this is the name (or names) from database -- this is server-side sorting, so what you pass depends on what your server expects to receive width the width of the column, in pixels align the alignment of the column sortable specifies if the data in the grid can be sorted on this column; if false, clicking on the header has no effect
pager	defines that we want to use a pager bar to navigate through the records. This must be a valid html element; in our example we gave the div the id of "pager", but any name is acceptable. Note that the Navigation layer (the "pager" div) can be positioned anywhere you want, determined by your html; in our example we specified that the pager will appear after the Body layer.
rowNum	sets how many records we want to view in the grid. This parameter is passed to the url for use by the server routine retrieving the data
rowList	an array to construct a select box element in the pager in which we can change the number of the visible rows. When changed during the execution, this parameter replaces the rowNum parameter that is passed to the url
sortname	sets the initial sorting column. Can be a name or number. This parameter is added to the url for use by the server routine
sortorder	sets the sorting order. This parameter is added to the url
viewrecords	defines whether we want to display the number of total records from the query in the pager bar
imgpath	the path to the images needed for the grid. The path should not end with '/'

caption	sets the caption for the grid. If this parameter is not set the Caption layer will be not visible

Having done this, we have now done half the work. The next step is to construct the server-side manipulation -- in the file pointed to in the *url* parameter in the grid.

The Server-side File

jqGrid can construct a grid from data in any of several formats, but the default is xml data with the following structure. Later we'll see how to use xml data in other structures and data in other formats.

Default xml Data Structure

```
<?xml version ="1.0" encoding="utf-8"?>
<rows>
  <page> </page>
  <total> </total>
  <records> </records>
    <row id = "unique_rowid">
      <cell> cellcontent </cell>
      <cell> <![CDATA[<font color="red">cell</font> content]]> </cell>
        …
    </row>
    <row id = "unique_rowid">
      <cell> cellcontent </cell>
      <cell> <![CDATA[<font color="red">cell</font> content]]> </cell>
        …
    </row>
      …
</rows>
```

The tags used in this example are explained in the following table.

Tag	Description
rows	the root tag for the grid
page	the number of the requested page
total	the total pages of the query
records	the total records from the query
row	a particular row in the grid
cell	the actual data. Note that CDATA can be used. This way we can add images, links and check boxes.

The number of cell tags in each row must equal the number of cells defined in the colModel. In our example, we defined six columns, so the number of cell tags in each row tag should be six.

Note the id attribute in the <row> tags. While this attribute can be omitted, it is a good practice to have a unique id for every row. If this attribute is omitted, jqGrid has two ways of dealing with need for unique ids:

1. if the property *key* in the colModel is set to true for a particular column, then jqGrid will assign the value of this column to be the id of the row; otherwise,
2. jqGrid sets the row id based on the order of the row.

If you are using a content-free primary key to identify your data rows, then do not include this value in the grid as a visible cell; instead, include it in the query and pass it as the row id attribute. It will always be available for jqGrid and even jQuery operations but not be visible on the page.

Now it is time to construct our file.

PHP and MySQL

```
<?php
//include the information needed for the connection to MySQL data base server.
// we store here username, database and password
include("dbconfig.php");

// to the url parameter are added 4 parameters as described in colModel
// we should get these parameters to construct the needed query
// Since we specify in the options of the grid that we will use a GET method
// we should use the appropriate command to obtain the parameters.
// In our case this is $_GET. If we specify that we want to use post
// we should use $_POST. Maybe the better way is to use $_REQUEST, which
// contain both the GET and POST variables. For more information refer to php documentation.
// Get the requested page. By default grid sets this to 1.
$page = $_GET['page'];

// get how many rows we want to have into the grid - rowNum parameter in the grid
$limit = $_GET['rows'];

// get index row - i.e. user click to sort. At first time sortname parameter -
// after that the index from colModel
$sidx = $_GET['sidx'];

// sorting order - at first time sortorder
$sord = $_GET['sord'];

// if we not pass at first time index use the first column for the index or what you want
if(!$sidx) $sidx =1;

// connect to the MySQL database server
$db = mysql_connect($dbhost, $dbuser, $dbpassword) or die("Connection Error: " . mysql_error());

// select the database
mysql_select_db($database) or die("Error connecting to db.");

// calculate the number of rows for the query. We need this for paging the result
$result = mysql_query("SELECT COUNT(*) AS count FROM invheader");
$row = mysql_fetch_array($result,MYSQL_ASSOC);
$count = $row['count'];

// calculate the total pages for the query
if( $count > 0 ) {
             $total_pages = ceil($count/$limit);
} else {
             $total_pages = 0;
}

// if for some reasons the requested page is greater than the total
// set the requested page to total page
if ($page > $total_pages) $page=$total_pages;

// calculate the starting position of the rows
$start = $limit*$page - $limit;
```

```
// if for some reasons start position is negative set it to 0
// typical case is that the user type 0 for the requested page
if($start <0) $start = 0;

// the actual query for the grid data
$SQL = "SELECT invid, invdate, amount, tax,total, note FROM invheader ORDER BY $sidx $sord LIMIT
$start , $limit";
$result = mysql_query( $SQL ) or die("Couldn't execute query.".mysql_error());

// we should set the appropriate header information
if ( stristr($_SERVER["HTTP_ACCEPT"],"application/xhtml+xml") ) {
              header("Content-type: application/xhtml+xml;charset=utf-8");
} else {
          header("Content-type: text/xml;charset=utf-8");
}
echo "<?xml version='1.0' encoding='utf-8'?>";
echo "<rows>";
echo "<page>".$page."</page>";
echo "<total>".$total_pages."</total>";
echo "<records>".$count."</records>";

// be sure to put text data in CDATA
while($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
echo "<row id='". $row[invid]."'>";
            echo "<cell>". $row[invid]."</cell>";
            echo "<cell>". $row[invdate]."</cell>";
            echo "<cell>". $row[amount]."</cell>";
            echo "<cell>". $row[tax]."</cell>";
            echo "<cell>". $row[total]."</cell>";
            echo "<cell><![CDATA[". $row[note]."]]></cell>";
echo "</row>";
}
echo "</rows>";
?>
```

That is all. Save the file with name example.php and your first grid is done.

COOP Example

COOP is inspirational simplicity with separation of the designer's presentation from the developer's logic. The technology starts quickly with powerful prototyping and finishes stronger with preDOM coding and clean versatile logic. The tip of the iceberg is how COOP integrates some of the greatest AJAX libraries into a single framework.

For more information refer to http://www.coldfusioncommunity.org/group/coop

This example, coop_jqGridExample.cfm, is provided by Timothy Farrar.

```
<!--- First import your COOP tagLibraries. This example assumes that your share directory with the
COOP library is at the root of your web server. --->
<cfimport prefix="coop" tagLib="/share/tags/coop">
<cfimport prefix="icejQuery" tagLib="/share/tags/coop/jquery">

<!--- Set up your coop page. --->
<coop:coop>
<html>
   <head>
       <title>JQGrid Example</title>
        </head>
   <body>
```

```
      <!--- Your grid tag will only require an id attribute here. The other settings will be
managed via the coProcessor. --->
       <icejQuery:jqGrid id="myJQGrid"/>
   </body>
</html>
</coop:coop>
```

Next, this is the COProcessor, called coop_jqGridExample.cfc

```
<cfcomponent>
   <cffunction name="onPageStart">
       <cfscript>
           var _init = structNew();
           /*
           Here we set up the attributes for the grid Object
           The gridObject attribute is an object that contains the griData,
           and either self generates or allows you to set other things required
           by the jQGrid.
           The getDataMethod attribute is the method name that will be called from the browser
           to populate the grid with data.
           *//
           createOJQGrid();
           _init.myJQGrid.jqGridObject = variables.oJQGrid;
           _init.myJQGrid.getDataMethod="getData";
                     return _init;
       </cfscript>
   </cffunction>
     <cffunction name="createOJQGrid" output="false">
       <!--- This function is where we create the jQGrid Object that is passed into the tag. --->
       <cfscript>
        if (NOT structKeyExists(variables,"ojQGrid")) {
           variables.oJqGrid = createObject("component","share.objects.coop.jquery.jqGridData");
           /* The gridObject's init method requires the following arguments:
           GridID - the ID of the grid you are setting up
           classPath - The classPath to the shareDirectory with a "." at the end.
           data - The data with which to populate the jQGrid
           *//
  variables.oJqGrid.init(gridID:"myJQGrid",classPath:"share.",data:getGalleries());
        }
       </cfscript>
   </cffunction>
     <cffunction name="getData" access="remote" output="true">
       <!---
       This is the method that is called to retrieve the data for the JQGrid.
       Note that the access level must be set to remote
         in order for it to be accessible from the browser
        --->
       <cfset var data =''>
       <cfset createOJQGrid()>
       <!---
        The data is obtained by caling the getData()
         method and passing it the grid ID and other
         arguments that the JQGrid plugin passes with a request
         --->
       <cfset data =
variables.oJQGrid.getData(gridID:'myJQGrid',page:arguments.page,sord:arguments.sord,sidx:arguments
.sidx,rows:arguments.rows)>
       <cfcontent reset="true"><cfoutput>#data#</cfoutput>
   </cffunction>
     <cffunction name="getGalleries" output="true">
       <cfquery name="photoQuery" datasource="coop">
           SELECT *
           FROM PHOTOS
       </cfquery>
```

```
        <cfreturn photoQuery>
    </cffunction>
</cfcomponent>
```

Data Types

With the first release of jqGrid, the only possible way to obtain data was via xml as described in the tutorial above. Later, many people requested the ability to obtain data via JSON, then with an array and finally with 'real' names. After lot of work and with the help of the community we now have a wide range of methods for obtaining data.

The related options (in options array) for manipulating different types of data are:

datatype: the possible options are - 'xml', 'json','clientSide' or 'local', 'xmlstring', 'jsonstring'

The default mapping for xml data is as follows:

```
xmlReader : {
  root: "rows",
  row: "row",
  page: "rows>page",
  total: "rows>total",
  records : "rows>records",
  repeatitems: true,
  cell: "cell",
  id: "[id]",
  subgrid: {
    root:"rows",
    row: "row",
    repeatitems: true,
    cell:"cell"
  }
};
```

If your server can provide data in this structure, you need to do nothing more; but if not, there is a way (several ways) to handle the data you are given. See XML Data.

The default mapping for json data is as follows:

```
jsonReader : {
  root: "rows",
  page: "page",
  total: "total",
  records: "records",
  repeatitems: true,
  cell: "cell",
  id: "id",
  subgrid: {
    root:"rows",
    repeatitems: true,
    cell:"cell"
  }
}
```

In colModel, the related options are *xmlmap* for the description of an xml field, and *jsonmap* for the description of a json field. For example:

```
colModel : [ {name:'amount',..., xmlmap:'amt'...}...]
```

will cause jqGrid to search in the xml data for an 'amt' tag (when the *repeatitems* option is set to false).

XML Data

As mentioned above, if we do not set the datatype and xmlReader parameter in the options array, the grid expects xml data, and the structure of this data is as described in our example. But what if we do not have the chance to manipulate the server response? The solution to this problem is xmlReader, again with some additions in colModel. Here is a brief description of xmlReader.

Important note: the rules of accessing the element from xml are the same as those used in jQuery library, i.e. CSS patterns. For more information refer to: http://www.w3.org/TR/REC-CSS2/selector.html

```
xmlReader : {
  root: "rows",
  row: "row",
  page: "rows>page",
  total: "rows>total",
  records : "rows>records",
  repeatitems: true,
  cell: "cell",
  id: "[id]",
  userdata: "userdata",
  subgrid: {
    root:"rows",
    row: "row",
    repeatitems: true,
    cell:"cell"
  }
}
```

The first setting here defines the *root* element. This describes where our data begins and all other loops begin from this element. For example,

```
...
<invoices>
  <request>true</request>
      ...
     <result>
       <row>
         <cell>data1</cell>
         <cell>data2</cell>
         <cell>data3</cell>
         <cell>data4</cell>
         <cell>data5</cell>
         <cell>data6</cell>
       </row>
        ...
     </result>
</invoices>
```

If we set the root element to "result" all data will be processed from there. In this case, because our rows are tagged <row> and our cells tagged <cell>, all that is needed is to set

```
xmlReader: { root:"result" }
```

The next element is the *row* element. This describes the information for particular row. Note that *row* must be a child of the *root* element. Here, if the xml looks like this,

```
<invoices>
  <request>true</request>
```

```
    ...
  <result>
    <invoice>
      <cell>data1</cell>
      <cell>data2</cell>
      <cell>data3</cell>
      <cell>data4</cell>
      <cell>data5</cell>
      <cell>data6</cell>
    </invoice>
      ...
  </result>
</invoices>
```

the setting to properly interpret this data would be

```
xmlReader: { root:"result", row:"invoice" }
```

The *page*, *total* and *record* elements describe the information needed for the pager. These elements can be, but do not have to be, a child of the *root* element. For example, to interpret this data,

```
<invoices>
  <request>true</request>
    ...
    <currentpage>1</currentpage>
    <totalpages>10</totalpages>
    <totalrecords>20</totalrecords>
    <result>
      <invoice>
        <cell>data1</cell>
        <cell>data2</cell>
        <cell>data3</cell>
        <cell>data4</cell>
        <cell>data5</cell>
        <cell>data6</cell>
      </invoice>
          ...
    </result>
</invoices>
```

the xmlReader will have to look like this:

```
xmlReader : {
  root:"result",
  row:"invoice",
  page:"invoices>currentpage",
  total:"invoices>totalpages",
  records:"invoices>totalrecords"
}
```

The *repeatitems* element tells jqGrid that the information for the data in the row is repeatable - i.e. the elements have the same tag cell described in cell element. For this example,

```
<invoices>
  <request>true</request>
    ...
    <currentpage>1</currentpage>
    <totalpages>10</totalpages>
    <totalrecords>20</totalrecords>
```

```
    <result>
      <invoice>
        <invcell>data1</invcell>
        <invcell>data2</invcell>
        <invcell>data3</invcell>
        <invcell>data4</invcell>
        <invcell>data5</invcell>
        <invcell>data6</invcell>
      </invoice>
        ...
    </result>
</invoices>
```

the xmlReader will look like this:

```
xmlReader : {
  root:"result",
  row:"invoice",
  page:"invoices>currentpage",
  total:"invoices>totalpages",
  records:"invoices>totalrecords",
  repeatitems:true,
  cell:"invcell"
}
```

Next is the *id* element. If *repeatitems* is set to true the id, if present in xml data, must be a attribute of the *row* element. Lets look at the example:

```
<invoices>
  <request>true</request>
    ...
    <currentpage>1</currentpage>
    <totalpages>10</totalpages>
    <totalrecords>20</totalrecords>
    <result>
      <invoice asin='12345'>
        <invcell>data1</invcell>
        <invcell>data2</invcell>
        <invcell>data3</invcell>
        <invcell>data4</invcell>
        <invcell>data5</invcell>
        <invcell>data6</invcell>
      </invoice>
        ...
    </result>
</invoices>
```

In this case the xmlReader is:

```
xmlReader: {
  root:"result",
  row:"invoice",
  page:"invoices>currentpage",
  total:"invoices>totalpages",
  records:"invoices>totalrecords",
  repeatitems:true,
  cell:"invcell",
  id:"[asin]"
}
```

Let's suppose that the structure of the xml document returned from the server has the following format:

```
<invoices>
  <request>true</request>
    ...
    <currentpage>1</currentpage>
    <totalpages>10</totalpages>
    <totalrecords>20</totalrecords>
    <result>
      <invoice>
        <asin>12345</asin>
        <invoiceno>data1</invoiceno>
        <invoicedate>data2</invoicedate>
        <invoiceamount>data3</invoiceamount>
        <invoicetax>data4</invoicetax>
        <invoicetotal>data5</invoicetotal>
        <notes>data6</notes>
      </invoice>
        ...
    </result>
</invoices>
```

where the <asin> tag describes the unique *id* and this should be set as the row id in the grid and not displayed in the grid. Following the rules we can construct the following:

```
xmlReader: {
  root:"result",
  row:"invoice",
  page:"invoices>currentpage",
  total:"invoices>totalpages",
  records:"invoices>totalrecords",
  repeatitems:false,
  id:"asin"
}
```

and our colModel from the example should look like this:

```
colModel :[
  {name:'invid', index:'invid', width:55, xmlmap:"invoiceno"},
  {name:'invdate', index:'invdate', width:90, xmlmap:"invoicedate"},
  {name:'amount', index:'amount', width:80, align:'right', xmlmap:"invoiceamount"},
  {name:'tax', index:'tax', width:80, align:'right', xmlmap:"invoicetax"},
  {name:'total', index:'total', width:80, align:'right', xmlmap:"invoicetotal"},
  {name:'note', index:'note', width:150, sortable:false, xmlmap:"notes"}
],
```

We can use another trick. If the names in colModel are not important for you, you can do the following.

```
colModel :[
  { name:"invoiceno", index:'invid', width:55},
  { name:"invoicedate", index:'invdate', width:90},
  { name:"invoiceamount", index:'amount', width:80, align:'right'},
  { name:"invoicetax", index:'tax', width:80, align:'right'},
  { name:"invoicetotal", index:'total', width:80, align:'right'},
  { name:"notes", index:'note', width:150, sortable:false}
],
```

In other words, jqGrid first looks to see if there is an *xmlmap* option available; if this option is not available the *name* is used as the *xmlmap*. But all of this is true only if the *repeatitems* element in xmlReader is set to false.

The *subgrid* option is included in several of the xmlReader examples above. The principles in constructing the rules for this data are the same as those described above. More details about subgrids can be found in the section on Subgrids.

XML String

The *xmlstring* option has similar features to the *xml* option. The only difference is that the data is passed as string. In this case we need to have a valid xml data string. To do that we can use a *datastr* option. This example shows how to do that.

```
<script>
var mystr =
"<?xml version='1.0' encoding='utf-8'?>
<invoices>
    <rows>
        <row>
            <cell>data1</cell>
            <cell>data2</cell>
            <cell>data3</cell>
            <cell>data4</cell>
            <cell>data5</cell>
            <cell>data6</cell>
        </row>
    </rows>
</invoices>";

jQuery(document).ready(function(){
  jQuery("#list").jqGrid({
    datatype: 'xmlstring',
    datastr : mystr,
    colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
    colModel :[
      {name:'invid', index:'invid', width:55, sorttype:'int'},
      {name:'invdate', index:'invdate', width:90, sorttype:'date', datefmt:'Y-m-d'},
      {name:'amount', index:'amount', width:80, align:'right', sorttype:'float'},
      {name:'tax', index:'tax', width:80, align:'right', sorttype:'float'},
      {name:'total', index:'total', width:80, align:'right', sorttype:'float'},
      {name:'note', index:'note', width:150, sortable:false} ],
    pager: jQuery('#pager'),
    rowNum:10,
    viewrecords: true,
    imgpath: 'themes/basic/images',
    caption: 'My first grid'
  });
});
</script>
```

As you can see, this example introduces another option in colModel: *sorttype*. This option describes how a particular column is to be sorted, because when using xmlstring as the source for the grid, jqGrid uses client-side sorting.

JSON Data

JSON data is handled in a fashion very similar to that of xml data. What is important is that the definition of the jsonReader matches the data being received

datatype: json, (or jsonstring)

The default definition of the jsonreader is as follows:

```
jsonReader : {
  root: "rows",
  page: "page",
  total: "total",
  records: "records",
  repeatitems: true,
  cell: "cell",
  id: "id",
  userdata: "userdata",
  subgrid: {root:"rows",
    repeatitems: true,
    cell:"cell"
  }
}
```

datastr:

If the parameter datatype is 'json', jqGrid expects the following default format for json data.

```
{
  total: "xxx",
  page: "yyy",
  records: "zzz",
  rows : [
    {id:"1", cell:["cell11", "cell12", "cell13"]},
    {id:"2", cell:["cell21", "cell22", "cell23"]},
      ...
  ]
}
```

The tags used in this example are described in the following table:

Tag	Description
total	total pages for the query
page	current page of the query
records	total number of records for the query
rows	an array that contains the actual data
id	the unique id of the row
cell	an array that contains the data for a row

In this case, the number of the elements in the cell array should equal the number of elements in colModel.

Let's consider our example in PHP and MySQL with JSON data. In this case I assume that the json service is enabled in PHP.

```
<?php
include("dbconfig.php");
$page = $_REQUEST['page'];  // get the requested page
$limit = $_REQUEST['rows']; // get how many rows we want to have into the grid
$sidx = $_REQUEST['sidx']; // get index row - i.e. user click to sort
$sord = $_REQUEST['sord']; // get the direction
if(!$sidx) $sidx =1;

// connect to the database
$db = mysql_connect($dbhost, $dbuser, $dbpassword) or die("Connection Error: " . mysql_error());
mysql_select_db($database) or die("Error conecting to db.");
$result = mysql_query("SELECT COUNT(*) AS count FROM invheader a, clients b WHERE
a.client_id=b.client_id".$wh);
$row = mysql_fetch_array($result,MYSQL_ASSOC);
$count = $row['count'];
if( $count >0 ) {
    $total_pages = ceil($count/$limit);
} else {
    $total_pages = 0;
}
if ($page > $total_pages) $page=$total_pages;
$start = $limit*$page - $limit; // do not put $limit*($page - 1)
if ($start<0) $start = 0;
$SQL = "SELECT invid,invdate,amount,tax,total,note FROM invheader ORDER BY ".$sidx." ".$sord. "
LIMIT ".$start." , ".$limit;
$result = mysql_query( $SQL ) or die("Could not execute query.".mysql_error());

// Construct the json data
$responce->page = $page; // current page
$responce->total = $total_pages; // total pages
$responce->records = $count; // total records
$i=0;
while($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
    $responce->rows[$i]['id']=$row[invid]; //id
    $responce-
>rows[$i]['cell']=array($row[invid],$row[invdate],$row[amount],$row[tax],$row[total],$row[note]);
    $i++;
}
echo json_encode($responce);
?>
```

The structure of the jsonReader is very similar to the xmlReader. The only missing part is the *row* element which is not needed for JSON data. Let's begin our walk through the jsonReader.

The first element is a *root* element. This element describes where our data begins. In other words, this points to the array that contains the data. If we set

```
jsonReader: { root:"invdata" }
```

then the returned string should be

```
{
  total: "xxx",
  page: "yyy",
  records: "zzz",
  invdata: [
    {id:"1", cell:["cell11", "cell12", "cell13"]},
    {id:"2", cell:["cell21", "cell22", "cell23"]}
  ]
}
```

The *page*, *total* and *record* elements describe the information needed for the pager. For example, if the jsonReader is set as follows,

```
jsonReader:{
  root: "invdata",
  page: "currpage"
  total: "totalpages"
  records: "totalrecords"
}
```

then the data should be

```
{
  totalpages: "xxx",
  currpage: "yyy",
  totalrecords: "zzz",
  invdata: [
    {id:"1", cell:["cell11", "cell12", "cell13"]},
    {id:"2", cell:["cell21", "cell22", "cell23"]}
  ]
}
```

The *cell* element describes the array which contains the data for the row.

```
jsonReader:{
  root: "invdata",
  page: "currpage"
  total: "totalpages"
  records: "totalrecords",
  cell: "invrow"
}
```

The data to match this description would be

```
{
  totalpages: "xxx",
  currpage: "yyy",
  totalrecords: "zzz",
```

```
  invdata : [
    {id:"1", invrow:["cell11", "cell12", "cell13"]},
    {id:"2", invrow:["cell21", "cell22", "cell23"]}
  ]
}
```

The *id* element descibes the unique id for the row

```
jsonReader:{
  root: "invdata",
  page: "currpage"
  total: "totalpages"
  records: "totalrecords",
  cell: "invrow",
  id: "invid"
}
```

The data for this description is:

```
{
  totalpages: "xxx",
  currpage: "yyy",
  totalrecords: "zzz",
  invdata : [
    {invid:"1", invrow:["cell11", "cell12", "cell13"]},
    {invid:"2", invrow:["cell21", "cell22", "cell23"]}
  ]
}
```

It is possible to set the *cell* element to an empty string. And, it is possible to set the *id* as number. Here is an example of these possibilities:

```
jsonReader:{
  root: "invdata",
  page: "currpage"
  total: "totalpages"
  records: "totalrecords",
  cell: "",
  id: "0"
}
```

In this case the *id* is the first element from the row data

```
{
  totalpages: "xxx",
  currpage: "yyy",
  totalrecords: "zzz",
  invdata: [
    {"1","cell11", "cell12", "cell13"},
    {"2", "cell21", "cell22", "cell23"}
  ]
}
```

The *repeatitems* element tells jqGrid that the information for the data in the row is repeatable - i.e. the elements have the same tag cell described in cell element. Setting this option to *false* instructs jqGrid to search elements in the json data by name. This is the name from colModel or the name described with the *jsonmap* option in colModel.

Here is an example:

```
jsonReader:{
  root: "invdata",
  page: "currpage"
  total: "totalpages"
  records: "totalrecords",
  repeatitems: false,
  id: "0"
}
```

The resulting data in our example should be:

```
{
  totalpages: "xxx",
  currpage: "yyy",
  totalrecords: "zzz",
  invdata: [
    {invid:"1",invdate:"cell11", amount:"cell12", tax:"cell13", total:"1234", note:"somenote" },
    {invid:"2",invdate:"cell21", amount:"cell22", tax:"cell23", total:"2345", note:"some note" }
  ]
}
```

The *id* element in this case is 'invid'.

A very useful feature here is that there is no need to include all the data that is represented in colModel. Since we make a search by name, the order does not need to match the order in colModel. Hence the following string will be correctly interpreted in jqGrid.

```
{
  totalpages: "xxx",
  currpage: "yyy",
  totalrecords: "zzz",
  invdata: [
    {invid:"1",invdate:"cell11",  note:"somenote" },
    {invid:"2", amount:"cell22", tax:"cell23", total:"2345" }
  ]
}
```

JSON String

The jsonstring option has the same features as json. The only difference is that the data is passed as string. In this case we need to have a valid json data string. To do that we can use a *datastr* option. See the xmlstring example.

Array Data

Despite the fact that the primary goal of jqGrid is to represent dynamic data returned from a database, jqGrid includes a wide range of methods to manipulate data at client side: Array data.

Related options in options array: datataype
Related options in colModel: sorttype, datefmt
Related methods : getRowData, delRowData, setRowData, addRowData

If we have defined a pager for grid with client side data, the buttons in pager are automatically disabled. With other words the current release of grid does not support client side paging.

First we must instruct jqGrid that the data that will be present is at client side. This is done with the option datatype. To use this we must set

```
datatype : "clientSide"
```

The other option that can be used is "local" i.e. datatype: "local"
That are the same things.

Having this it is a good idea to set the sorttypes for the columns. If the sorttype
is not set the default sorttype is "text". Let's consider our example in terms of array data.

```
<script>
jQuery(document).ready(function(){
    jQuery("#list").jqGrid({
        datatype: 'clientSide',
        colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
        colModel :[
        {name:'invid',index:'invid', width:55, sorttype:'int'},
        {name:'invdate',index:'invdate', width:90, sorttype:'date', datefmt:'Y-m-d'},
        {name:'amount',index:'amount', width:80, align:'right',sorttype:'float'},
        {name:'tax',index:'tax', width:80, align:'right',sorttype:'float'},
        {name:'total',index:'total', width:80,align:'right',sorttype:'float'},
        {name:'note',index:'note', width:150, sortable:false} ],
        pager: jQuery('#pager'),
        rowNum:10,
        viewrecords: true,
        imgpath: 'themes/basic/images',
        caption: 'My first grid'
});
});
</script>
```

You can see the new setting here: datatype, sortype and datefmt.

The possible values for the sorttype are:
int - the data is interpreted as integer,
float - the data is interpreted as decimal number
date - the data is interpreted as data
text - the data is interpreted as text

We need this information for the appropriate sorting of these types. Additionally for the sorttype date we must known the format of the data that will be present in the grid. The default format is a ISO format 'Y-m-d'. The description of the date format is like a PHP way. For more information refer to php.net.
The limitation of date format is that the date can be represented only as numbers and not as number and string. By example if the date is represented as '03-Mar-2008' the sorting will be not correct.

Let's add some data. This can be done with the method addRowData.
The parameters to this method are:

```
addRowData( rowid, data, position )
```

where:
rowid - this value will be set as the id of the row
data - is the array of data in pair name:value, where the name is the name from colModel.
position - tells where to add the data - "last" add the data at last position, "first" add the data as first row.

```
<script>
...
myfirstrow = {
invid:"1",
invdate:"2007-10-01",
note:"note",
amount:"200.00",
tax:"10.00",
total:"210.00"}
jQuery("list").addRowData("1", myfirstrow);
...
</script>
```

With this line we have added our first row. It is important to note that the order of the name can be in arbitrary way. More over we can set here only one pair name value Like this.

```
jQuery("list").addRowData("2", {amount:"300.00"});
```

with this line we have added second row with only one value in the amount column.

To get data from the particular row we should use getRowData method:

```
getRowData( rowid ),
```

where rowid is the id for the row which values we will obtain

```
jQuery("list").getRowData("1");
```

will return array as pair name value - the result is

```
{invid:"1",
invdate:"2007-10-01",
note:"note",
amount:"200.00",
tax:"10.00",
total:"210.00"}
```

To delete a row we should use delRowData method:

```
delRowData( rowid )
```

where rowid is the id of the row that can be deleted.

```
jQuery("list").delRowData("2");
```

will delete the row with the id = 2.
The method return true if the deleting is successfull, false othervise

To change all or part of values to a given row we can use a setRowData method.

```
setRowData( rowid, data )
```

where
rowid is the id of the row which values will be changed
data is a array of data that contain the new values. The structure of array is in type name:value.

```
jQuery("list").setRowData( "1", { tax:"5", total:"205" })
```

will change the values tax and total of row with id = 1.

The method return true if update is successful, otherwise false.

User Data

In some cases we need to have custom data returned from the request that is not automatically displayed by jqGrid, that we use either in a later process or to display additional information somewhere in the html page or associated with the grid. To do that a *userdata* tag can be used.

```
xmlReader: {userdata: "userdata",... }
```

In the data received from the server, this could be structured as follows (in xml):

```
<invoices>
  <request>true</request>
    <userdata name="totalinvoice"> 240.00 </userdata>
    <userdata name="tax">  40.00</userdata>
      ...
    <result>
      <row>
        <cell>data1</cell>
        <cell>data2</cell>
        <cell>data3</cell>
        <cell>data4</cell>
        <cell>data5</cell>
        <cell>data6</cell>
      </row>
         ...
    </result>
</invoices>
```

If using json data, the definition might look like this:

```
jsonReader : {
...
userdata: "userdata",
...
}
```

and the data received, like this:

```
{
  total: "xxx",
  page: "yyy",
  records: "zzz",
  userdata: {totalinvoice:240.00, tax:40.00},
  rows : [
    {id:"1", cell:["cell11", "cell12", "cell13"]},
    {id:"2", cell:["cell21", "cell22", "cell23"]},
    ...
  ]
}
```

When this data has been received, this information is stored in the *userData* array of the options array. Whichever format the data comes in, in this case we would have:

```
userData = {totalinvoice:240.00, tax:40.00}
```

The data can be accessed two ways.

1. Using getReturnedData (provided by Paul Tiseo): this method directly returns the userData array

   ```
   jQuery("grid_id").getReturnedData()
   ```

2. Using a getGridParam method. To do that we need to request this data:

   ```
   jQuery("grid_id").getGridParam('userData')
   ```

Both methods return the same array.

Basic Grid

An instance of jqGrid is a javascript object, with properties, events and methods. Properties may be strings, numbers, arrays, boolean values or even other objects.

When the grid is created, normally several properties are set in the same statement (although these properties can be individually overridden later): see Properties

Events raised by the grid, which offer opportunities to perform additional actions, are described in Events.

The grid also offers several methods for getting or setting options or data: see Methods

A key property of the grid is the column model (colModel) that defines the contents of the grid: colModel Properties

Properties

Calling Convention:

```
jQuery("#grid_id").jqGrid(properties );
```

Where:

- *grid_id* is the id of the <table> element defined separately in your html and used as the name of your grid
- *properties* is an array of settings in *name: value* pairs format. Some of these settings are values, some are functions to be performed on an event. Some of these settings are optional while others must be present for jqGrid to work.

An example, taken from the tutorial:

```
jQuery("#list").jqGrid({
    url:'example.php',
    datatype: 'xml',
    mtype: 'GET',
    colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
    colModel :[
      {name:'invid', index:'invid', width:55},
      {name:'invdate', index:'invdate', width:90},
      {name:'amount', index:'amount', width:80, align:'right'},
      {name:'tax', index:'tax', width:80, align:'right'},
      {name:'total', index:'total', width:80, align:'right'},
      {name:'note', index:'note', width:150, sortable:false} ],
    pager: jQuery('#pager'),
    rowNum:10,
    rowList:[10,20,30],
    sortname: 'id',
    sortorder: "desc",
    viewrecords: true,
    imgpath: 'themes/basic/images',
    caption: 'My first grid'
  });
```

The available properties are listed here, in alphabetic order. Some have more details described elsewhere, available by clicking on the link (the name of the property).

Property	Type	Description	Default
altRows	boolean	Set a zebra-striped grid	true
caption	string	Defines the Caption layer for the grid. This caption appear above the Header layer. If the string is empty the caption does not appear.	empty string
colModel	array	Array which describes the parameters of the columns. For a full description of all valid values see colModel API.	empty array
colNames	array	Array which describes the column labels in the grid	empty array
datastr	string	The string of data when datatype parameter is set to xmlstring or jsonstring	null
datatype	string	Defines what type of information to expect to represent data in the grid. Valid options are xml - we expect xml data; xmlstring - we expect xml data as string; json - we expect JSON data; jsonstring - we expect JSON data as string; clientSide - we expect data defined at client side (array data)	xml
editurl	string	Defines the url for inline and form editing.	null
firstimg , previmg , nextimg, lastimg	string	Links to image url which are used in the pager bar for the first, previous, next and last buttons	first.gif, prev.gif, next.gif, last.gif
hiddengrid	boolean	If set to true the grid initially is hidden. The data is not loaded (no request is sent) and only the caption layer is shown. When the show/hide button is clicked the first time to show grid, the request is sent to the server, the data is loaded, and grid is shown. From this point we have a regular grid. This option has effect only if the *caption* property is not empty and the *hidegrid* property (see below) is set to true.	false
hidegrid	boolean	Enables or disables the show/hide grid button, which appears on the right side of the Caption layer. Takes effect only if the *caption* property is not an empty string.	true
height	string	The height of the grid. Can be set as percentage or any valid measured value	150px
imgpath	string	Defines the path to the images that are used in the grid. Set this option without / at end	empty string
jsonReader	array	Array which describes the structure of the expected json data. For a full description and default setting, see JSON Data	
loadonce	boolean	If this flag is set to true, the grid load only once the data from the server according to	false

		the parameter datatype. After the first request (xml or json) the datatype parameter is set automatically to clientSide. All other manipulations are done at client side. The manipulations at pager (if any) are disabled.	
loadtext	string	The text which appear when requesting and sorting data	Loading...
loadui	string	This option controls what to do when an ajax operation is in progress. • disable - disables the red box indicator on upper left corner. This way you can use your own indicator. • enable (default) - enables the red box indicator • block - enables the red box indicator and blocks all actions in the grid until the ajax request is finished. Note that this disables paging, sorting and all actions on toolbar, if any.	
mtype	string	Defines the type of request to make ("POST" or "GET")	GET
multikey	string	This parameter have sense only multiselect option is set to true. Defines the key which will be pressed when we make multiselection. The possible values are: shiftKey - the user should press Shift Key altKey - the user should press Alt Key ctrlKey - the user should press Ctrl Key	empty string
multiselect	boolean	If this flag is set to true a multi selection of rows is enabled. A new column at left side is added. Can be used with any datatype option.	false
page	integer	The requested initial page number when we use datatypes xml or json (data returned from server)	1
pager	DOM element or string	Sets the pager bar for the grid. Must be a valid html element. If the element has class “scroll”, then the width is equal to the grid. Usage: If parameter is a DOM element, jQuery("#mypager"); if using a string, "mypager", where mypager is the id of the pager. Note the missing "#"	
postData	array	This array is passed directly to the url. This is associative array and can be used this way: {name1:value1...}. See API methods for manipulation.	empty array
recordtext	string	Displays the text associated with the display of total records	Rows

rowList	array	This parameter construct a select box element in the pager in which the user can change the number of the visible rows.	empty array
rowNum	integer	The initial number of rows that must be returned from the server	20
shrinkToFit	boolean	This option describes the type of calculation of the initial width of each column against with the width of the grid. If the value is true and the value in width option is set then: Every column width is scaled according to the defined option width. Example: if we define two columns with a width of 80 and 120 pixels, but want the grid to have a 300 pixels - then the columns are recalculated as follow: 1- column = 300(new width)/200(sum of all width)*80(column width) = 120 and 2 column = 300/200*120 = 180. The grid width is 300px. If the value is false and the value in width option is set then: The width of the grid is the width set in option. The column width are not recalculated and have the values defined in colModel.	true
sortascimg, sortdescimg	string	Links to image url which are used when the user sort a column	sort_asc.gif, sort_desc.gif
sortname	string	The initial sorting name when we use datatypes xml or json (data returned from server)	none (empty string)
sortorder	string	The initial sorting order when we use datatypes xml or json (data returned from server)	asc
subGrid	boolean	Is set to true this enables using of sub grid. Add additional column at the left side of the grid. For more information see Setting subgrid	false
subGridModel	array	This option has effect only if subGrid option is set to true. This option describes the model of the subgrid. For more details see Subgrids	empty array
subGridUrl	string	This option has effect only if subGrid option is set to true. This describes the url for the subgrid data. Additinally to this url is added parameter “id” which is the id of the row.	empty string
toolbar	array	This option defines the toolbar of the grid. This is array with two values in which the first value enables the toolbar and the second defines the position relative to body Layer. Possible values "top" or "bottom"	[false,"top"]
url	string	The url of the file that holds the request	null
userData	array	This array contain custom information from the request. Can be used at any time. See API methods for manipulation	empty array
viewrecords	boolean	Display the total records from the query in	false

		the pager bar	
width	number	If this option is not set, the width of the grid is a sum of the width's of the columns defined in the colModel (in pixels). If this option is set, the initial width of each column is set according to the value of *shrinkToFit* option.	none

xmlReader	array	Array which describes the structure of the expected xml data. For a full description refer to Data Types.	{
			root: "rows",
			row: "row",
			page: "rows>page",
			total: "rows>total",
			records : "rows>records",
			repeatitems: true,
			cell: "cell",
			id: "[id]",
			subgrid: {
			root:"rows",
			row: "row",
			repeatitems: true,
			cell:"cell"
			}
			}

$.jgrid.default	array	This array is used to define a grid option common to all jqGrids in the application. Typically, this is called once to set the default for one or more grid parameters -- any parameter can be changed. Typical implementation; $.extend($.jgrid.defaults,{rowNum:10})	empty array

colModel Properties

The colModel property defines the individual grid columns as an array of properties.
Syntax:

```
colModel: [
    {name:'name1', index:'index1'...},
    {...},
     ...
  ]
```

The available *colModel* properties are listed here, in alphabetic order. All of these properties are values, there are no events or methods associated with the *colModel*. The only required property is *name*.

Property	Type	Description	Default
align	string	Defines the alignment of the cell in the Body layer, not in header cell. Possible values: left, center, right.	left
datefmt	string	Only in combination with sorttype:date. Determines the expected date format for that column. Uses a PHP-like date formatting.	ISO Date (Y-m-d)
editable	boolean	Defines if the field is editable. This option is used in inline and form modules.	false
editoptions	array	Array of allowed options (attributes) for edittype option	empty array
editrules	array	editrules: {edithidden:true(false), required:true(false), number:true(false), minValue:val, maxValue:val} With this option we can: 1. include fields hidden in the grid as editable in the form. If the field is hidden in the grid and *edithidden* this is set to true, the field can be edited when add or edit methods are called. 2. perform a client-side validation in the formedit. This is done with: ○ required:true(false), number:true(false), minValue:val, maxValue:val parameters ○ requiered if set to true the value will be checked if it is empty ○ number - if set to true the value will be checked if this is a number ○ minValue - valid number - if set the value will be checked and if the value is less than this a error message will be displayed. ○ maxValue the same as minValue but the value will be checked if it is a greater of a given maxValue.	empty array
edittype	string	Defines the edit type for inline and form editing Possible values: text, textarea, select, checkbox See Inline editing and form editing	text
hidden	boolean	Defines if this column is hidden at initialization.	false
index	string	Set the index name when sorting. Passed as sidx parameter.	the order of cell
jsonmap	string	Defines the json mapping for the column in the incoming json string.	none
key	boolean	In case if there is no id from server, this can be set as as id for the unique row id. Only one column can have this property. If there are more than one key the grid finds the first one and the second is ignored.	false
name	string	Set the unique name in the grid for the column. This property is required. As well as other words used as property/event names, the reserved words (which	

		cannot be used for names) include *subgrid* and *cb*.	
resizable	boolean	Defines if the column can be resized	true
search	boolean	When used in formedit, disables or enables searching of that column	true
sortable	boolean	Defines is this can be sorted.	true
sorttype	string	Used when datatype is clientSide. Defines the type of the column for appropriate sorting. Possible values: • int - for sorting integer • float - for sorting decimal numbers • date - for sorting date • text - for text sorting	text
width	number	Set the initial width of the column, in pixels	150
xmlmap	string	Defines the xml mapping for the column in the incomming xml file. Use a CCS specification for this	none

Height and Width

Height

The height of the grid can be controlled via the *height* property.

The height property can be set in pixels, a percent or any valid height measure. The default setting is pixels (px); the default value for height is 150, i.e. 150px. To set the grid height in 200 pixels we need to set only the number - i.e.

```
height: 200
```

It is important to note that this setting controls only the height of the Body layer. The height of whole grid is

- the height of the Caption layer (if present),
- plus the height of the Header layer
- plus the height of the Body layer (as set in this option)
- plus the height of the pager (if set as a part of grid).

Setting the height to 100% means that the Body layer will be set to contain all of the returned rows without scrolling. Any other setting will fix the height of the Body layer and show the scroll bar as needed.

Width

The width of the grid is set only in pixels.

By default, the *width* property of the grid is not set, and the width of the grid is calculated as sum of the width properties of the individual columns set in the *colModel*. (If the *width* property in colModel is not set for a column, the width of that column defaults to 150px).

However, setting the width based on the colModel can be somewhat misleading. The true width of a grid defined this way will be

- the sum of the widths defined in the colModel
- plus padding and border settings for the cells (set in the CSS).

Suppose we have 5 columns with width settings that sum to 500 and in the CSS we have the following definition for the table data:

```
table.scroll tbody td  {
    padding: 2px;
    text-align: left;
    border-bottom: 1px solid #D4D0C8;
    border-left: 1px solid #D4D0C8;
    text-overflow: ellipsis;
    overflow: hidden;
    white-space: nowrap;
}
```

The width of this grid will be 500 (the settings in colModel) plus 5*3 (2 padding pixels + 1 border left pixel) for a total of 515px.

If this somewhat larger width is not a problem for you, then you need do nothing more. However, in some cases it is needed to have the width of the grid fixed and independent of the widths set for the columns in colModel. To accomplish this, we can play with two other options in the grid's options array: *width* and *shrinkToFit*.

The default value of *shrinkToFit* is true which means that if we set the width parameter of the grid, the widths of all the columns are recalculated as:

new colModel.width = colModel.width * width / swidth

where:

- *colModel.width* is the width set for this column in the colModel
- *width* is width set for the grid
- *swidth* is the sum of all widths set in the colModel

The effect of this recalculation is that all columns are shrunk proportionately. (Or, if the fixed width of the grid is larger than the sum of the columns, all columns will be proportionately expanded).

If the value of *shrinkToFit* is false then jqGrid does not make any recalculation of the initial column width and the grid will have the *width* set in options array, with a horizontal scroll bar.

Obsolete Properties

The following property, part of jqGrid up to and including version 3.1, has been removed starting with version 3.2.

Property	Description	Use Instead
rowheight	This option was used to define the height of a single row so that the overall height of the grid could be set according to the number of returned rows, making scrolling unnecessary	height: '100%'

Events

The action to take on an event is set as a property of the grid, e.g.

```
onSelectRow: function(id){
  if(id && id!==lastSel){
    jQuery('#tbleditable').restoreRow(lastSel);
    lastSel=id;
  }
  jQuery('#tbleditable').editRow(id, true); },
```

This example specifies the action to take when a row is selected.

The events that you can use to perform some additional action are listed here, in alphabetic order:

Event	Parameters	Description
afterInsertRow	rowid, rowdata	This event fires after every inserted row; *rowid* is the id of the inserted row, *rowdata* is an array of the data to be inserted into the row. This is array of type *name: value*, where the name is a *name* from colModel
gridComplete	none	This fires after all the data is loaded into the grid and all other processes are complete
loadBeforeSend	xhr	A pre-callback to modify the XMLHttpRequest object (xhr) before it is sent. Use this to set custom headers etc. The XMLHttpRequest is passed as the only argument.
loadComplete	none	This event is executed immediately after every server request
loadError	xhr,st,err	A function to be called if the request fails. The function gets passed three arguments: The XMLHttpRequest object (XHR), a string describing the type of error (st) that occurred and an optional exception object (err), if one occurred.
onSelectRow	rowid	Raised immediately after row was clicked. Calling convention: onSelectRow: function(rowid) {

		// here is the code
		}

ondblClickRow	rowid	Raised immediately after row was double clicked. Calling convention:
		ondblClickRow: function(rowid) {
		// here is the code
		}

onPaging	pgButton	This event fires after click on [page button] and before populating the data. Also works when the user enters a new page number in the page input box (and presses [Enter]). pgbutton(string) can be - first,last,prev,next

onRightClickRow	rowid	Raised immediately after row was right clicked. Calling convention:
		onRightClickRow: function(rowid) {
		// here is the code
		}

onSelectAll	array of the selected ids	This event fires (if defined) when multiselect is true and you click on the header checkbox. Parameter passed to this event is array of selected rows. If the rows are unselected, the array is empty.

onSelectRow	rowid	Raised immediately after row was clicked. Calling convention:
		onSelectRow: function(rowid) {
		// here is the code
		}

onSortCol	index - the index name from colModel colindex - the index of column	Raised immediately after sortable column was clicked and before sorting the data
		Calling convention:
		onSortCol: function(index, colindex) {
		// here is the code
		}

subGridRowExpanded	pID - the unique id of the div element where we can put contents when subgrid is enabled, id - the id of the row	This event is raised when the subgrid is enabled and is executed when the user clicks on the plus icon of the grid. Can be used to put custom data in the subgrid.
subGridRowColapsed	pID - the unique id of the div element where we can put contents when subgrid is enabled, id - the id of the row	This event is raised when the user clicks on the minus icon.

Methods

Calling Convention:

```
jQuery("#grid_id").jqGridMethod( parameter1,...parameterN )
```

Where:

- *grid_id* is the id of the already constructed jqGrid.
- *jqGridMethod* is a method applied to this jqGrid.
- *parameter1,...parameterN* - a list of parameters

IMPORTANT: Some 'get' and 'set' methods (e.g., getUrl()) supported up to version 3.1 have been removed in version 3.2 and replaced by more generic methods (e.g., getGridParam(url) or setGridParam({url:value})). See Replaced Methods for details.

Where a method is not designed to return a requested value, then what is returned is the jqGrid object and a set of such methods can be chained, e.g.,

```
jQuery("#grid_id").setGridParam({..}).hideCol("somecol").trigger("reloadGrid")
```

Available methods (as of version 3.2)

addRowData
Parameters: rowid - string, data - array, position - string (first, last) - default last
Returns: true on success, otherwise false
Inserts a new row with *id* = rowid at the *position* using the data in *data*.The syntax of the *data* array is: {name1:value1,name2: value2…} where *name* is the name of the column as described in the colModel and the *value* is the value.

clearGridData
Parameters: none
Returns: jqGrid object
Clears the currently loaded data from grid

delRowData
Parameters: rowid - string,
Returns: true on success, otherwise false
Deletes the row with the id = rowid. This operation does not delete a data from the server.

FormToGrid
Parameters: rowid - string, formid - string
Returns: jqGrid object
Reads data from a form (previously defined in html) identified by formid and loads data into the grid in row with rowid. If the names of both grid and form are the same the data from the form replaces the data in the grid. Note that all fields from grid can be replaced, including hidden. This is the opposite of *GridToForm*

getDataIDs
Parameters: none
Returns: array of the id's in the current grid view. Empty array if no data is available.

getGridParam
Parameters: name - string
Returns: the value of the requested parameter.
The name is the name from options array. For a particular options, see below. If the name is not set the entry options are returned.

Option	Returns
getGridParam("url")	the current url from options array
getGridParam("sortname")	the name of last sorted column
getGridParam("sortorder")	the last sorted order
getGridParam("selrow")	the id of the selected row, null if row is not selected
getGridParam("page")	the current page number.
getGridParam("rowNum")	the current number of requested rows
getGridParam("datatype")	the current datatype.
getGridParam("records")	the current number of records in grid.
getGridParam("selarrrow")	array of id's of the selected rows when multiselect options is true. Empty array if not selection.

getRowData
Parameters: rowid - string
Return: array with data of the requested id = rowid. The returned array is of type name:value, where the name is a name from colModel and the value is a actual value. Returns empty array if the row can not be found.

GridToForm
Parameters: rowid - string, formid - string
Returns: jqGrid object
Reads data from a given rowid and fills the form (previously defined in html) identified by formid. If the names of both grid and form are the same the data from the grid replaces the data in the form. Note that all fields from grid can be used, including hidden.

hideCol
Parameters: colname - mixed (string or array)
Returns: jqGrid object
Hides a column with a given colname. If the colname is a string, only the specified column is hidden. If the colname is array of type ["name1","name2"] then the columns with names 'name1' and 'name2' will be hidden at the same time. The colname must be a valid *name* from colModel. The width of the grid is changed according to the following rules: if the grid currently has no horizontal scroll bar, the width of the grid is decreased by the width of the hidden column(s). If a scrollbar is visible, the width is adjusted which may or may not change the width of the grid.

resetSelection
Parameters: none
Returns: jqGrid object
Resets (unselects) the selected row(s). Also works in multiselect mode.

setCaption
Parameters: caption - string
Returns: jqGrid object
Sets a new caption of the grid. If the Caption layer was hidden it is shown.

setCell
Parameters: rowid, colname, nData, prop
Returns: jqGrid object
This method can change the content of particular cell and can set class or style properties.
Where:

- *rowid*: the id of the row,
- *colname*: the name of the column (this parameter can be a number beginning from 0)
- *nData*: the content that can be put into the cell. If empty string the content will not be changed
- *prop*: if prop is string then we add a class to the cell using addClass; if prop is array we set the new css properties via css

Example
```
setCell("10","tax",'',{color:'red','text-align':'center'})
```
will set the contents of the *tax* field in row *10* to red and centered.

setGridParam
Parameters: object - array
Returns: jqGrid object
Sets a particular parameter. Note - for some parameters to take effect a trigger("reloadGrid") should be executed. Note that with this method we can override events like onSelectRow, etc. Example:
```
setGridParam({
  url:"newurl",
  page:1,
  onSelectRow:function(id){/*here is the new code*//}
});
```

Method	Description
setGridParam({url:newvalue})	Parameters: url - string Set a new url, replacing the older.
setGridParam({sortname:newvalue})	Parameters: sortname - string Set a new sort name
setGridParam({sortorder:newvalue})	Parameters: sortorder - string (asc or desc) Set a new sort order
setGridParam({page:newvalue})	Parameters: page - integer >0 Set a new page number
setGridParam({rowNum:newvalue})	Parameters: rownum - integer > 0 Set a new number of requested rows.
setGridParam({datatype:newvalue})	Parameters: datatype - string (xml,json.xmlstring,jsonstring, clientSide) Set a new datatype.

setLabel
Parameters: colname, newlabel, sattr
Returns: jqGrid object
Sets a new label in the header for the specified column; can also set attributes and classes (sattr). The parameters are:

- colname(mixed) is either the name of the column (from colModel) or the number of the column in colModel beginning from 0.
- newlabel(string) is the label that we want to change. Can be a empty string.
- sattr(mixed) - if this parameter is array - we add this as attributes to this header element. if the parameter is string we add a class to this element

setRowData
Parameters: rowid - string, data - array
Return: boolean
Updates the values (using data array) in the row with the given rowid. The syntax of data array is:

{name1:value1,name2: value2…} where the name is the name of the column as described in the colModel and the value is the new value. Return true on success, false otherwise.

setSelection
Parameters: rowid - string
Returns: jqGrid object
Toggles a selection of the row with id = rowid

showCol
Parameter: colname - mixed (string or array)
Returns: jqGrid object
Shows a column with a given colname. If the colname is a string we show only one column. If colname is array of type ["name1","name2"] then the columns with names 'name1' and 'name2' will be shown at the same time The colname must be a valid name from colModel. The width of the grid changes by the width of the newly-shown columns.

.trigger("reloadGrid");
Parameter: none
Returns: jqGrid object
Reloads the grid with the current settings. This means that a new request is send to the server if datatype is xml or json.
This method should be applied to an already-constructed grid - e.g.,
```
jQuery("#grid_id").trigger("reloadGrid");
```

Advanced Methods

Advanced methods offer the ability to dynamically change properties of the *colModel*. To keep the basic code small, these reside in a separate module (grid.custom.js) that must be installed for these to be available. See Installation

getColProp
Parameters: colname
Returns: an array of the properties of the given column from *colModel*

setColProp
Parameters: colname, properties
Returns: jGrid object
Sets new properties in *colModel*. This method is ideal for dynamically changing properties of the column. Note that some properties - like *width* and *align* - have no effect. Example:
```
jQuery("#grid_id").setColProp('colname',{editoptions:{value:"True:False"}})
```
will change the editoptions values.

sortGrid
Parameters: colname,reload
Returns: jGrid object
Sorts the given colname and shows the appropriate sort image. The same (without sorting image) can be done using
```
setGridParam({sortname:'myname'}).trigger('reloadGrid')
```
If the reload is set to true, the grid reloads with the current page and sortorder settings.

GridDestroy
Parameters: grid_id
Returns: true on success, otherwise false
Destroys the entry grid from the DOM (clears all the html associated with the grid and unbinds all events)

GridUnload
Parameters: grid_id
Returns: true on success, otherwise false
The only difference to previous method is that the grid is destroyed, but the table element and pager (if any) are left ready to be used again.

Obsolete Methods

The following methods, all part of jqGrid up to and including version 3.1, have been removed starting with version 3.2, replaced by a more generic method.

'Get' Methods	Returns	Use Instead
getUrl	the current url from options array	getGridParam("url")
getSortName	the name of last sorted column	getGridParam("sortname")
getSortOrder	the last sorted order	
getSelectedRow	the id of the selected row, null if row is not selected	getGridParam("selrow")
getPage	the current page number	getGridParam("page")
getRowNum	the current number of requested rows	getGridParam("rowNum")
getDataType	the current datatype	getGridParam("datatype")
getRecords	the current number of records in grid.	getGridParam("records")
getMultiRow	array of id's of the selected rows when multiselect options is true. Empty array if no selection.	getGridParam("selarrrow")

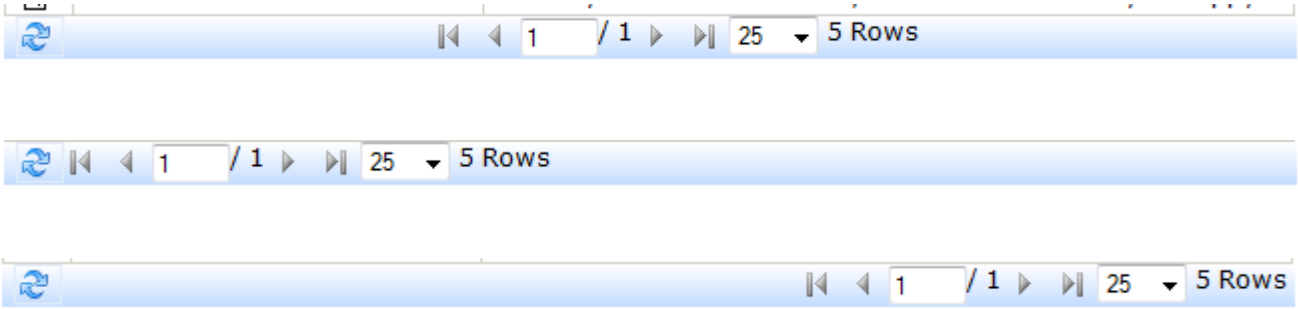
'Set' Methods	to set	Use Instead
setUrl	a new url, replacing the older.	setGridParam({url:newvalue})
setSortOrder	a new sort order	setGridParam({sortname:newvalue})
setPage	a new page number	setGridParam({page:newvalue})
setRowNum	a new number of requested rows	setGridParam({rowNum:newvalue})
setDataType	a new datatype	setGridParam({datatype:newvalue})

Pager (Navigation bar)

The pager, or Navigation Bar, for a basic grid is defined first in html -- normally, but not necessarily, placed so it appears at the bottom of the grid. Note that it is a , not a <table>

```
<body>
<table id="list" class="scroll"></table>
<div id="pager" class="scroll" style="text-align:center;"></div>
</body>
```

In this example, the pager controls are centered, but they could be aligned left or right to suit your preferences, as shown in these three examples:



The pager is then connected to the grid with a grid property:

```
pager: jQuery('#pager'),
```

or

```
pager: 'pager_id',
```

Properties

Several properties of the grid govern the function and appearance of the Navigation bar:

Property	Type	Description	Default
page	integer	The requested initial page number when we use datatypes xml or json (data returned from server)	1
pager	DOM element or string	Sets the pager bar for the grid. Must be a valid html element. If the element has class “scroll”, then the width is equal to the grid. Usage: If parameter is a DOM element, jQuery("#mypager"); if using a string, "mypager", where mypager is the id of the pager. Note the missing "#"	
recordtext	string	Displays the text associated with the display of total records	Rows
rowList	array	This parameter construct a select box element in the pager in which the user can change the number of the visible rows.	empty array
rowNum	integer	The initial number of rows that must be returned from the server	20

Events

One event of the grid ralates to the Navigation bar:

Event	Parameters	Description
onPaging	pgButton	This event fires after click on [page button] and before populating the data. Also works when the user enters a new page number in the page input box (and presses [Enter]). pgbutton(string) can be - first,last,prev,next

Methods

Adding buttons to the Navigation Bar is controlled by a method of the grid.

To use this method we need to enable the form editing feature. For more information refer to Installation

Calling Convention:

```
jQuery("#grid_id").navGrid("#pager",{parameters})
```

Where:

- *grid_id* - the id of the already constructed jqGrid.
- *pager* - the id of the navigation bar
- *parameters* - an array of settings, defined below

The *navGrid* method accepts the following settings to govern which buttons appear on the Navigation bar; any of them may be set to true or false. The default for all is true, but may be changed by, for example

```
{refresh: true, edit: true, add: true, del: false, search: true}
```

or

```
{del: false}
```

Custom Buttons

An additional method *navButtonAdd* (new in Version 3.2) supports adding custom buttons. This method must be chained with the setting of the Standard Buttons:

Calling Convention:

```
jQuery("#grid_id").navGrid("#pager",{standard parameters}).navButtonAdd("#pager",{custom parameters});
```

The Custom parameters are

```
{ caption: 'NewButton', buttonimg: '', onClickButton: null, position: "last" }
```

where

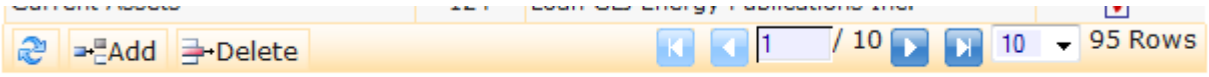
- *caption*: (string) the caption of the button, can be a empty string.
- *buttonimg*: (string) full path to valid image. If empty string, no image will be attached.
- *onClickButton*: (function) action to be performed when a button is clicked. Default null.
- *position*: ("first" or "last") the position where the button will be added (i.e., before or after the standard buttons)

Multiple buttons can be added by continuing the chain.

```
jQuery("#grid_id").navGrid('#Pager',{
        edit:false,add:false,del:false,search:false
      }).navButtonAdd('#Pager',{
        caption:"Add", buttonimg:"fullpath/row_add.gif", onClickButton: function(){ alert("Adding
Row")}, position:"last"
        }).navButtonAdd('#Pager',{
          caption:"Del", buttonimg:"fullpath/row_del.gif", onClickButton: function(id){
alert("Deleting Row: "+id)}, position:"last"
         });
```

Example I

We want to mimic the look of form editing when in-line editing (i.e., showing buttons on the Navigation bar rather than in the toolbar or on the form), like this:

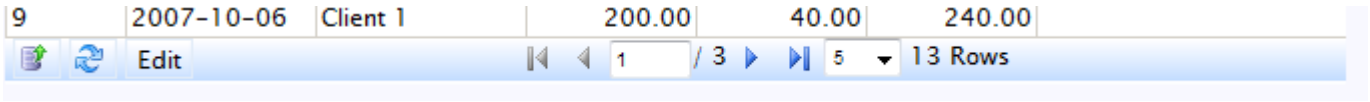


The code:

```
jQuery("#grid_id").navGrid('#Pager',{
        edit:false,add:false,del:false,search:false
      }).navButtonAdd('#Pager',{
        caption:"Add", buttonimg:"fullpath/row-insert-under.gif", onClickButton: function(){
        var datarow = {name1: value1, name2: value2', ...};
        var su=jQuery("#grid_id").addRowData("X",datarow,"last");
        if(su) { jQuery("#grid_id").setSelection('X') };   }, position:"last"
        }).navButtonAdd('#Pager',{
          caption:"Delete", buttonimg:"fullpath/row-delete.gif", onClickButton: function(){
          var gr = jQuery("#grid_id").getGridParam("selrow");
          if( gr != null ) {
            jQuery("#grid_id").delGridRow(gr,{afterSubmit: function(xhr,postdata){ alert ('After
Submit: ' + postdata); return [true]},
        url: 'delete.php'});
          } else {
            alert("Please Select Row to delete!");
          };
         }, position:"last"
         });
```

Example II

This example shows uses one of the methods new to version 3.2 to synchronize the grid with a form manually defined in html. (A button on the form moves the data back again).



The code:

```
jQuery("#tbleditable").navGrid('#pcustbut',{edit:false,add:false,del:false})
   .navButtonAdd('#pcustbut',{caption:"Edit",
   onClickButton:function(){
      var gsr = jQuery("#custbut").getGridParam('selrow');
      if(gsr){
         jQuery("#custbut").GridToForm(gsr,"#order");
      } else {
         alert("Please select Row")
      }
}
});
```

Form Editing

jqGrid supports creating a form, on the fly, to enter or edit grid data.

To do this we need to enable this feature. For more information refer to Installation

Properties

All the properties of the grid are the same as these for Inline editing -- see Inline Editing: Properties -- with the addition of the following option in the *colModel*:

editrules
Calling Convention:

```
{edithidden:true(false), required:true(false), number:true(false), minValue:val, maxValue:val}
```

With this option we can:

1. edit, in the form, fields that are hidden in the grid. If the field is hidden in the grid and *edithidden* is set to true, the field can be edited when add or edit methods are called.
2. perform a client-side validation in the formedit. This is done with:
    - required: true(false) - if set to true, the value will be checked and if empty, an error message will be displayed.
    - number: true(false) - if set to true, the value will be checked and if this is not a number, an error message will be displayed.
    - minValue: valid number - if set, the value will be checked and if the value is less than this, an error message will be displayed.
    - maxValue: valid number - if set, the value will be checked and if the value is more than this, an error message will be displayed.

Searching

Searching is way of querying data using different criteria. With this method we can perform a single field search.

This method uses colModel names and url parameters from jqGrid and so can be called only on already constructed grid.

Calling Convention:

```
jQuery("#grid_id").searchGrid( properties );
```

Where:

- *grid_id* is the id of the parent grid
- *properties* is an array of settings in *name: value* pairs format.

Properties

Property	Description	Default
top	the initial top position of search dialog	0
left	the initial left position of search dialog	0
width	the width of search dialog	300
height	the height of Search dialog	200
modal	sets dialog in modal mode	false
drag	sets the dialog to dragable	true
Find	the text of the button clicked to start the Find	"Find"
Clear	the text of the button when you click to clear search string	"Reset"
dirty	applicable only in navigator	false

Notes

If the top and left off-set properties are not set, the dialog appears at the upper left corner of the grid. Top and left off-sets are in relation to the viewing window, not the grid, so {top:10, left:10} will be indented slightly from the window, and may be nowhere near the grid.

To exclude a field from the search possibilities, set the *search* option in colModel to false. Example:

```
colModel[{name:'somename'…, search:false}…]
```

When the find button is clicked, jqGrid adds three parameters to the url, in *name=value* pairs:

- *sField*: the 'searchField', the value comes from the *index* in colModel
- *sValue*: the 'searchString', the value is the entered value
- *sOper*: the 'searchOper', the value is the type of search - see sopt array, below

---

Translation string for the search options:

```
odata : ['equal', 'not equal', 'less', 'less or equal','greater','greater or equal', 'begins
with','ends with','contains' ],
```

If you want to change or remove the order change it in sopt:

```
sopt: null // ['bw','eq','ne','lt','le','gt','ge','ew','cn']
```

By default all options are allowed. The codes are as follows:

bw - begins with (LIKE val%)
eq - equal (=)
ne - not equal (<>)
lt - little (<)
le - little or equal (<=)
gt - greater (>)
ge - greater or equal (>=)
ew - ends with (LIKE %val)
cn - contain (LIKE %val%)

Typically this method is applied to the click action of a button or link. For example,

```
jQuery("#bsdata").click(function(){
            jQuery("#search").searchGrid( {sopt:['cn','bw','eq','ne','lt','gt','ew']} );
});
```

We can set common options for all search dialogs using the $.jgrid.search object with $extend()

The default values are:

```
jQuery.jgrid.search = {
    caption: "Search...",
    Find: "Find",
    Reset: "Reset",
    odata : ['equal', 'not equal', 'less', 'less or equal','greater','greater or equal', 'begins
with','ends with','contains' ]
});
```

jQuery.extend($.jgrid.search,{Find:'Search'}) will replace the text of search button from Find to Search.

Add Row

This method can be used to add data to the server.

This method uses colModel and editurl parameters from jqGrid

Calling Convention:

```
jQuery("#grid_id").editGridRow( "new", options );
```

The options are the same as those in Edit row.

Notes

jqGrid adds two parameters to the values that are posted to the server:

- id=_empty and
- oper=add

to identify to the server that the operation is an insert.

Edit Row

This method is the same as inline editing except that the data is represented in form via modal dialog.

This method uses *colModel* and *editurl* properties from jqGrid

Calling Convention:

```
jQuery("#grid_id").editGridRow( rowid, properties );
```

where

- *grid_id*: the *id* of the parent grid
- *rowid*: the *id* of the row to edit
- *properties*: an array of *name: value* pairs, including any of the following properties or events.

Properties

Property	Description	Default
top	the initial top position of confirmation dialog	0
left	the initinal left position of confirmation dialog	0
width	the width of confirmation dialog	300
height	the height of confirmation dialog	200
modal	sets dialog in modal mode	false
drag	the dialog is dragable	true
msg	message to display when deleting the row	"Delete selected row(s)"
addCaption	the caption of the dialog if the mode is adding	"Add Record"
editCaption	the caption of the dialog if the mode is editing	"Edit Record"
bSubmit	the text of the button when you click to delete	"Submit"
bCancel	the text of the button when you click to close dialog	"Cancel"
url	url where to post data. If set, replaces the editurl	
processData	Words displayed when posting data	"Processing..."
closeAfterAdd	when add mode, close the dialog after add record	false
clearAfterAdd	when add mode, clear the data after adding data	true
closeAfterEdit	when in edit mode, close the dialog after editing	
reloadAfterSubmit	reload grid data after posting	true

Events

Event	Description	Default
onInitializeForm	fires once when creating the data for editing and adding. Receives, as parameter, the id of the constructed form.	null
beforeInitData	fires before initialize the form data. Receives, as parameter, the id of the constructed form.	null
beforeShowForm	fires before showing the form; receives as Parameter the id of the constructed form.	null
afterShowForm	fires after showing the form; receives as Parameter the id of the constructed form.	null
beforeSubmit	fires before the data is submitted to the server. Parameter is of type id=value1,value2,... When called the event can return array where the first parameter can be true or false and the second is the message of the error if any. Example: `[false,"The value is not valid"]`	null

afterSubmit	fires after response has been received from server. Typically used to display status from server (e.g., the data is successfully saved or the save cancelled for server-side editing reasons). Receives as parameters the data returned from the request and an array of the posted values of type id=value1,value2	null

Notes

If the top and left off-set properties are not set, the dialog appears at the upper left corner of the grid. Top and left off-sets are in relation to the viewing window, not the grid, so {top:10, left:10} will be indented slightly from the window, and may be nowhere near the grid.

---

jqGrid adds two parameters to the values that are posted to the server:

- id=rowid and
- oper=edit

to identify to the server that the operation is an update.

We can set common options for all add and/or edit dialogs using the $.jgrid.edit object with $.extend().

The default values are:

```
jQuery.jgrid.edit = {
    addCaption: "Add Record",
    editCaption: "Edit Record",
    bSubmit: "Submit",
    bCancel: "Cancel",
    processData: "Processing...",
    msg: {
        required:"Field is required",
        number:"Please enter valid number!",
        minValue:"value must be greater than or equal to ",
        maxValue:"value must be less than or equal to"
    }
};
```

Delete Row

With this method we can perform a delete operation at server side.

This method uses colModel and editurl parameters from jqGrid

Calling Convention:

```
jQuery("#grid_id").delGridRow( row_id_s, options );
```

where

- *grid_id*: the *id* of the parent grid

- *row_id_s*: the *id* of the row(s) to delete; can be a single value or list of ids separated by comma
- *options*: an array of *name: value* pairs, including any of the following properties or events.

Properties

Property	Description	Default
top	the initial top position of confirmation dialog	0
left	the initial left position of confirmation dialog	0
width	the width of confirmation dialog	300
height	the height of confirmation dialog	200
modal	sets dialog in modal mode	false
drag	the dialog is dragable	true
msg	message to display when deleting the row	"Delete selected row(s)"
caption	the caption of the dialog	"Delete Record"
bSubmit	the text of the button when you click to delete	"Delete"
bCancel	the text of the button when you click to close dialog	"Cancel"
url	url where to post data. If set, replaces the editurl	
reloadAfterSubmit	reload grid data after posting	true

Events

Event	Description	Default
beforeShowForm	fires before showing the form; receives as Parameter the id of the constructed form.	null
afterShowForm	fires after showing the form; receives as Parameter the id of the constructed form.	null
beforeSubmit	fires before the data is submitted to the server. Parameter is of type id=value1,value2,... When called the event can return array where the first parameter can be true or false and the second is the message of the error if any. Example: `[false,"The value is not valid"]`	null
afterSubmit	fires after response has been received from server. Typically used to display status from server (e.g., the data is successfully deleted or deletion cancelled for referential integrity reasons). Receives as parameters the data returned from the request and an array of the posted values of type id=value1,value2	null

Notes

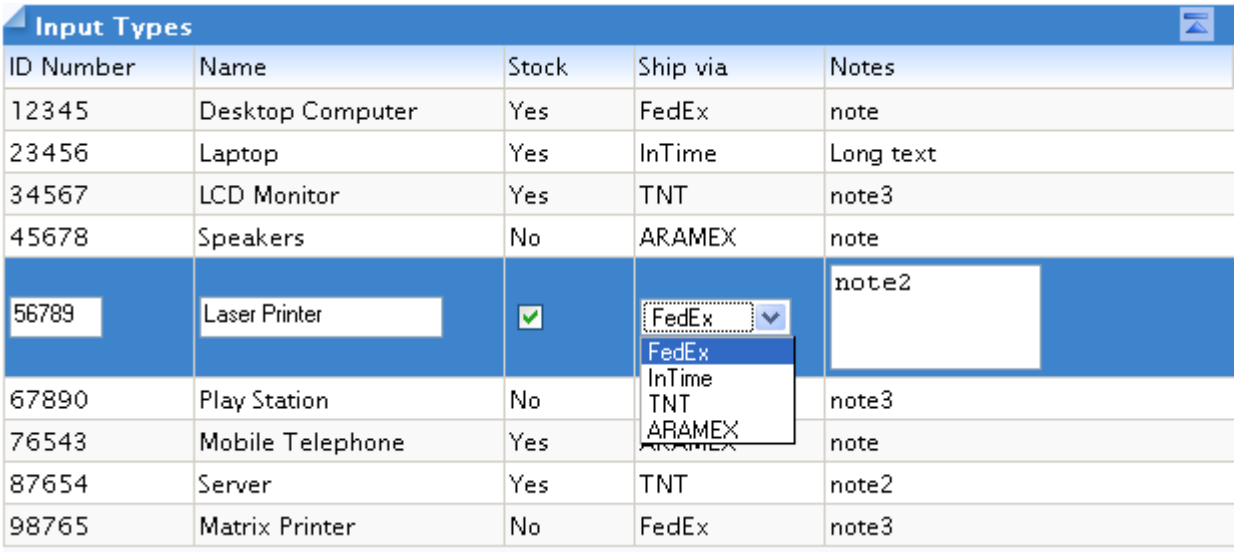
If the top and left off-set properties are not set, the dialog appears at the upper left corner of the grid. Top and left off-sets are in relation to the viewing window, not the grid, so {top:10, left:10} will be indented slightly from the window, and may be nowhere near the grid.

We can set common options for all delete dialogs using the $.jgrid.del object with $.extend().
The default values are:

```
jQuery.jgrid.del = {
    caption: "Delete",
    msg: "Delete selected record(s)?",
    bSubmit: "Delete",
    bCancel: "Cancel",
    processData: "Processing..."
};
```

Inline Editing

Inline editing is a quick way to update database information by supporting editing directly in the row of the grid, as shown in this example:



To do this we need to enable this feature. For more information refer to Installation.

This feature simply modifies some of the properties, and uses the methods, of the parent grid.

Properties

By default, columns are not editable so to use this option, we must add to the settings in the *colModel* for the columns we wish to be able to edit (it is not necessary to make all columns editable). There are four settings to consider:

- editable
- edittype
- editoptions, and
- editrules

For example,

```
{name:'stock', index:'stock', width:60,
editable:true,
edittype:"checkbox",
editoptions: {value:"Yes:No"}},
```

editable: defines if this field is editable (or not). Default is false. To make a field editable, set this to true: editable:true

edittype: defines the type of of the editable field. Possible values: 'text', 'textarea', 'select', 'checkbox'. The default value is 'text'.

editoptions: an array of allowed options (attributes) for the chosen *edittype*

Details of *edittype* and *editioptions* appear below.

If we are going to save the results of the edit into a server-side database, we also need to specify the server-side method that is going to accept the edited data. This is set as a grid option: *editurl*

What jqGrid does

edittype is 'text'

When *edittype* is 'text', jqGrid constructs a input tag of type text:

```
<input type="text" ...../>
```

In *editoptions* we can set all the possible attributes for this field. For example,

```
editoptions: {size:10, maxlength: 15}
```

will cause jqGrid to construct the following input

```
<input type="text" size="10" maxlength="15" />
```

In addition to the these settings, jqGrid adds the following:

- *id*: the id that is added to this element is a combination of the id of the row and the name - rowid_name
- *name*: the name from colModel
- *value*: the contents of the cell.

Consider the example above and suppose that the *id* of the row is 12 and *name* is invdate then the result is:

```
<input type="text" id="12_invdate" name="invdate" size="10" maxlength="15" value="someval"/>
```

edittype is 'textarea'

When *edittype* is 'textarea', jqGrid constructs a input tag of type textarea

```
<input type="textarea" .../>
```

In *editoptions* we can add additional attributes to this type. Typically, these govern the size of the box:

```
editoptions:{rows:"2",cols:"10"}
```

To these attributes jqGrid adds *id* and *name* attributes just as for text type.

edittype is 'checkbox'

When *edittype* is 'checkbox', jqGrid constructs a input tag as follows:

```
<input type="checkbox" .../>
```

editoptions is used to define the checked and unchecked values. The first value is checked. For example

```
editoptions: { value:"Yes:No" }
```

defines a checkbox in which when the value is Yes the checkbox becomes checked, otherwise unchecked. This value is passed as parameter to the editurl.

To these attributes jqGrid adds *id* and *name* attributes just as for text type.

***edittype* is 'select'**

When *edittype* is 'select', jqGrid constructs a input tag as follows:

```
<select>
<option value='val1'> Value1 </option>
<option value='val2'> Value2 </option>
...
<option value='valn'> ValueN </option>
</select>
```

To construct this element, *editoptions* must contain an array with value:label pairs. In every pair, the value is separated from the label with a colon (:); pairs are separated with a semi-colon (;). For example,

```
editoption: { value: "FE:FedEx; IN:InTime; TN:TNT" }
```

will construct

```
<select>
<option value='FE'> FedEx </option>
<option value='IN'> InTime </option>
<option value='TN'> TNT </option>
</select>
```

To this element, jqGrid adds the *id* and *name* attributes as above.

With these settings in the *colModel*, we next need to look at the methods that support inline editing.

Methods

For inline editing, we have three additional methods (of the Grid API) available:

- editRow
- saveRow
- restoreRow

These methods can be called, of course, only on an already-constructed grid, from a button click or from an event of the grid itself:

```
onSelectRow: function(id){
  if(id && id!==lastSel){
    jQuery('#tbleditable').restoreRow(lastSel);
    lastSel=id;
  }
  jQuery('#tbleditable').editRow(id, true);
},
```

In this example, if another was row being edited and has not yet been saved, the original data will be restored and the row "closed" before "opening" the currently-selected row for editing (where *lastSel* was previously defined as a var).

editRow

Calling convention:

```
editRow(rowid, keys, oneditfunc, succesfunc, url, extraparam, aftersavefunc, onerrorfunc)
```

where

- *rowid*: the id of the row to edit
- *keys*: when set to true we can use [Enter] key to save the row and [Esc] to cancel editing.
- *oneditfunc*: fires after successfully accessing the row for editing, prior to allowing user access to the input fields. The row's *id* is passed as a parameter to this function.

If *keys* is true, then the remaining settings -- *succesfunc*, *url*, *extraparam*, *aftersavefunc* and *onerrorfunc* -- are passed as parameters to the *saveRow* method when the [Enter] key is pressed (*saveRow* does not need to be defined as jqGrid calls it automatically). For more information see *saveRow* method below.

When this method is called on particular row, jqGrid reads the data for the editable fields and constructs the appropriate elements defined in *edittype* and *editoption*.

saveRow

Calling convention:

```
saveRow (rowid, succesfunc, url, extraparam, aftersavefunc, onerrorfunc)
```

where

- *rowid*: the id of the row to save
- *succesfunc*: if defined, this function is called immediately after the request is successful. To this function is passed the data returned from the server. Depending on the data from server this function should return true or false.
- *url*: if defined, this parameter replaces the editurl parameter from options array.
- *extraparam*: an array of type *name: value*. When set these values are posted along with the other values to the server.
- *aftersavefunc*: if defined, this function is called after the data is saved to the server. Parameters passed to this function are the *rowid* and the result from the request.
- *onerrorfunc*: if defined, this function is called after the data is saved to the server. Parameters passed to this function are the *rowid* and the result from the request.

When this method is called, the data from the particular row is POSTED to the server in format *name: value*, where the *name* is a name from colModel and the *value* is the new value. jqGrid also adds, to the posted data, the pair *id: rowid*.

restoreRow

Calling convention:

```
restoreRow(rowid)
```

where

- *rowid* is the row to restore

This method restores the data to original values before the editing of the row.

Example

```
<html>
<head>
<script type="text/javascript">
jQuery(document).ready(function(){
  var lastsel2
  jQuery("#rowed5").jqGrid({
    datatype: "local",
    height: 250,
    colNames:['ID Number','Name', 'Stock', 'Ship via','Notes'],
    colModel:[
      {name:'id',index:'id', width:90, sorttype:"int", editable: true},
      {name:'name',index:'name', width:150,editable: true,editoptions:{size:"20",maxlength:"30"}},
      {name:'stock',index:'stock', width:60, editable: true,edittype:"checkbox",editoptions:
{value:"Yes:No"}},
      {name:'ship',index:'ship', width:90, editable: true, edittype:"select",
editoptions:{value:"FE:FedEx;IN:InTime;TN:TNT;AR:ARAMEX"}},
      {name:'note',index:'note', width:200, sortable:false,editable: true,edittype:"textarea",
editoptions:{rows:"2",cols:"10"}}
            ],
    onSelectRow: function(id){
      if(id && id!==lastsel2){
        jQuery('#rowed5').restoreRow(lastsel2);
        jQuery('#rowed5').editRow(id,true);
          lastsel2=id;
      }
    },
    editurl: "server.php",
    caption: "Input Types"
  });
  var mydata2 = [
    {id:"12345",name:"Desktop Computer",note:"note",stock:"Yes",ship:"FedEx"},
    {id:"23456",name:"Laptop",note:"Long text ",stock:"Yes",ship:"InTime"},
    {id:"34567",name:"LCD Monitor",note:"note3",stock:"Yes",ship:"TNT"},
    {id:"45678",name:"Speakers",note:"note",stock:"No",ship:"ARAMEX"},
    {id:"56789",name:"Laser Printer",note:"note2",stock:"Yes",ship:"FedEx"},
    {id:"67890",name:"Play Station",note:"note3",stock:"No", ship:"FedEx"},
    {id:"76543",name:"Mobile Telephone",note:"note",stock:"Yes",ship:"ARAMEX"},
    {id:"87654",name:"Server",note:"note2",stock:"Yes",ship:"TNT"},
    {id:"98765",name:"Matrix Printer",note:"note3",stock:"No", ship:"FedEx"}
    ];
  for(var i=0;i"#rowed5").addRowData(mydata2[i].id,mydata2[i]);
});
</script>
</head>
<body>
<table id="rowed5" class="scroll"></table>
</body>
</html>
```

Will produce the following:

Input Types

ID Number	Name	Stock	Ship via	Notes
12345	Desktop Computer	Yes	FedEx	note
23456	Laptop	Yes	InTime	Long text
34567	LCD Monitor	Yes	TNT	note3
45678	Speakers	No	ARAMEX	note
56789	Laser Printer	☑	FedEx	note2
67890	Play Station	No		note3
76543	Mobile Telephone	Yes		note
87654	Server	Yes	TNT	note2
98765	Matrix Printer	No	FedEx	note3

FedEx
InTime
TNT
ARAMEX

Multi-Selecting Items

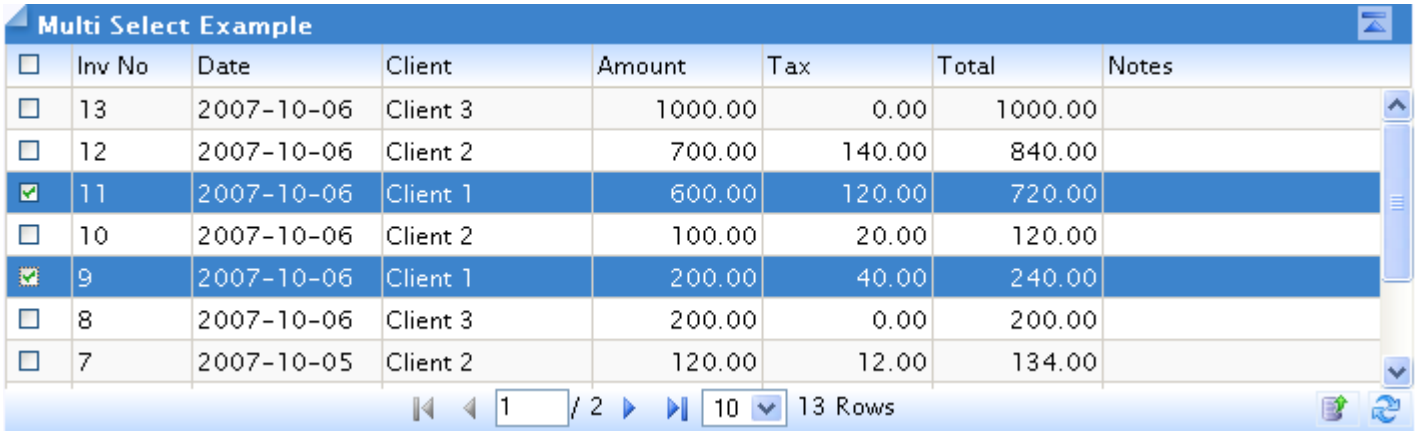
Multi-selection is a way to select more than one row in grid so that some action can be performed on all of them at once. jqGrid allows more than one way of multi-selection.

There are two settings to consider:

- multiselect
- multikey

Simple Multi-Select

The default value of *multiselect* is false, which disables this feature. When multiselect is set to true an additional column is added at left side of the grid. This adds 28px to the grid's width. When the grid is constructed the content of this column if filled with a check box element. When we select some row the check box's state becomes checked. When we select another row the previous row does not change its state. When we click on a row that is selected, the state becomes unchecked and the row is unselected.



As seen in the figure above, in the header layer we have a common check box. When we check this box all the rows will be selected. When we unchecked this box all the rows are unchecked.

Multi-Select With Key

Another feature of jqGrid is that we can define a key that must be pressed in order to make a multiselection. In other words, the selection is done only when the user holds down the defined key. To achieve this we should use *multikey* option in the options array. For example,

```
multikey: 'altKey'
```

In this case the multiselection is done only when the user holds down the "Alt" key. The possible values are: 'shiftKey', 'altKey', and 'ctrlKey'.

Identifying the Selected Rows

To obtain selected rows we can use *getMultiRow* method. Using our example we can write this

```
jQuery("#grid_id").getMultiRow();
```

which will return an array with the selected rows (i.e., ["11","9"] from the figure above). The values in array are the id's of the selected rows.

To retrieve a single row, the last one selected, we can use *getGridParam(selrow)*

```
jQuery("#grid_id").getGridParam('selrow');
```

This returns the id of the last selected row as a scalar value.

Dynamically Enabling and Disabling Multi-select

To dynamically disable multiselect:

```
jQuery("#grid_id").setGridParam({multiselect:false}).hideCol('cb');
```

to enable multi-select:

```
jQuery("#grid_id").setGridParam({multiselect:true}).showCol('cb');
```

Where

- *grid_id* is to be replaced by the name of your grid, but
- *cb* is a keyword, not to be replaced

Note that if multiselect initially is not set we can not enable it using the code above. To do that we should initially enable it in the jqGrid options and disable it using the code above.

Subgrids

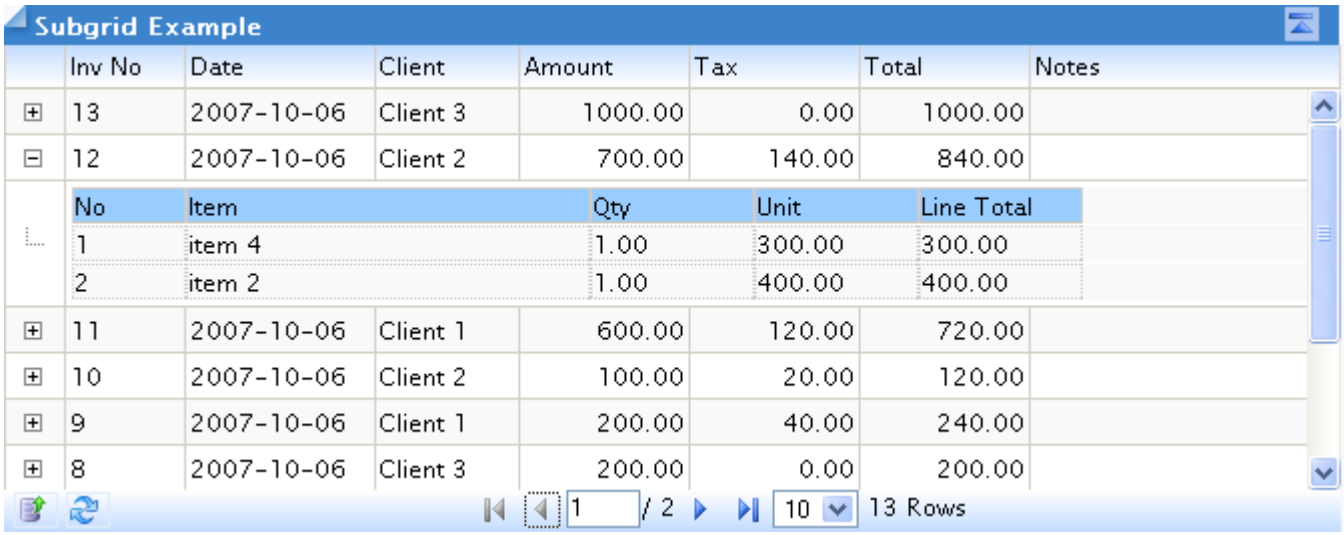
There are times when we need to be able to easily display (or edit) records that are the children of the records in the grid and, of course, we would want to show only those records that are related to the selected record in the grid, not all of them at once.

jqGrid offers three ways of handling child records:

1. a subGrid
2. a grid as a subGrid, and
3. Master/Detail Grids

A Subgrid

Using a subGrid is the easiest method for displaying data from child records, as shown in this sample:



Clicking on the plus or minus icons beside each "parent" record reveals or hides the associated "child" records.

But this approach does have limitations: data in a subgrid cannot be sorted, or edited, and the alignment of the data in the columns is always "left". If you need more power than what this offers, then consider using a grid as a subgrid or, even, Master/Detail grids. But for a quick way to display details, follow the method described here.

Defining a SubGrid

When defining a subgrid, three options need to be considered.

- subGrid
- subGridUrl

- subGridModel

subGrid
This is a boolean value which enables or disables the subgrid. If the grid is enabled a additional column at left side is added to the main grid. This column contains a 'plus' image which indicate that the user can click on it to expand the row. By default all rows are collapsed.

Default value for this option is false. To enable a subgrid set it to true.

subGridUrl
This option points to the file from which we get the data for the subgrid. jqGrid adds the id of the row to this url as parameter. If there is a need to pass additional parameters, use the params option in *subGridModel* (see below)

subGridModel
An array in which we describe the column model for the subgrid data. The syntax is

```
subGridModel : [
{ name : ['name_1','name_2',...,'name_n'],
width : [width_1,width_2,...,width_n] ,
params : [param_1,...,param_n]}
]
```

where

- *name*: an array containing the labels of the columns of the subgrid.
- *width*: an array containing the width of the columns. This array should have the same length as in name array.
- *params*: an array in which we can add a name from main grid's *colModel* to pass as additional parameter to the subGridUrl.

The subgrid model is described in xmlReader and jsonReader. For xml data, the mapping would follow this example

```
xmlReader : {
root: "rows",
row: "row",
page: "rows>page",
total: "rows>total",
records : "rows>records",
repeatitems: true,
cell: "cell",
id: "[id]",
subgrid: {root: "rows", row: "row", repeatitems: true, cell: "cell"}
}
```

and for json mapping, like the following:

```
jsonReader : {
root: "rows",
page: "page",
total: "total",
records: "records",
repeatitems: true,
cell: "cell",
```

```
id: "id",
subgrid: {root: "rows", repeatitems: true, cell: "cell"}
}
```

The mapping rules are the same as those in the basic grid.

For more information see the discussion of xml and JSON in Data Types.

An Example

Continuing to use the example from the tutorial, let's suppose that there is a need to display the line items for each invoice in a subgrid. The Java script code should look like this.

```
<script type="text/javascript">
jQuery(document).ready(function(){
  jQuery("#list").jqGrid({
    url:'example.php',
    datatype: 'xml',
    colNames:['Inv No', 'Date', 'Amount', 'Tax', 'Total', 'Notes'],
    colModel :[
      {name:'invid', index: 'invid', width: 55},
      {name:'invdate', index: 'invdate', width:  90},
      {name:'amount', index: 'amount', width: 80, align: 'right'},
      {name:'tax', index: 'tax', width: 80, align: 'right'},
      {name:'total', index: 'total', width: 80,align: 'right'},
      {name:'note', index: 'note', width: 150, sortable: false} ],
    pager: jQuery('#pager'),
    rowNum:10,
    rowList:[10,20,30],
    sortname: 'id',
    sortorder: "desc",
    viewrecords: true,
    imgpath: 'themes/basic/images',
    caption: "My first grid,
    subGrid: true,
    subGridUrl : "subgrid.php",
    subGridModel [ {
      name:  ['No', 'Item', 'Qty', 'Unit', 'Line Total'],
      width : [55, 200, 80, 80, 80],
      params: ['invdate'] }
    ]
  });
});
</script>
```

The next step is to configure the subgrid.php file. The example is in PHP and MySQL

```
<?php
// get the id passed automatically to the request
$id = $_GET['id'];
// get the invoice data passed to this request via params array in
//subGridModel. We do not use it here - this is only demostration
$date_inv = $_GET['invdate'];


// connect to the database
$db = mysql_connect($dbhost, $dbuser, $dbpassword) or die("Connection Error: " . mysql_error());

mysql_select_db($database) or die("Error conecting to db.");

// construct the query
```

```
$SQL = "SELECT num, item, qty, unit FROM invlines WHERE id=".$id." ORDER BY item";
$result = mysql_query( $SQL ) or die("Couldn?t execute query.".mysql_error());

// set the header information
if ( stristr($_SERVER["HTTP_ACCEPT"],"application/xhtml+xml") ) {
            header("Content-type: application/xhtml+xml;charset=utf-8");
} else {
            header("Content-type: text/xml;charset=utf-8");
}

echo "<?xml version='1.0' encoding='utf-8'?>";
echo "<rows>";
// be sure to put text data in CDATA
while($row = mysql_fetch_array($result,MYSQL_ASSOC)) {
            echo "<row>";
            echo "<cell>". $row[num]."</cell>";
            echo "<cell><![CDATA[". $row[item]."]]></cell>";
            echo "<cell>". $row[qty]."</cell>";
            echo "<cell>". $row[unit]."</cell>";
            echo "<cell>". number_format($row[qty]*$row[unit],2,'.',' ')."</cell>";
            echo "</row>";
}
echo "</rows>";

?>
```

After this, the grid will look like the example at the top of this page.

Dynamically Enabling or Disabling a Subgrid

A subGrid can be enabled (or disabled) dynamically (to respond to changes in the data in the main grid, for example).

To disable a subgrid:

```
$("#grid_id").hideCol('subgrid');
```

to enable a subgrid:

```
$("#grid_id").showCol('subgrid');
```

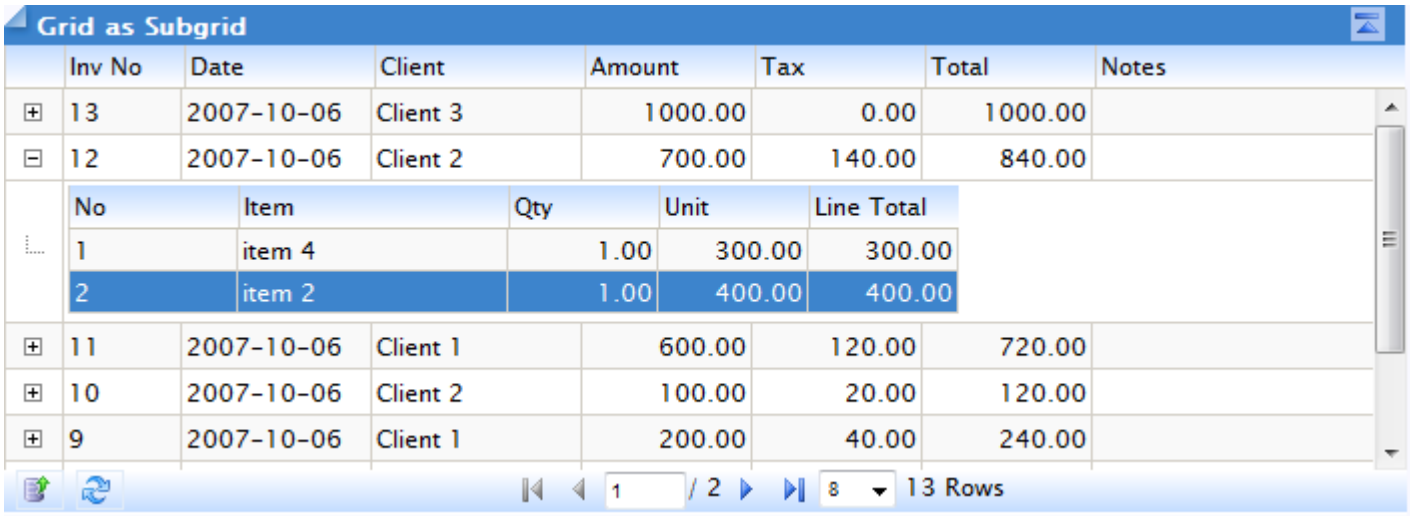
Where

- *grid_id* is to be replaced by the name of your grid, but
- *subgrid* is a keyword, not to be replaced

To make this work, multiselect must be initially set to true in the jqGrid properties; only then can we enable and disable it using the code above.

A Grid as Subgrid

In this alternative to a subGrid, we use the subGrid functions of the main grid to create not a subGrid, but another grid, with all of the power and capacity of the main grid but appearing, as before, under the "parent" record with the same ability to reveal and hide it.



Note that in this sample, the focus is on the second "child" row, something that cannot be done in a true subGrid, and that the numeric columns are now right-aligned.

Defining a Grid as a subGrid

We use two events described in options array: *subGridRowExpanded* and *subGridRowColapsed* [note the unconventional spelling].

When these events are defined the population of the data in the subgrid is not executed. This way we can use the subGridUrl to get our custom data and put it into the expanded row. Having this it is easy to construct another grid which will act as subgrid.

Here is this technique. We again use our example.

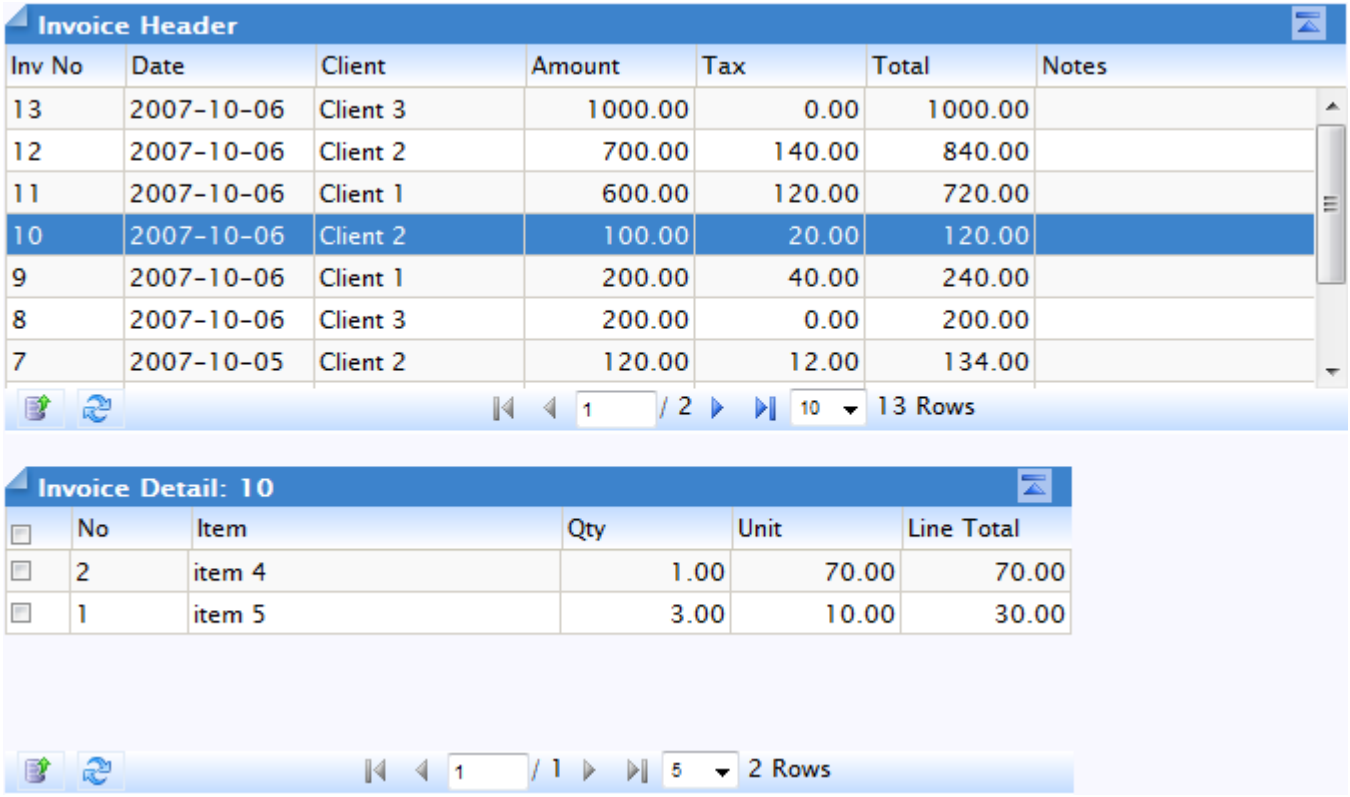
```
<script type="text/javascript">
jQuery("#listsg11").jqGrid({
  url:'example.php',
  datatype: "xml",
  height: 200,
  colNames:['Inv No','Date', 'Amount','Tax','Total','Notes'],
  colModel :[
    {name:'invid',index:'invid', width:55},
    {name:'invdate',index:'invdate', width:90},
    {name:'amount',index:'amount', width:80, align:'right'},
    {name:'tax',index:'tax', width:80, align:'right'},
    {name:'total',index:'total', width:80,align:'right'},
    {name:'note',index:'note', width:150, sortable:false} ],
  rowNum:8,
  rowList:[8,10,20,30],
  imgpath: gridimgpath,
  pager: jQuery('#pager'),
```

```
  sortname: 'id',
  viewrecords: true,
  sortorder: "desc",
  subGrid: true,
  subGridRowExpanded: function(subgrid_id, row_id) {
    // we pass two parameters
    // subgrid_id is a id of the div tag created within a table
    // the row_id is the id of the row
    // If we want to pass additional parameters to the url we can use
    // the method getRowData(row_id) - which returns associative array in type name-value
    // here we can easy construct the following
    var subgrid_table_id;
    subgrid_table_id = subgrid_id+"_t";
      jQuery("#"+subgrid_id).html("<table id='"+subgrid_table_id+"' class='scroll'></table>");
        jQuery("#"+subgrid_table_id).jqGrid({
          url:"subgrid.php?q=2&id="+row_id,
          datatype: "xml",
          colNames: ['No','Item','Qty','Unit','Total'],
          colModel: [
            {name:"num",index:"num",width:80,key:true},
            {name:"item",index:"item",width:130},
            {name:"qty",index:"qty",width:80,align:"right"},
            {name:"unit",index:"unit",width:80,align:"right"},
            {name:"total",index:"total",width:100,align:"right",sortable:false}
                        ],
          height: 100%,
          rowNum:20,
          imgpath: gridimgpath,
          sortname: 'num',
          sortorder: "asc"
        })
   }
});
</script>
```

Note that subGridRowColapsed is not defined. This is true because when the row is collapsed the contents of the div tag are removed.

Master/Detail Grids

A third option is to present two separate grids and synchronize the contents of the second (the 'Detail' grid) with the selected row of the 'Master' grid.



Again, the Detail grid is a full-feature grid: you can do whatever you want with it in terms of configuration and function.

Defining Master/Details Grids

First, define two grids in your HTML; in our example, Invoice Header and Invoice Detail (the ids used here are not significant, you can call them whatever you want):

```
<table id="master" class="scroll"></table>
<div id="pagermaster" class="scroll" style="text-align:center;"></div>
<table id="detail" class="scroll"></table>
<div id="pagerdetail" class="scroll" style="text-align:center;"></div>
```

Then, in the definition of your Master grid, add the following, which says that whenever a row is selected in the Master grid, the Details grid is sychronized.

```
   onSelectRow: function(id) {
    if(id == null) {
      id=0;
        if(jQuery("#details").getRecords()>0) {

jQuery("#details").setGridParam({url:"subgrid.php?q=1&id="+id,page:1}).trigger("reloadGr
id");
```

```
        }
      } else {

jQuery("#details").setGridParam(url:"subgrid.php?q=1&id="+id,page:1).trigger("reloadGrid
);
      }
    }
```

Notice this passes the *id* of the Master row to be used as a parameter in the url to retrieve the Details data. The value of the *setGridParam({url: subgrid.php?q=1... })* property, of course, will need to be changed to meet your needs.

Notice also how *setGridParam* is used to set two parameters of the grid at once (the *url* and the *page* number) and how triggering the grid reload is chained.

Now these grids can be defined and operated independently while still being co-ordinated.

User Modules

Tony, the developer of jqGrid, welcomes contributions from the user community to enhance the functions of jqGrid. All will receive acknowledgement here.

Posting Data

Author: Paul Tiseo

Module name: grid.postext.js

Description: The main purpose of this module is to manipulate the parameters passed to the to url via an array and to get user-defined data from the response. For user-defined data, please refer to Data Types.

A new option, postData, is added to the option array of the grid. By default this is an empty array. The values of this array are added via $.extend to the ajax request.

Installation: To enable this module you should enable it in jquery.jqGrid.js.

Manipulating parameters

To manipulate the values of the array we can use the following methods:

jQuery("#grid_id").getPostData() returns all parameters passed to the grid url. The returned value is array of type name:value.

jQuery("#grid_id").setPostData(newdata) sets a new set of parameters overriding the existing ones.
newdata should be array of type name:value. Example {myparam:"myvalue"} Note that the page, rowNum, sortorder, sortname parameters are not changed. To change these use setGridParam method.

jQuery("#grid_id").appendPostData(newdata) replaces or appends new parameters to the array. *newdata* should be array of type name;value

jQuery("#grid_id").setPostDataItem(Key, Val) sets new or replaces the value of the existing item in the array. Key is the name and Val is the value of the item.

jQuery("#grid_id").getPostDataItem(key) returns the value of the requested item with name key

jQuery("#grid_id").removePostDataItem(key) deletes a specified item with name = key from the array.