

# LINB4.0 Cookbook - Getting Started Guide

linb 2011-5-1 All rights reserved.

Preface .....	7
Chapter 1. Preparation .....	8
1.1. Download the package .....	8
1.2. The package folder .....	8
1.3. Glance at examples and API .....	9
Chapter 2. Hello World .....	10
2.1. The first application .....	10
2.2. Render onto a html node .....	11
2.3. Do it in Designer .....	12
2.4. Application loading process .....	17
2.5. Code Editor .....	19
2.5.1. Highlight code from Outline window .....	20
2.5.2. Code Folding .....	20
2.5.3. Code Intellisense .....	20
2.5.3.1. When context doesn't recognize the input string .....	21
2.5.3.2. Type dot after a recognizable variable .....	22
2.5.3.3. When use shortcut [Alt+1 ], or dbclick .....	23
2.5.4. Find the object definition code .....	23
2.5.5. Generate event code automatically .....	24
Chapter 3. Controls Facebook .....	25
3.1. Script testing environment .....	25
3.2. "Hello world" in env.html .....	26
3.3. Control creation and runtime update .....	27
3.4. Button related .....	28
3.4.1. onClick event .....	28
3.4.2. Boolean Controls .....	29
3.4.3. Link Control .....	29
3.5. Label related .....	30
3.6. Input related .....	30
3.6.1. setValue/setValue/getUIValue/setUIValue .....	30
3.6.2. Dirty Mark .....	31
3.6.3. Password Input .....	31
3.6.4. Multi-lines .....	32
3.6.5. Input validation .....	32
3.6.5.1. valueFormat property .....	32
3.6.5.2. beforeFormatCheck event .....	33
3.6.6. Dynamic input validation .....	33
3.6.7. Error Mark .....	33

3.6.7.1.	Default Error Mark.....	33
3.6.7.2.	Validation Tips .....	34
3.6.7.3.	Binding Validation .....	34
3.6.7.4.	Custom Error Mark .....	34
3.6.8.	Mask Input .....	35
3.6.9.	linb.UI.ComboInput .....	36
3.6.9.1.	Pop list for selection.....	36
3.6.9.2.	combobox, listbox and helpinput .....	36
3.6.9.3.	Date Picker.....	37
3.6.9.4.	Time Picker .....	38
3.6.9.5.	Color Picker .....	38
3.6.9.6.	File Picker .....	39
3.6.9.7.	Getter.....	39
3.6.9.8.	Custom Pop Window .....	40
3.6.9.9.	Command Buttons.....	41
3.6.10.	RichEditor .....	41
3.7.	List related .....	42
3.7.1.	A Simple one.....	42
3.7.2.	A little bit complicated.....	43
3.7.3.	RadioBox .....	44
3.7.4.	IconList and Gallery .....	44
3.7.5.	Item selection.....	45
3.7.6.	Container related.....	45
3.7.7.	Pane and Panel .....	46
3.7.8.	Block.....	47
3.8.	Dialog related.....	47
3.8.1.	Normal state.....	47
3.8.2.	Min and Max status.....	48
3.8.3.	Modal Mode.....	49
3.9.	Layout Control.....	50
3.10.	Multi-pages Controls .....	51
3.10.1.	noPanel property .....	52
3.10.2.	ButtonViews types .....	53
3.10.3.	Page selection.....	53
3.10.4.	Pages .....	54
3.10.4.1.	Close and options Button .....	54
3.10.4.2.	Add/Remove Pages .....	56
3.10.5.	Dynamic content loading .....	57
3.10.5.1.	onIniPanelView .....	57
3.10.5.2.	beforeUIValueSet/afterUIValueSet .....	57
3.11.	Menus and toolbars.....	58
3.11.1.	Pop Menu .....	58
3.11.2.	MenuBar .....	59
3.11.3.	Toolbars.....	60

3.12.	TreeBar and TreeView .....	61
3.12.1.	Three selection mode .....	61
3.12.1.1.	No-selection .....	61
3.12.1.2.	Single-selection .....	62
3.12.1.3.	Multi-selection .....	62
3.12.2.	Group Item .....	63
3.12.3.	Expand all nodes by default .....	63
3.12.4.	Mutex Expand .....	64
3.12.5.	Dynamic Destruction .....	65
3.12.6.	Dynamically loading .....	65
3.13.	TreeGrid .....	66
3.13.1.	Header and Rows .....	66
3.13.1.1.	Sets standard format .....	67
3.13.1.2.	Sets simplified format .....	67
3.13.2.	getHeader .....	67
3.13.3.	getRows .....	68
3.13.4.	Active Modes .....	69
3.13.4.1.	non-active appearance .....	70
3.13.4.2.	row-active appearance .....	70
3.13.4.3.	cell-active appearance .....	71
3.13.5.	Selection Mode .....	71
3.13.5.1.	Non-selection .....	71
3.13.5.2.	Single row selection .....	72
3.13.5.3.	Multi-row selection .....	73
3.13.5.4.	Single cell selection .....	73
3.13.5.5.	Multi-cells selection .....	73
3.13.6.	The Tree Grid .....	74
3.13.7.	Column config .....	75
3.13.7.1.	The first column .....	75
3.13.7.2.	Column width .....	76
3.13.7.3.	Drag&Drop to modify column width .....	76
3.13.7.4.	Drag&Drop to modify column position .....	77
3.13.7.5.	Default Sorting .....	77
3.13.7.6.	Custom Sorting .....	78
3.13.7.7.	Hide columns .....	78
3.13.7.8.	Setting Cell Types in column header .....	79
3.13.7.9.	column header style .....	79
3.13.7.10.	column header icon .....	80
3.13.7.11.	Update column header dynamically .....	81
3.13.8.	Row config .....	81
3.13.8.1.	Row height .....	81
3.13.8.2.	Drag&Drop to modify row height .....	82
3.13.8.3.	Setting cell type in row .....	82
3.13.8.4.	Row style .....	83

3.13.8.5.	Row numbers .....	83
3.13.8.6.	Custom row numbers .....	84
3.13.8.7.	Alternate Row Colors.....	85
3.13.8.8.	Group .....	85
3.13.8.9.	Preview and Summary region .....	86
3.13.8.10.	Update row dynamically .....	87
3.13.9.	Cell config.....	88
3.13.9.1.	Cell types .....	88
3.13.9.2.	Cell style .....	89
3.13.9.3.	Update cell dynamically.....	90
3.13.10.	Editable .....	90
3.13.10.1.	Editable TreeGrid.....	91
3.13.10.2.	Editable column .....	91
3.13.10.3.	Editable row .....	92
3.13.10.4.	Editable cell .....	92
3.13.10.5.	The Editor .....	93
3.13.10.6.	Custom the editor .....	94
3.13.11.	Add/Remove rows .....	95
3.14.	Other standard controls .....	96
3.14.1.	ProgressBar .....	96
3.14.2.	Slider.....	97
3.14.3.	Image .....	98
3.14.4.	PageBar.....	98
Chapter 4.	Data exchanging(Ajax) .....	99
4.1.	Fiddler.....	100
4.2.	To get the contents of the file.....	100
4.3.	Synchronous data exchange.....	100
4.4.	Cross-domain .....	101
4.4.1.	To monitor SAjax.....	101
4.4.2.	To monitor IAjax.....	102
4.5.	File Upload .....	103
4.5.1.	Selecting upload file with ComboInput .....	103
4.5.2.	Upload by IAjax.....	104
4.6.	A request wrapper for real application.....	104
4.7.	XML Data .....	105
4.8.	An overall example .....	105
Chapter 5.	Distributed UI .....	107
5.1.	Shows dialog from a remote file .....	107
5.2.	linb.Com and linb.ComFactory.....	108
5.2.1.	linb.ComFactory config .....	108
5.2.2.	linb.Com.Load .....	109
5.2.3.	newCom and getCom.....	110
5.2.4.	linb.UI.Tag .....	111
5.2.5.	Destroy com.....	111

5.2.6.	If com exists in memory .....	111
Chapter 6.	Some fundamental things .....	112
6.1.	Pop-up window .....	112
6.1.1.	alert window .....	112
6.1.2.	confirm window .....	112
6.1.3.	prompt window .....	113
6.1.4.	pop window .....	113
6.2.	Asynchronous execution .....	113
6.2.1.	asyRun .....	113
6.2.2.	resetRun .....	114
6.3.	Skin switcher .....	114
6.3.1.	Switch skin for whole application .....	114
6.3.2.	Change skin for a single control .....	114
6.4.	Locale switcher .....	115
6.5.	DOM Manipulation .....	115
6.5.1.	Node generation and insertion .....	116
6.5.2.	Attributes and CSS .....	116
6.5.3.	className .....	117
6.5.4.	Dom events .....	117
6.5.5.	Node Drag&Drop .....	118
6.5.5.1.	Drag&Drop profile .....	119
6.5.5.2.	Events in Drag&Drop .....	120
6.6.	linb.Template .....	121
6.6.1.	example 1 .....	121
6.6.2.	example 2 .....	122
6.6.3.	A SButton based on linb.Template .....	123
6.7.	About Debugging .....	124
6.7.1.	The code package for debugging .....	124
6.7.2.	Debugging Tools .....	124
6.7.3.	jsLinb Monitor Tools .....	125
Chapter 7.	Some typical issues .....	126
7.1.	Layout .....	126
7.1.1.	Docking .....	126
7.1.2.	linb.UI.Layout .....	126
7.1.3.	Relative Layout .....	127
7.2.	UI Control's Drag&Drop .....	129
7.2.1.	Drag&Drop control among containers .....	129
7.2.2.	List sorting 1 .....	129
7.2.3.	List sorting 2 .....	130
7.3.	Form .....	131
7.3.1.	Form 1 .....	131
7.3.2.	DataBinder .....	132
7.4.	Custom UI Styles .....	134
7.4.1.	Custom only one instance only - 1 .....	134

7.4.2.	Custom only one instance only - 2.....	135
7.4.3.	Custom only one instance only - 3.....	135
7.4.4.	Custom only one instance only - 4.....	135
7.4.5.	Custom only one instance only - 5.....	136
7.4.6.	Custom only one instance only - 6.....	136
7.4.7.	Custom style for an UI Class .....	137
7.4.8.	Custom style for all UI Class - skin .....	137
7.4.8.1.	First: Copy .....	137
7.4.8.2.	Second: Little by little, modify pictures and CSS.....	138
The end	.....	139

# Preface

"jsLinb" is a Cross-Browser JavaScript framework with cutting-edge functionality for rich web application. And "Visual JS" is a web based tool for AJAX RIA application UI rapid design and involved scripts programming. With this powerful builder, developers can build your web application just like what you do in VB or Delphi.

## Features & Resource

1. WYSIWYG GUI builder. Do everything by drag & drop. Significant development time reduction.
2. Source code editor Integrated (Code Intellisense, Code folding, Syntax Check and Undo/Redo).
3. More than 40 common components, including Tabs, Dialog, TreeGrid, TimeLine and many other web GUI components.
4. Rich client-side API, works with any backend (php, .Net, Java, python) or static HTML pages. Extremely easily to build php application with php server side wrappers.
5. Wide cross-browser compatibility, IE6+, firefox1.5+, opera9+, safari3+ and Google Chrome.
6. Detailed Manual and Full API Documentation with tons of samples. Ever increasing code snippets.
7. Compatible with jquery, prototype, mootools and other frameworks.
8. Dual licenses - Commercial License and LGPL license both available.

Cookbooks are expected to include three parts: Getting Started Guide, Improved Tutorial, and Advanced Tutorial.

This tutorial is the first part: Getting Started Guide.

Go to <http://www.linb.net> for the latest information.

If you have any good suggestions, you can contact me at [linb.net \[at\] gmail.com](mailto:linb.net[at]gmail.com).

# Chapter 1. Preparation

First of all, note that all instances of this tutorial are based on version 4.0. Therefore, our first task is to download the 4.0 release package, and to establish the local environment.

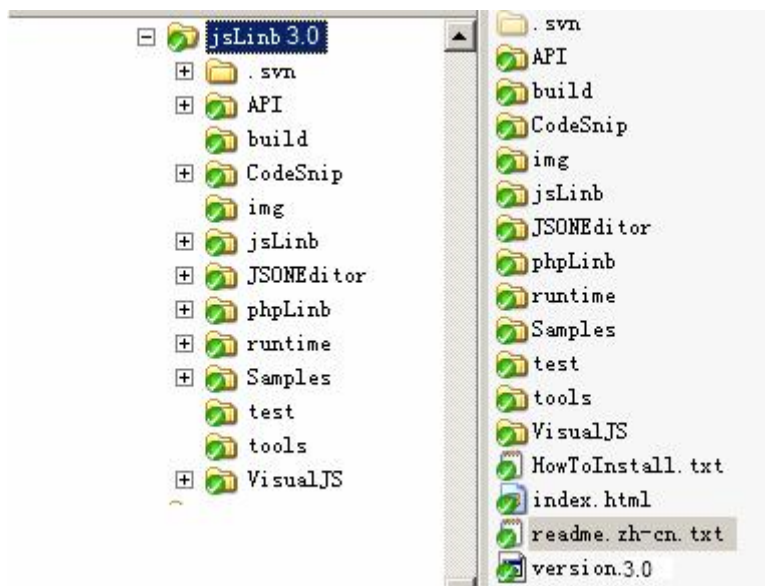
## 1.1. Download the package

4.0 Release package can be downloaded from <http://code.google.com/p/linb/downloads/list>. It's the latest stable 4.0 version, but not the latest code. I suggest you get the latest 4.0 code from SVN. For those who are not familiar with SVN, should learn how to use SVN first. After all, a lot of open-source projects use SVN to manage code. SVN requires a client program to connect to what is called a "repository" where the files are stored. On commonly used SVN client is called TortoiseSVN, which is freely available. Other clients exist, but TortoiseSVN is recommended due to its simplicity of use.

4.0 version "repository" URL: <http://linb.googlecode.com/svn/trunk/jsLinb4.0/>.

## 1.2. The package folder

If you downloaded package from google group, extract the package to a local folder. If you fetch the code from SVN, does not need to extract.



*The contents of the package folder*

By default, most of the examples in the package can be run in local disk directly, but a small number of examples need php background environment, or mysql database. In this case, you need to prepare



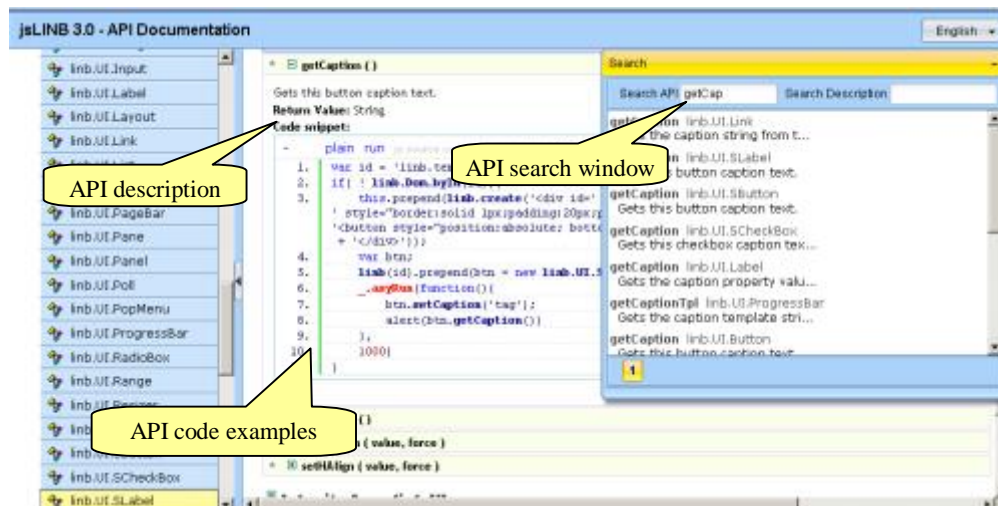
Apache server (version 2 and above), php (version 5 and above) and mysql (version 5 and above). At last, copy the package to apache web directory.

## 1.3. Glance at examples and API

If your Apache/php environment works well, after you copied the package folder to Apache's web directory (this tutorial assumes that your root directory is <http://localhost/jsLinb/>), you should be able to open the page with your browser: <http://localhost/jsLinb/>.



You can browse <http://localhost/jsLinb/Examples/> for examples, and <http://localhost/jsLinb/API/> for API.



A simple glance at API is strongly recommended. Learn about how to search a specific API, and how to run the inner code examples.

## Chapter 2. Hello World

### 2.1. The first application

As many would expect or not expect, the first example is "Hello World".

Now, create a new folder "mycases" in the package folder (again, (this tutorial assumes that your root directory is <http://localhost/jsLinb/>), add a sub folder "chapter1" in it, and create a file named "helloworld.html" in "chapter1". Enter the following code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta http-equiv="imagetoolbar" content="no" />
  <script type="text/javascript" src="../../runtime/jsLinb/js/linb-all.js"></script>
  <title>jsLinb Case</title>
</head>
<body>
  <script type="text/javascript">
    linb.main(function(){
      linb.alert("Hi", "Hello World");
    });
  </script>
</body>
</html>
```

strict mode is recommended

main function

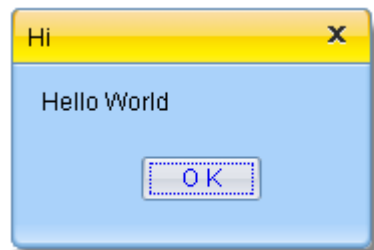
Include lib file

<cookbook/chapter1/helloworld.html>

There is a **cookbook** folder in the package folder. You can find the source code for each example in this tutorial.

You can double-click the helloworld.html to open the file.

Or open URL <http://localhost/jsLinb/cookbook/chapter1/helloworld.html> in your browser (firefox or chrome is recommended here). And you can see the following result:



You may have noticed, no lib CSS file was included in the html file. Yes, jsLinb generate CSS automatically, no lib CSS.

File "linb-all.js" contains all codes except several advanced controls (Button, Input, CombInput,

Tabs, TreeBar, and TreeGrid etc.). This file can be found in “runtime/js” folder.



## 2.2.Render onto a html node

“Replace an html node (such as a div) with an advanced control.” A project manager said. "Our web page engineer is responsible to design an html file including a DIV with a unique ID, and JavaScript engineer is responsible to build an advanced UI control, and replace that DIV."

The following example in file chapter1/renderonto.html:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta http-equiv="imagetoolbar" content="no" />
  <script type="text/javascript" src="../../runtime/jsLinb/js/linb-debug.js"></script>
  <title>jsLinb Case</title>
</head>
<body>
  <div id="grid" style="position:absolute;left:100px;top:100px;width:300px;height:200px;"></div>
  <script type="text/javascript">
    linb.main(function(){
      var grid = new linb.UI.TreeGrid();
      grid.setGridHandlerCaption('grid')
      .setRowNumbered(true)
      .setHeader(['col 1','col 2','col 3'])
      .setRows([
        ['a1','a2','a3'],
        ['b1','b2','b3'],
        ['c1','c2','c3'],
        ['d1','d2','d3'],
        ['e1','e2','e3'],
        ['f1','f2','f3']
      ]);
      grid.renderOnto('grid');
    });
  </script>
</body>
</html>
```

The DIV with id “grid”

Sets grid caption

Show line number

Sets columns

Sets rows

Render onto that DIV

[cookbook/chapter1/renderonto.html](#)

The result is:

grid	col 1	col 2	col 3
1	a1	a2	a3
2	b1	b2	b3
3	c1	c2	c3
4	d1	d2	d3
5	e1	e2	e3
6	f1	f2	f3

There are two ways to get the same result; codes were in renderonto2.html and renderonto3.html.

renderonto2.html :

```
linb.main(function(){
  (new linb.UI.TreeGrid({
    gridHandlerCaption:'grid',
    rowNumbered:true,
    header:['col 1','col 2','col 3'],
    rows:[['a1','a2','a3'],['b1','b2','b3'],['c1','c2','c3'],
           ['d1','d2','d3'],['e1','e2','e3'],['f1','f2','f3']]
  })).renderOnto('grid');
});
```

Gives properties directly

[cookbook/chapter1/renderonto2.html](#)

renderonto3.html :

```
linb.main(function(){
  linb.create('TreeGrid',{
    gridHandlerCaption:'grid',
    rowNumbered:true,
    header:['col 1','col 2','col 3'],
    rows:[['a1','a2','a3'],['b1','b2','b3'],['c1','c2','c3'],
           ['d1','d2','d3'],['e1','e2','e3'],['f1','f2','f3']]
  }).renderOnto('grid');
});
```

Using linb.create

[cookbook/chapter1/renderonto3.html](#)

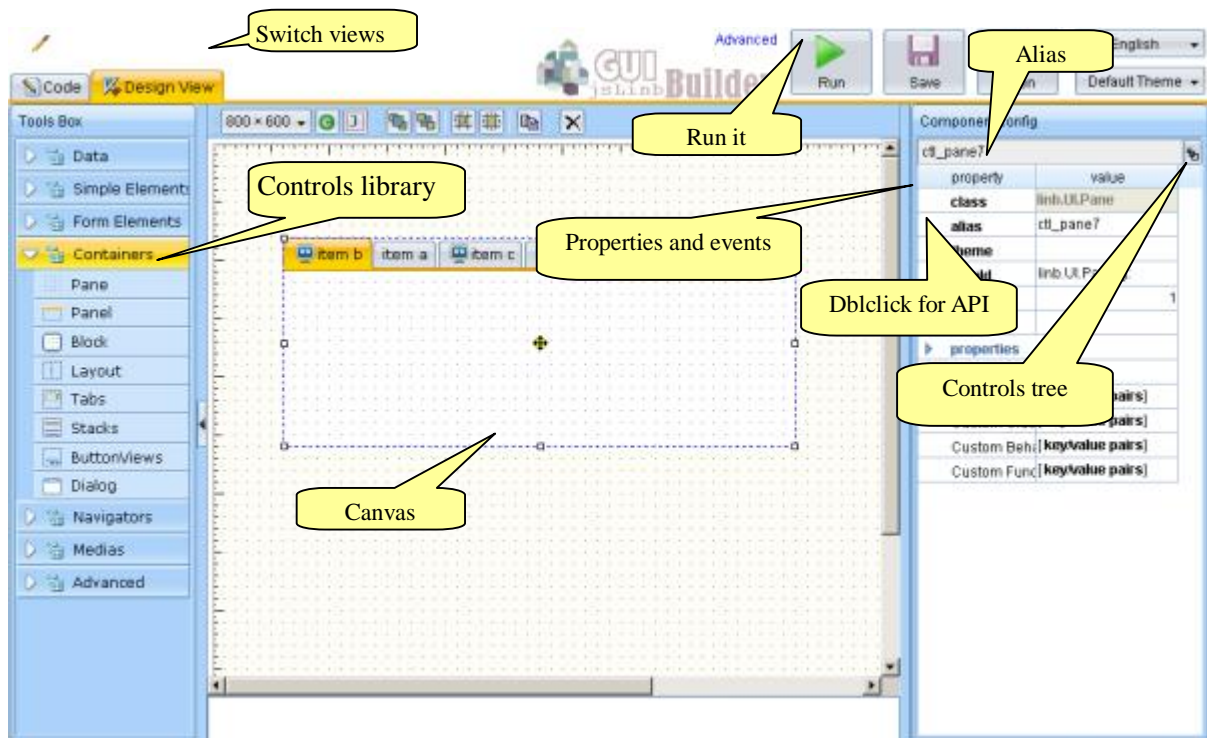
These three approaches generated the same result. You can use any of those in your project according to your habits. But the first approach (using new and setXX) is recommended.

## 2.3. Do it in Designer

There are two types of designer in jsLinb: One is a simplified version, the other one is an advanced version integrated with document management features. The purposes of the two designers are to reduce development time on UI layout. We only use the simplified version in this introduction

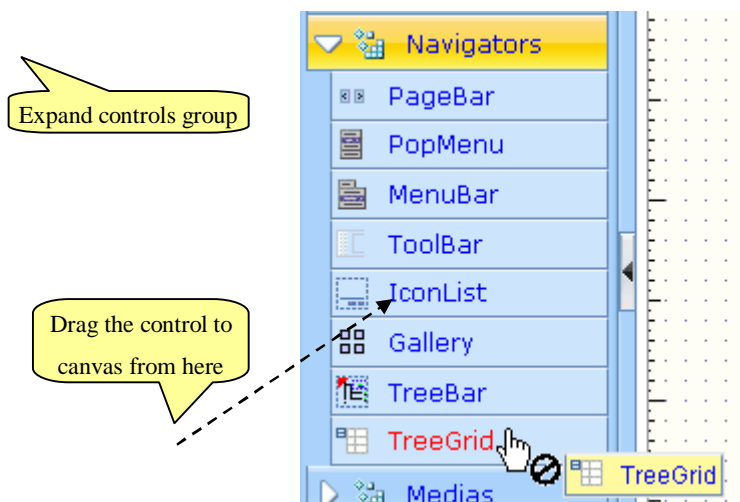
tutorial.

Go to <http://localhost/jsLinb/VisualJS/UIBuilder.html> for the simplified one.



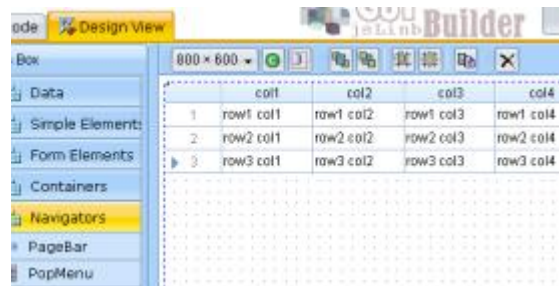
Now, we are trying to create the previous section's grid example in Designer.

1. Open the navigators group in "Tools Box", and drag the "TreeGrid" control to the Canvas area.



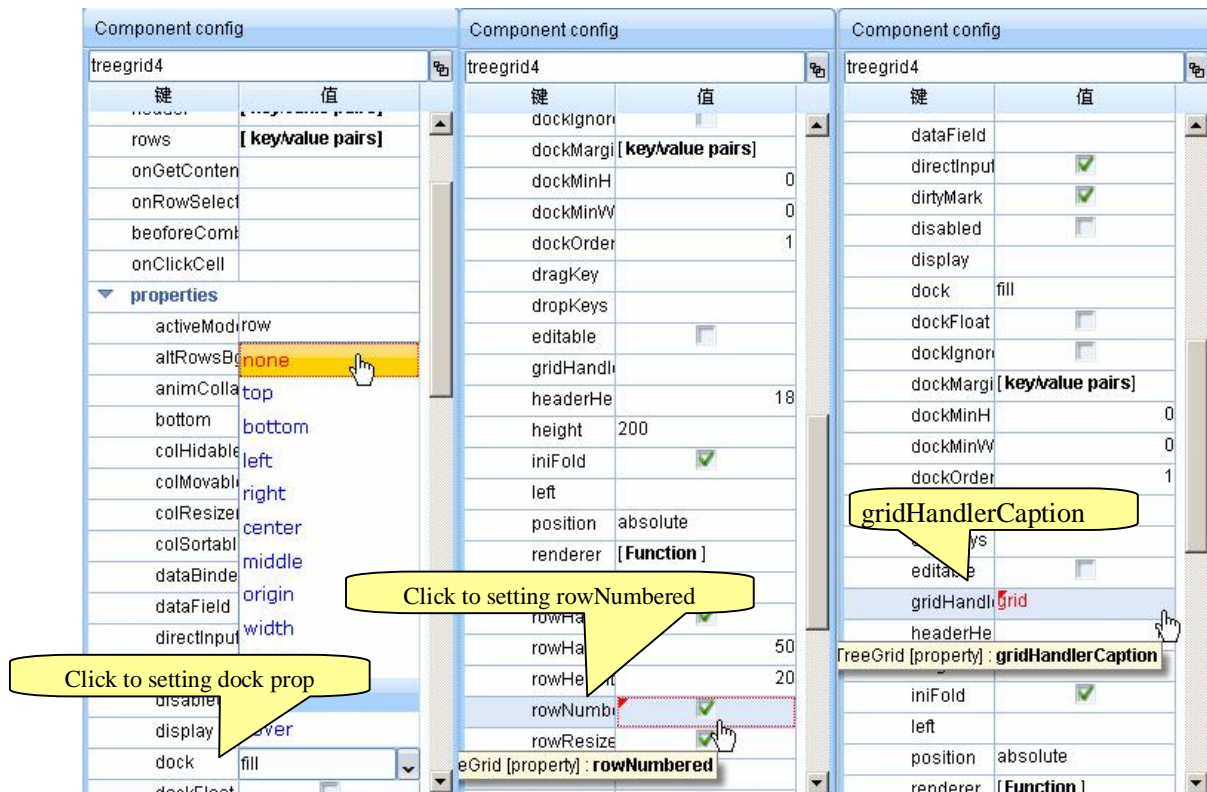
2. Click to select the "treegrid"



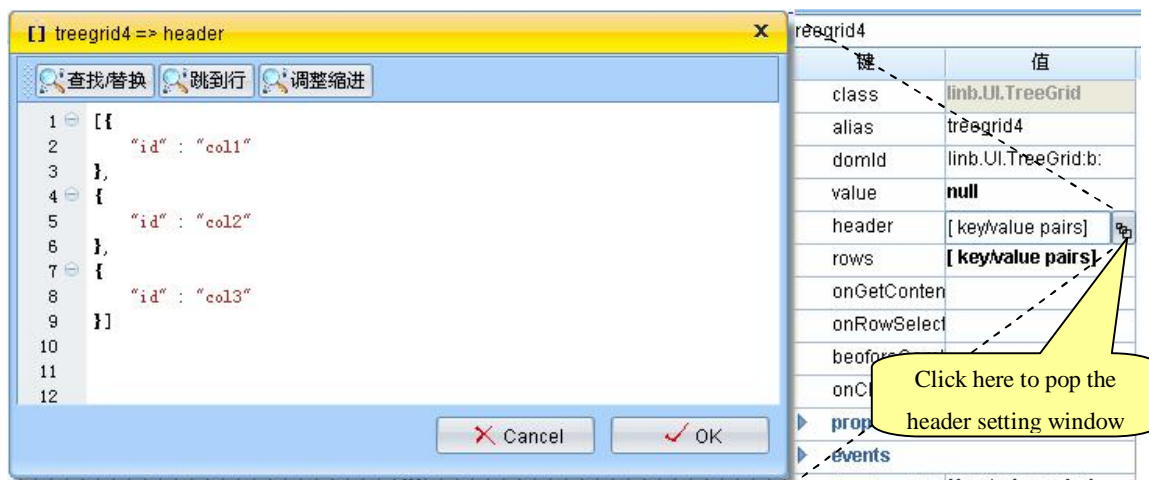


3. Sets this grid's properties according to the following picture.

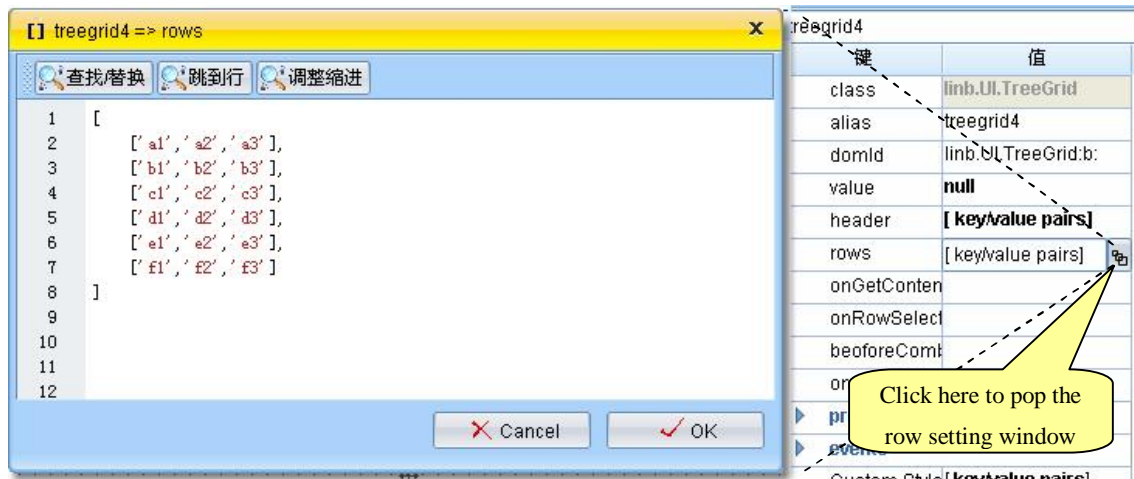
- ┆ Sets dock to 'none';
- ┆ Sets rowNumbered to false;
- ┆ Sets gridHandlerCaption to 'grid'.



4. Sets header and rows

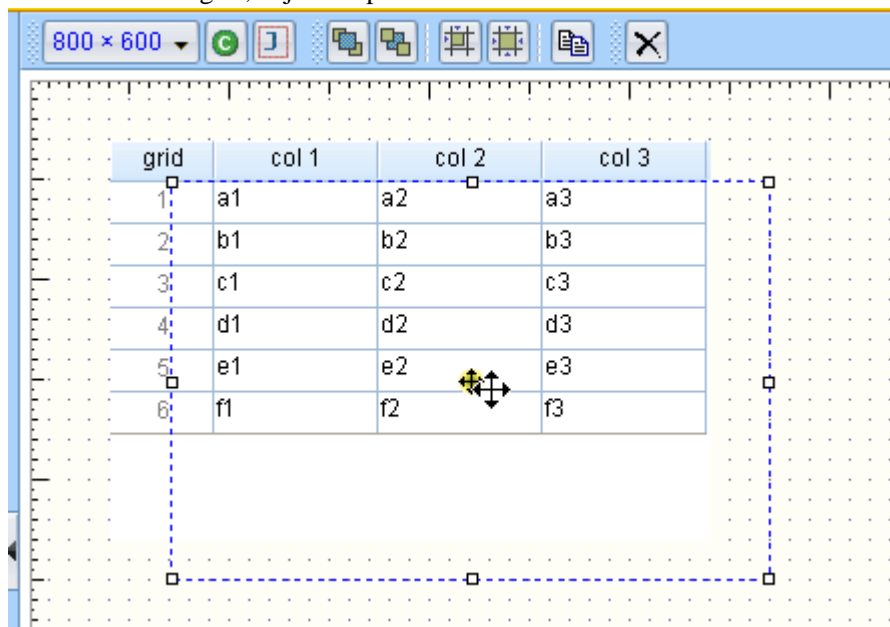


## Sets header data

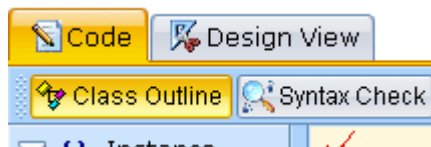


## Sets rows data

- Click to select the grid, adjust its position and size



- Now, switch to "Code" view



```

1 // The default code is a com class (inherited from linb.Com)
2 Class('{className}', 'linb.Com', {
3     // Ensure that all the value of "key/value pair" does not refer to external variables
4     Instance: {
5         // To initialize instance (e.g. properties)
6         initialize : function() {
7             // To determine whether or not the com will be destroyed, when the first UI control be destroyed
8             this.autoDestroy = true;
9             // To initialize properties
10            this.properties = {};
11        },
12        // To initialize internal components (mostly UI components)
13        // *** If you're not a skilled, dont modify this function manually ***
14        iniComponents : function() {
15            // [[code created by jsLinb UI Builder
16            var host=this, children=[], append=function(child){children.push(child.get(0))};
17
18            append(
19                (new linb.UI.TreeGrid)
20                .setHost(host, "ctl_treegrid2")
21                .setDock("none")
22                .setLeft(0)
23                .setTop(0)
24                .setGridHandlerCaption("grid")
25                .setHeader([{"id": "col1", "width": 80, "type": "label", "caption": "col1"}, {"id": "col2", "width":
26                .setRows([{"cells": [{"value": "row1 col1", "id": "c_a"}, {"value": "row1 col2", "id": "c_b"}, {"v
27
28

```

Code created by jsLinb UI Builder

Above code is serialized by Designer. Header **data** and rows data will not look the same as your setting.

- Click "Run" Button to open the test window, you will see the same result with section 2.2.



- Copy the code from this test page, and paste to a new file designer.grid.html.



```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta http-equiv="imagetoolbar" content="no" />
  <title>Web application powered by LINB framework</title>
</head>
<body>
  <div id="loading"></div>
  </div>
  <script type="text/javascript" src="../../runtime/jsLinb/js/linb-all.js"></script>
  <script type="text/javascript">
    Class('App', 'linb.Com',{
      Instance:{
        initComponents:function(){
          // [[code created by jsLinb UI Builder
          var host=this, children=[], append=function(child){children.push(child.get(0));

          append((new linb.UI.TreeGrid)
            .host(host,"treegrid4")
            .setDock("none")
            .setLeft(60)
            .setTop(50)
            .setRowNumbered(true)
            .setGridHandlerCaption("grid")
            .setHeader([{"id":"col 1", "width":80, "type":"label", "caption":"col 1"}, {"id":"col 2",
"width":80, "type":"label", "caption":"col 2"}, {"id":"col 3", "width":80, "type":"label", "caption":"col 3"}])
            .setRows([{"cells":[{"value":"a1"}, {"value":"a2"}, {"value":"a3"}], "id":"j"},
{"cells":[{"value":"b1"}, {"value":"b2"}, {"value":"b3"}], "id":"k"}, {"cells":[{"value":"c1"}, {"value":"c2"},
{"value":"c3"}], "id":"l"}, {"cells":[{"value":"d1"}, {"value":"d2"}, {"value":"d3"}], "id":"m"}, {"cells":[{"value":"e1"},
{"value":"e2"}, {"value":"e3"}], "id":"n"}, {"cells":[{"value":"f1"}, {"value":"f2"}, {"value":"f3"}], "id":"o"}])
          );

          return children;
          // ]]code created by jsLinb UI Builder
        }
      }
    });
    linb.Com.load('App', function(){
      linb('loading').remove();
    });
  </script>
</body>
</html>

```

Showing a loading picture

Places js lib files in body

Class created in Designer.  
You can save this part of code to **App/js/index.js**

Load UI in asynchronous mode  
If no App Class in memory, lib will load the Class from **App/js/index.js** file first.

[cookbook/chapter1/designer.grid.html](#)

## 2.4. Application loading process

In section 2.3, we put all html and JavaScript code in a single file. For a bigger application, it's not a wise solution. A real application may include dozens of classes. For a developer, maintaining each class in a separate file is always a must.

OK. Let's separate "designer.grid.html" into two files **a** designer.grid.standard.html, and **App/js/index.js**.

designer.grid.standard.html is:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/CSS" />
  <meta http-equiv="imagetoolbar" content="no" />
  <title>Web application powered by LINB framework</title>
</head>
<body>
  <div id="loading"></div>
  <script type="text/javascript" src="../../runtime/jsLinb/js/linb-all.js"></script>
  <script type="text/javascript">
    linb.Com.load('App', function(){
      linb('loading').remove();
    });
  </script>
</body>
</html>
```

Load App class from App/js/index.js asynchronously

At last, remove loading picture

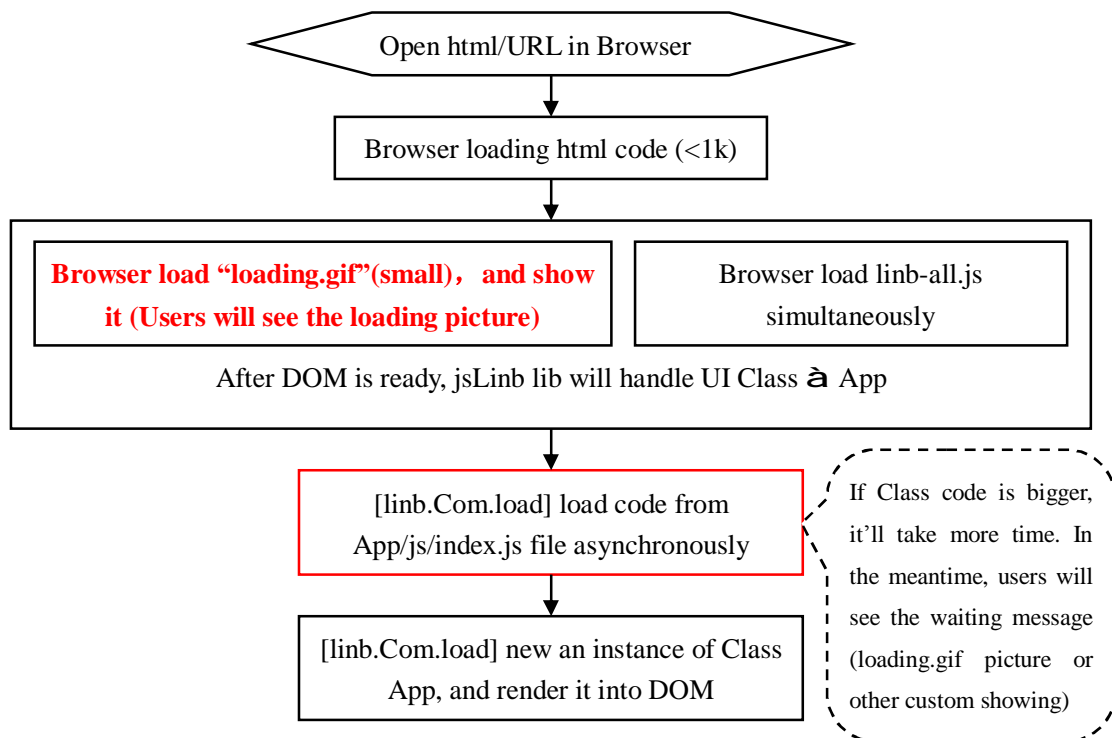
[cookbook/chapter1/designer.grid.standard.html](#)

App/js/index.js is:

```
Class('App', 'linb.Com',{
  Instance:{
    iniComponents:function(){
      // [[code created by jsLinb UI Builder
      var host=this, children=[], append=function(child){children.push(child.get(0))};
      append((new linb.UI.TreeGrid)
        .host(host, "treegrid4")
        .setDock("none")
        .setLeft(60)
        .setTop(50)
        .setRowNumbered(true)
        .setGridHandlerCaption("grid")
        .setHeader([{"id":"col 1", "width":80, "type":"label", "caption":"col 1"}, {"id":"col 2", "width":80,
"type":"label", "caption":"col 2"}, {"id":"col 3", "width":80, "type":"label", "caption":"col 3"}])
        .setRows([{"cells":[{"value":"a1"}, {"value":"a2"}, {"value":"a3"}], "id":"j"}, {"cells":[{"value":"b1"},
{"value":"b2"}, {"value":"b3"}], "id":"k"}, {"cells":[{"value":"c1"}, {"value":"c2"}, {"value":"c3"}], "id":"l"},
{"cells":[{"value":"d1"}, {"value":"d2"}, {"value":"d3"}], "id":"m"}, {"cells":[{"value":"e1"}, {"value":"e2"},
{"value":"e3"}], "id":"n"}, {"cells":[{"value":"f1"}, {"value":"f2"}, {"value":"f3"}], "id":"o"}])
      );
      return children;
      // ]]code created by jsLinb UI Builder
    }
  }
});
```

[cookbook/chapter1/App/js/index.js](#)

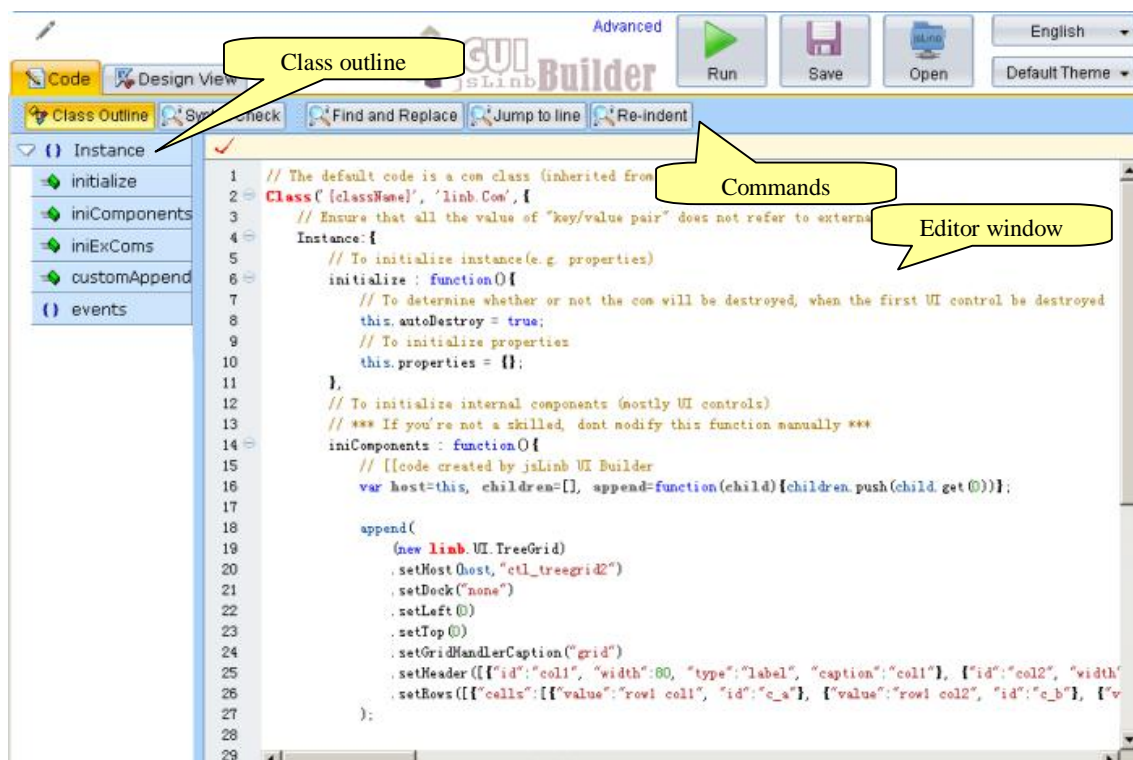
When we open **designer.grid.standard.html** in Browser, the loading process will be:



## 2.5. Code Editor

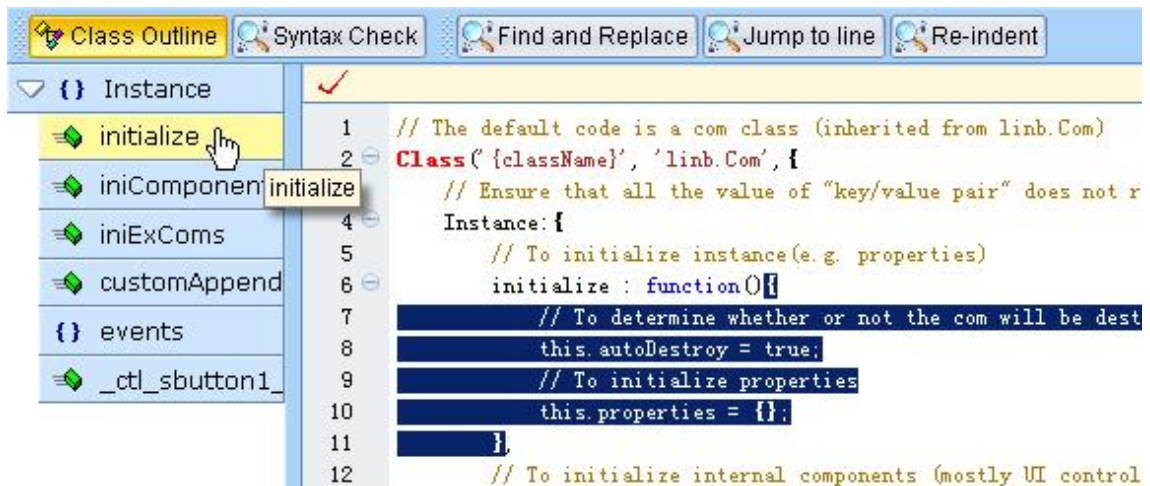
By the way, in order to get better performance, firefox and chrome are recommended here.

There are two views in Builder: “Design view” and “Code” view. The default view is “Design view”. Click “Code” tab to switch to “Code” view.



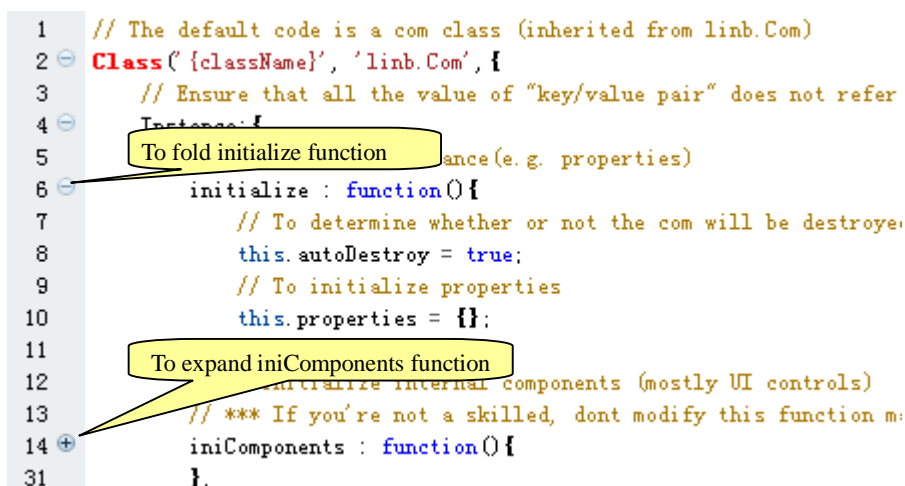
## 2.5.1. Highlight code from Outline window

“Class Outline” is located in the left side of “Code” view. By clicking any member or method name in “Class Outline”, Builder will highlight its code in “Editor window”, and scroll “Editor window” to show the code.



## 2.5.2. Code Folding

To make your code view more clear to read and understand, jsLinb Builder lets you fold certain parts of it. Click the left side “plus” or “minus” will fold or expand the block code.



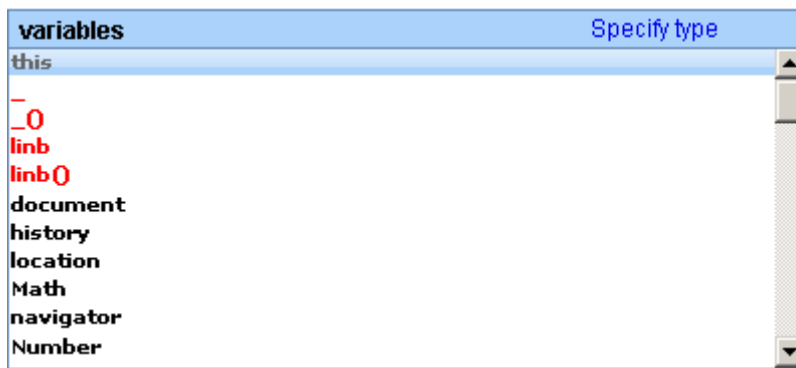
Note: Due to the browser’s poor performance on iframe, please try not to frequent collapse or expand the large body function or object. Chrome’s performance is better than others.

## 2.5.3. Code Intellisense

Three types Code Intellisense are supported.

- I When context does not recognize the input string;
- I Type dot after a recognizable variable

- When use shortcut [Alt+1 ], or dbclick a recognizable variable

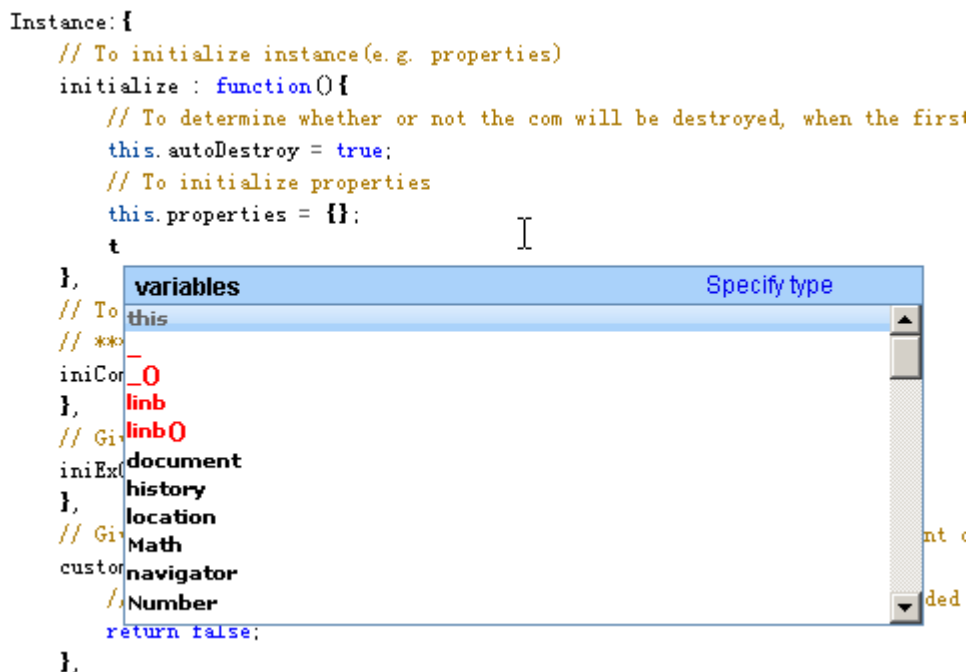


Keyboard actions for Code Intellisense pop Window:

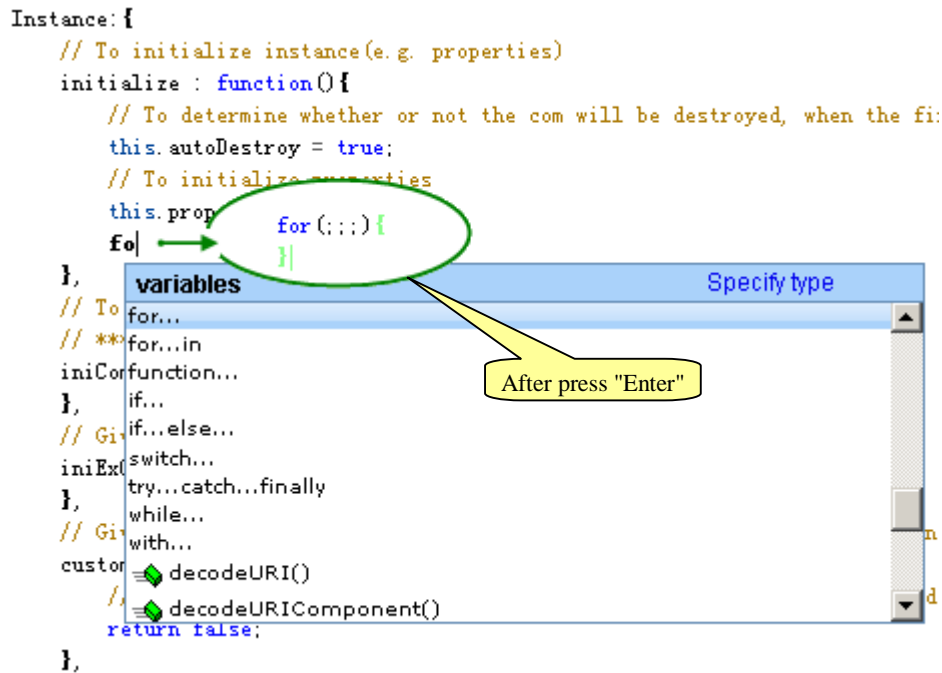
- “up”: Focus to next item in code list
- “down”: Focus to previous item in code list
- “enter”: Select the current focused item, and input to editor window
- “esc”: Close the pop window
- Other visible chars: Find and focus the first matched item

### 2.5.3.1. When context doesn't recognize the input string

When you input a string, if editor doesn't recognize this string, it will pop a list window including local variables, global variables, global functions and JavaScript reserved keywords. In the below picture, type 't' will trigger editor to pop a list window, “this” is the default focused item.



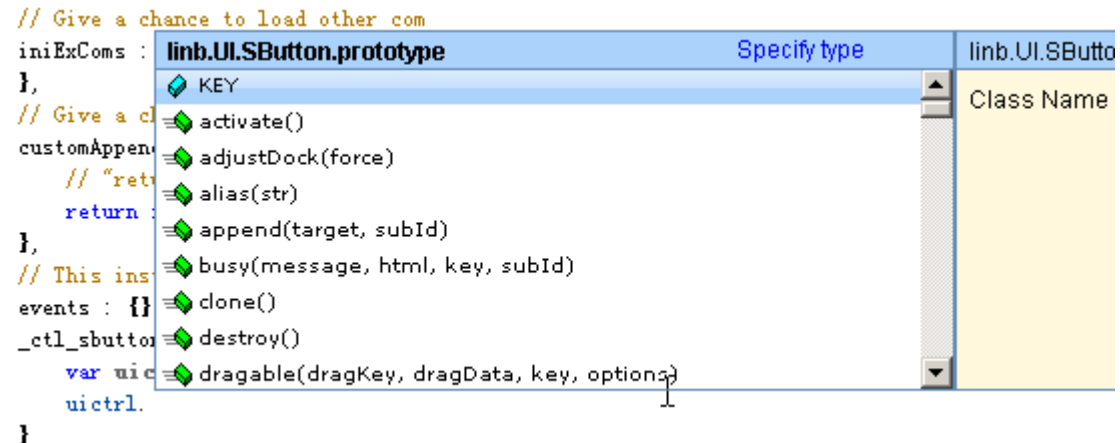
If the input string is “fo”, the “for loop statement” will be the default focused item.



In this case, "Enter" keypress will cause "for loop statement" code to be inserted into the editor automatically.

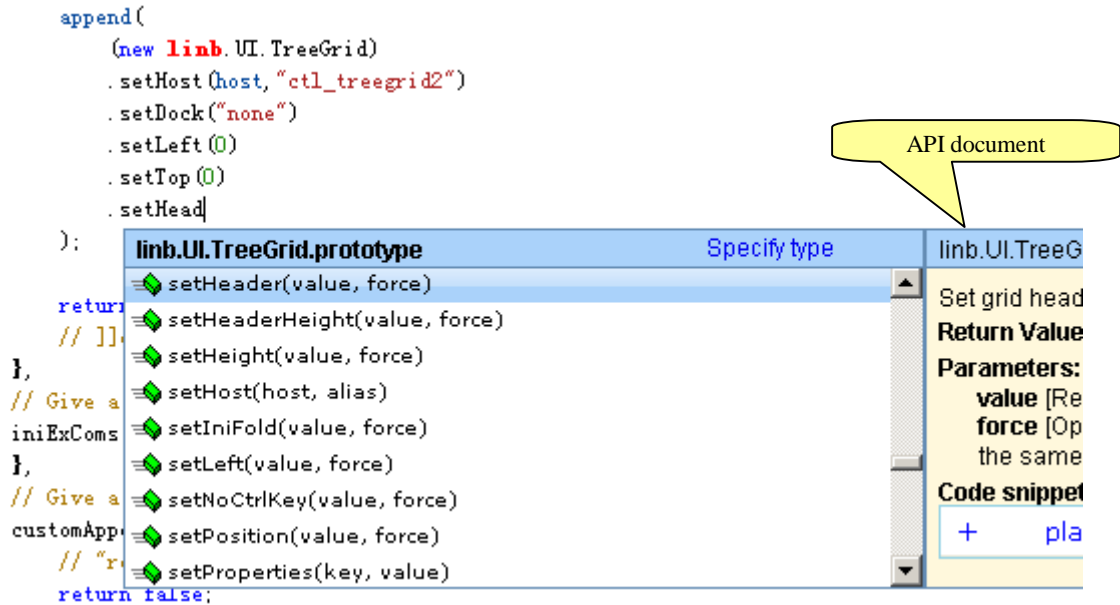
### 2.5.3.2. Type dot after a recognizable variable

After an editor recognizable variable, if you type char ".", editor will pop an available members and functions list for the variable.



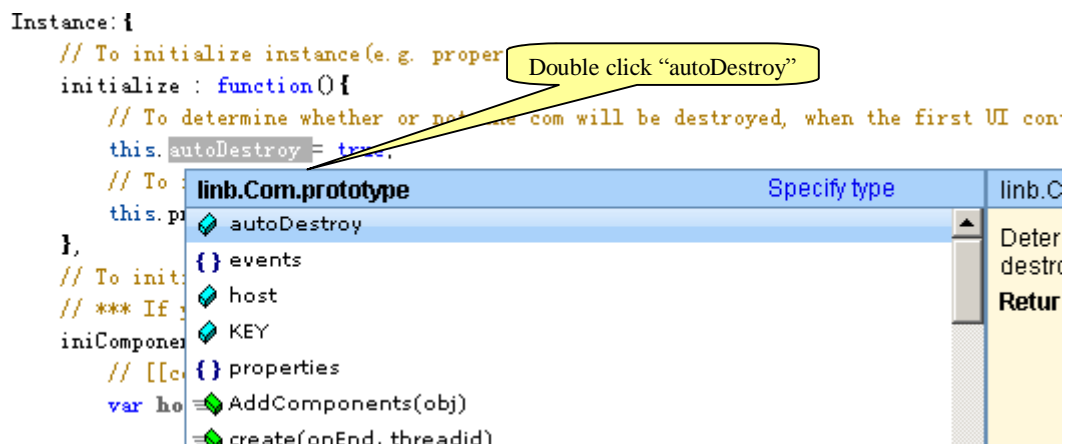
Chainable methods can show Code Intellisense window too.

Chainable methods



### 2.5.3.3. When use shortcut [Alt+1 ], or dbclick

When a variable was focused, press shortcut [Alt+1] will trigger editor to pop the Code Intellisense window. Double click this variable string will get the same result.

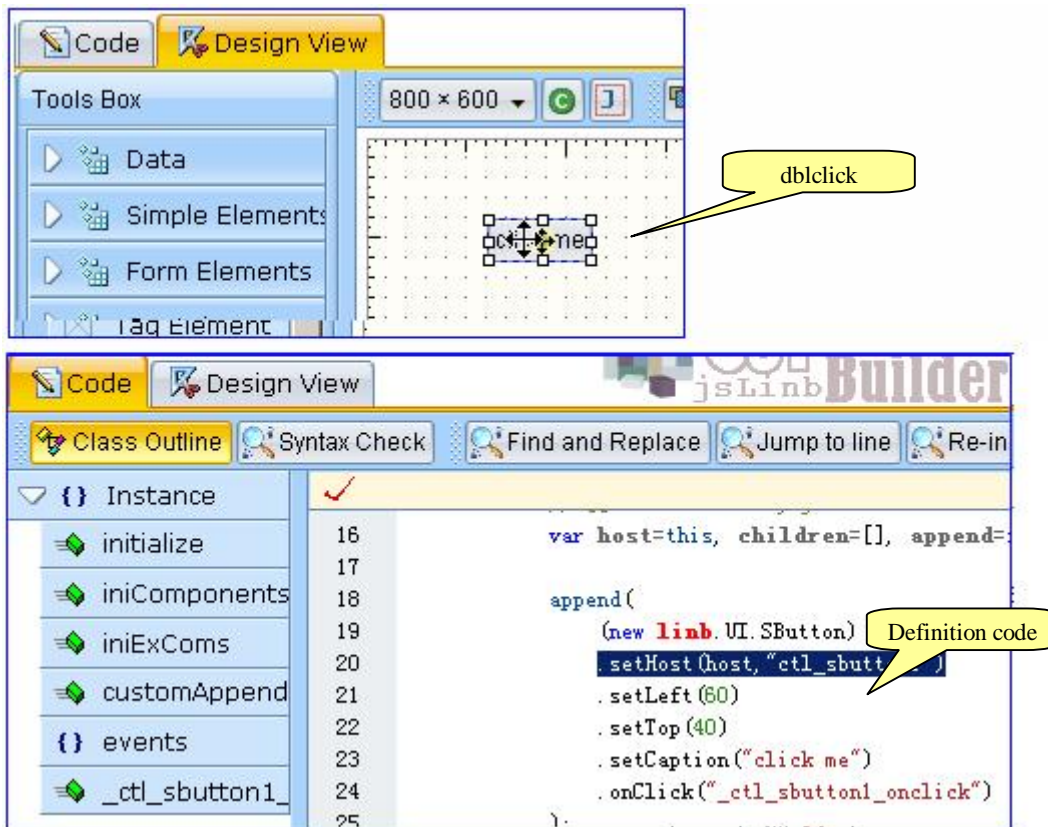


### 2.5.4. Find the object definition code

In “Design View”, double click a control will cause:

- 1) Switch to “Code” view;
- 2) Highlight the control’s definition code;
- 3) Scroll the definition code to view.



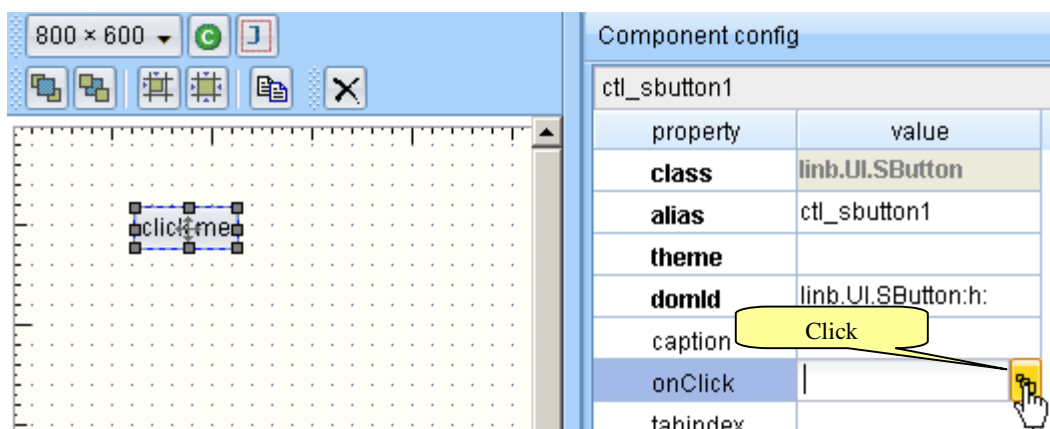


### 2.5.5. Generate event code automatically

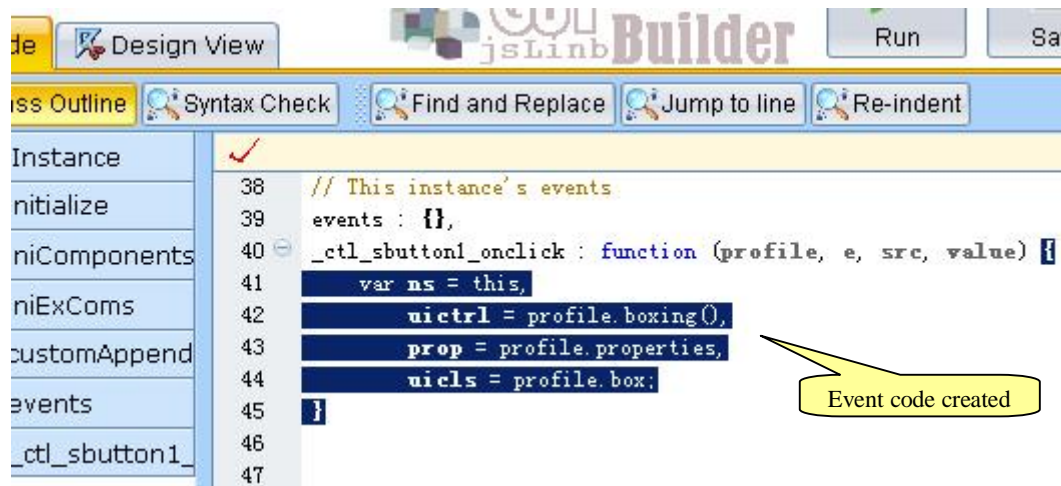
In the “Design View”, select a control; the right side “Component config” window will be refreshed.

Find an event (e.g. onClick event), click its event button will cause:

- 1) Switch to “Code” view;
- 2) Create event code, and insert into the editor;
- 3) Scroll the event code to view.







## Chapter 3. Controls Facebook

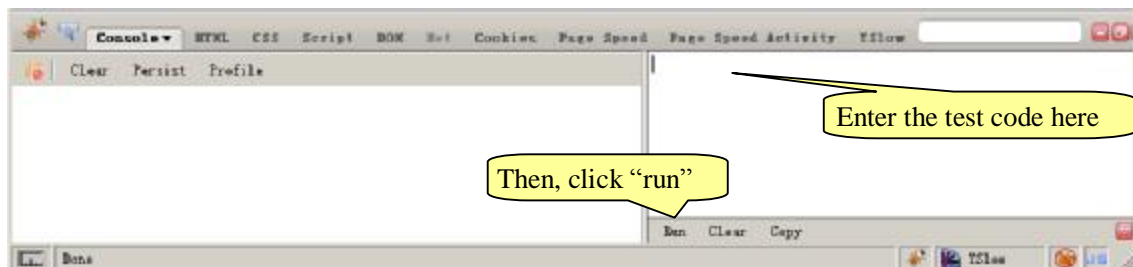
Many beginners are particularly interested in UI controls. In this chapter we'll give a rough look at the basic controls. Since each control has a lot of functions, here is a brief introduction, it is impossible to explain all the functions. You can browse API to understand the specific function of each control in detail!

### 3.1. Script testing environment

At first, we have to build a testing environment for executing example codes. About Browsers, firefox is recommended, if firefox is not preferred, ie8 or chrome is ok too.

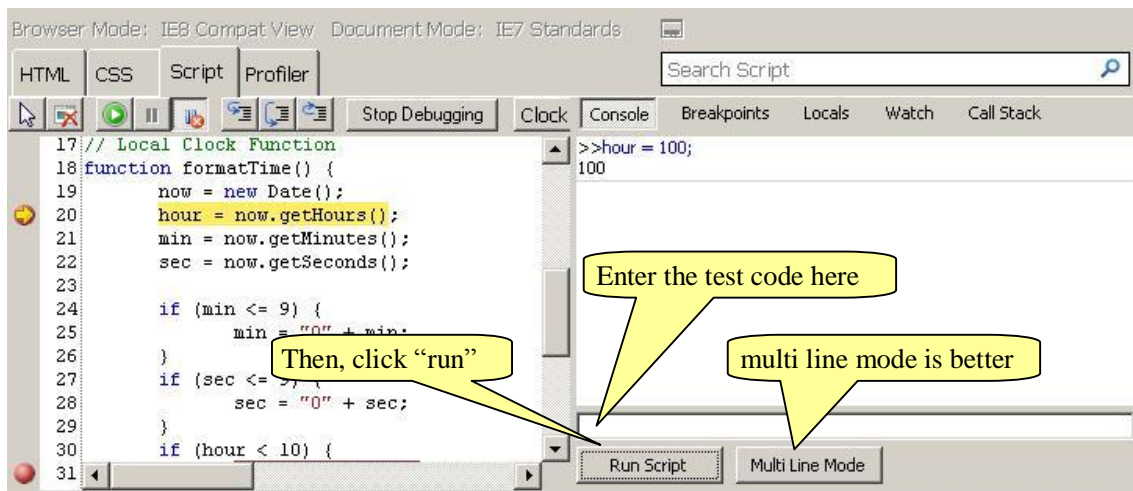
#### For firefox:

1. You need firefox and firebug;
2. Ensure the file “[env.html](#)” and all cookbook folders in [cookbook](#) dir;
3. Open URL [cookbook/env.html](#) in firefox;
4. Open firebug console, switch to the multi-line mode



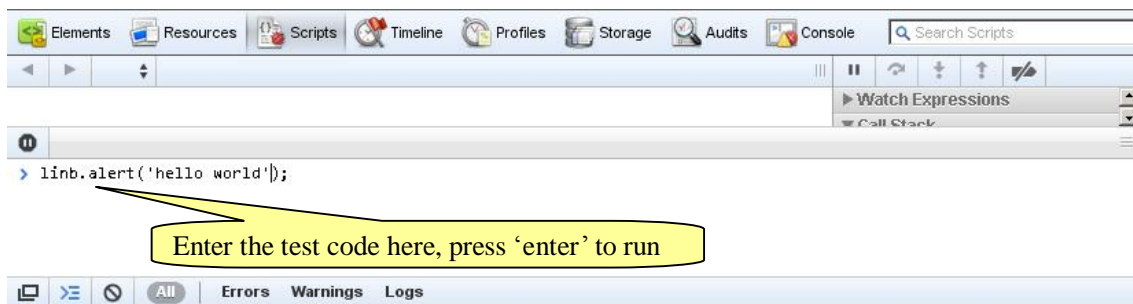
#### For IE8:

1. You need IE8;
2. Open URL [cookbook/env.html](#) ;
3. Open developer tools, switch to the multi-line mode

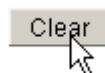


### For Chrome:

1. You need the latest Chrome;
2. Open URL [cookbook/env.html](#) ;
3. Open developer tools



There's a "Clear" button in [cookbook/env.html](#), You can click this button to clean up the current page's DOM. In some cases, you want to clean up both DOM and memory, press 'F5' to refresh your browser.



## 3.2. "Hello world" in env.html

Input the following code into script window, and run it.

```
linb.alert("Hi", "Hello World!");
```

**Output:**



Click "Clear" button to clean the DOM.

### 3.3. Control creation and runtime update

There are three approaches to create jsLinb control.

```
// Approach 1
linb.create("SButton", {
  caption: "Using linb.create function",
  position: "relative"
}).show();

// Approach 2
(new linb.UI.SButton({
  caption: "Using new and key/value pairs",
  position: "relative"
})).show();

// Approach 3
(new linb.UI.SButton())
.setCaption("Using new and get/set")
.setPosition("relative")
.show();
```

We use new/setXX mode in Desinger

The above three approaches will create entirely consistent UI.

You can use setXXX function to update the control after it was rendered into DOM (runtime update).

```
var dlg=linb.create("Dialog", {caption: "runtime "}).show();
_.asRun(function(){
  dlg.setCaption("updated");
},500);
_.asRun(function(){
  dlg.setMaxBtn(false);
},1000);
_.asRun(function(){
  dlg.setStatus("max");
},1500);
_.asRun(function(){
  dlg.destroy();
},2000);
```

Create a Dialog

To modify caption

To hide the max button

To modify status

To destroy it

## 3.4. Button related

This section relates to the following controls: `linb.UI.Link`, `linb.UI.SButton`, `linb.UI.Button`, `linb.UISCheckBox` and `linb.UI.CheckBox`.

### 3.4.1. onClick event

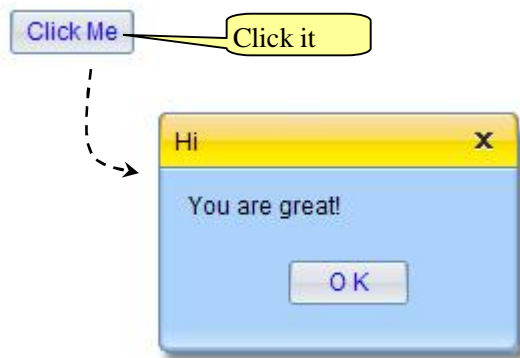
**Input:**

```
var btn=new linb.UI.SButton();
btn.setCaption("Click Me")
.onClick(function(){
    linb.alert("Hi","You are great!");
});
btn.show();
```

Sets caption

Adds onClick event

**Output:**



**Input:**

```
var btn=new linb.UI.Button();
btn.setCaption("Click Me")
.onClick(function(){
    linb.alert("Hi","You are great!");
});
btn.show();

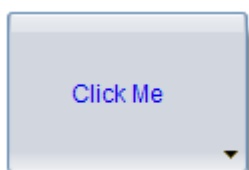
_.asynRun(function(){
    btn.setHeight(80)
    .setShadow(true)
    .setType("drop")
},1000);
```

Sets caption

Adds onClick event

Execute code after 1 second

**Output:**



**NOTE**

**linb.UI.SButton / SLabel / SCheckbox** are enough for most cases; Only if you need more complex feature, you should use those complex control: **linb.UI.Button / Label / Checkbox**.

## 3.4.2. Boolean Controls

There are three controls can represent and modify Boolean value:

**Input:**

```
var btn= (new linb.UI.Button({position: "relative", caption:"Button", type:"status"})).show();
var scb= (new linb.UI.SCheckBox({position: "relative", caption:" SCheckBox"})).show();
var cb= (new linb.UI.CheckBox({position: "relative", caption:" CheckBox"})).show();

_._asRun(function(){
    btn.setValue(true,true);
    scb.setValue(true,true);
    cb.setValue(true,true);
},1000);
```

Sets position to 'relative'

Sets values to true after 1 second

**Output:**

Button ☒ SCheckBox ☒ CheckBox

## 3.4.3. Link Control

You can take linb.UI.Link as a simple button.

**Input:**

```
var btn=new linb.UI.Link();
btn.setCaption("Click Me");
.onClick(function(){
    linb.alert("Hi","You are great!");
});
btn.show();

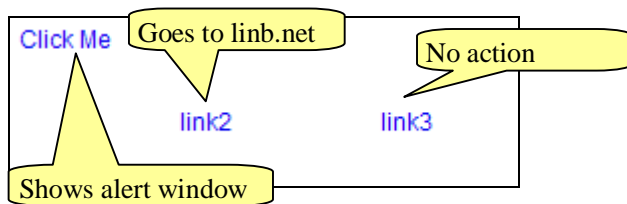
// href property
linb.create("Link",{href: "http://www.linb.net"}).show(null,null,100,100)

// href was disabled when return false
linb.create("Link",{href: "http://www.longboo.com"}).show(null,null,200,100)
.onClick(function(){
    return false;
});
```

Sets caption

Adds onClick event

**Output:**



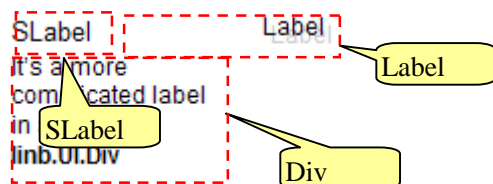
## 3.5. Label related

This section relates to the following controls: `linb.UI.SLabel`, `linb.UI.Label` and `linb.UI.Div`. These three controls can be used as “label”, `linb.UI.SLabel` is the simplest one, but it’s enough for most cases; If you need more complex feature like shadow, resizer or border, you should choose `linb.UI.Label`; Or if you want to input more complex html code in the control, `linb.UI.Div` is better.

**Input:**

```
(new linb.UI.SLabel()).setCaption("SLabel").setPosition("relative").show();
(new linb.UI.Label()).setCaption("Label").setPosition("relative").setShadowText(true).show();
(new linb.UI.Div()).setHtml("It's a more complicated label in a <br /><b>linb.UI.Div</b>")
.setPosition("relative").show();
```

**Output:**



## 3.6. Input related

This section relates to the following controls: `linb.UI.Input`, `linb.UI.ComboInput` and `linb.UI.RichEditor`. `linb.UI.ComboInput` is an enhanced version of `linb.UI.Input`, it can input/edit value through a pop window; `linb.UI.RichEditor` is a rich text input/edit control.

### 3.6.1. setValue/setValue/getUIValue/setUIValue

From the users point of view, value controls (all derived from the `linb.absValue` control) in `jsLinb` has two values: the “UI value”(`getUIValue/setUIValue`) and the “control value”(`getValue/setValue`).

“UI value” dose not always equal to “control value”. For example, for an empty input control

1. Keyboard input “**abc**”: “UI value” is “**abc**”, “control value” is **empty**;
2. Calls “updateValue” function: “UI value” is “**abc**”, “control value” is “**abc**”;
3. Calls “setValue(**bcd**)”: “UI value” is “**bcd**”, “control value” is “**bcd**”;
4. Calls “setUIValue(**efg**)”: “UI value” is “**efg**”, “control value” is “**bcd**”

5. Calls “resetValue(‘x’): “UI value” is “x”, “control value” is “x”;

```
var input = (new linb.UI.Input()).show();
linb.message(input.getUIValue()+"."+input.getValue());
_.asynRun(function(){
  input.setUIValue('uivalue');
  linb.message(input.getUIValue()+"."+input.getValue());
},2000);
_.asynRun(function(){
  input.updateValue();
  linb.message(input.getUIValue()+"."+input.getValue());
},4000);
```

A new Input

Sets UI value

Updates UI value to control

### 3.6.2. Dirty Mark

If the control’s dirtyMark property is set to true, when “UI value” does not equal to “control value”, a “Dirty Mark” will appear. The “Dirty Mark” will disappear when “UI value” equals to “control value”.

Dirty Mark

uivalue

```
var input = (new linb.UI.Input()).show();
_.asynRun(function(){
  input.setUIValue('uivalue');
},1000);
_.asynRun(function(){
  input.updateValue();
  input.setDirtyMark (false);
},2000);
_.asynRun(function(){
  input.setUIValue('uivalue 2');
},3000);
```

Dirty Mark appears

Dirty Mark disappears

If DirtyMark is disabled

Nothing happen

### 3.6.3. Password Input

Sets Input’s type property to “password”.

**Input:**

```
var input = (new linb.UI.Input({type: 'password'})).show();
_.asynRun(function(){
  input.setUIValue('123456').updateValue();
  linb.pop(input.getValue());
},1000);
```

Sets type

**Output:**



### 3.6.4. Multi-lines

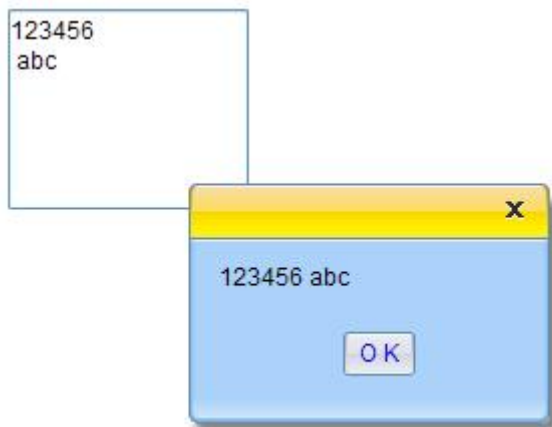
Sets Input's multiLine property to true.

**Input:**

```
var input = (new linb.UI.Input()).setMultiLines(true).setHeight(100).show();  
  
_.asynRun(function(){  
  input.setUITValue('123456 \n abc').updateValue();  
  linb.pop(input.getValue());  
},1000);
```

Sets multiLine to true

**Output:**



### 3.6.5. Input validation

#### 3.6.5.1. valueFormat property

“valueForamt” property represents a regular expression.

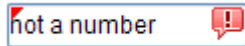
**Input:**



```
var input = (new linb.UI.Input())
    .setValueFormat("^-?\\d\\d*$")
    .show();
```

Number only

Executes the above code, input some **charts**, and let it lose the mouse focus, the “Error Mark” will appear.



### 3.6.5.2. beforeFormatCheck event

**Input:**

```
var input = (new linb.UI.Input())
    .beforeFormatCheck(function(profile,value){
        if(value!=parseFloat(value).toString())
            return false;
    })
    .show();
```

Number only

In above methods, “beforeFormatCheck” has priority. That means, when "beforeFormatCheck" returns ‘false’, "valueFormat" property will be ignored.

### 3.6.6. Dynamic input validation

In previous section examples, “Error Mark” appears only when the control loses focus. If you want to a real-time input validation , you need to set dynCheck property to true.

```
var input = (new linb.UI.Input())
    .setDynCheck(true)
    .setValueFormat("^-?\\d\\d*$")
    .show();
```

Sets dynCheck

### 3.6.7. Error Mark

#### 3.6.7.1. Default Error Mark

The default “Error Mark” is an icon at the right side of Input.



The default Error Icon

### 3.6.7.2. Validation Tips

There are three tool tips in linb.UI.Input control:

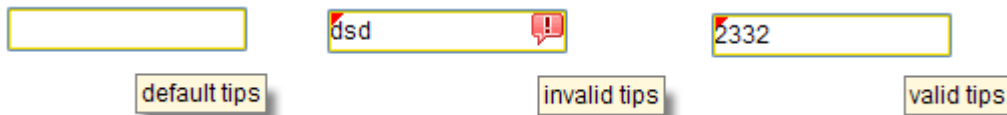
- ! tips: the default tool tips
- ! tipsOK: the valid tool tips
- ! tipsErr: the invalid tool tips

**Input:**

```
var input = (new linb.UI.Input())
    .setTips("default tips")
    .setTipsErr("invalid tips ")
    .setTipsOK("valid tips")
    .setValueFormat("^-?\\d\\d*$")
    .show();
```

Sets those tips

**Output:**



### 3.6.7.3. Binding Validation

You can bind the validation tips to a linb.UI.Div, linb.UI.SLabel or linb.UI.Span.

**Input:**

```
var slbl= (new linb.UI.SLabel({position:'relative'})).setCustomStyle({KEY:'padding-left:10px'});
var input = (new linb.UI.Input({position:'relative'}))
    .setValueFormat("^-?\\d\\d*$")
    .setTipsBinder(slbl)
    .setDynCheck(true)
    .setTips(" default tips")
    .setTipsErr(" invalid tips ")
    .setTipsOK(" valid tips")

input.show();
slbl.show();
```

Sets tipsBinder

Show SLabel here

**Output:**



### 3.6.7.4. Custom Error Mark

We can custom “Error Mark” in beforeFormatMark event.

**Input:**

```
var input = (new linb.UI.Input())
    .setValueFormat("^-?\\d\\d*$")
    .beforeFormatMark(function(profile,err){
        if(err)
            linb.alert("Invalid input!", "Only number allowed!", function(){
                profile.boxing().activate();
            });
        return false;
    }).show();
```

Customs information and  
aciton

Return false to ignore the default action

Output:



### 3.6.8. Mask Input

Mask Input examples:

11/11/1111	<input type="text" value="__/__/____"/>	(111) 111-1111	<input type="text" value="( ) _ _ - _ _"/>
~1.11	<input type="text" value="_. _"/>	(111) a-a *\$*	<input type="text" value="( ) _ _ _ \$ _"/>

In chapter2\Input\index.html

There is a mask property in linb.UI.Input control. It's a string. In this string,

- | '~' represents [+]
- | '1' represents [0-9]
- | 'a' represents [A-Za-z]
- | 'u' represents [A-Z]
- | 'l' represents [a-z]
- | '\*' represents [A-Za-z0-9]
- | Other visible char represents itself

Input:

```
var input = (new linb.UI.Input())
    .setMask("(1111)11111111-11")
    .show();
```

Output:

**NOTE**

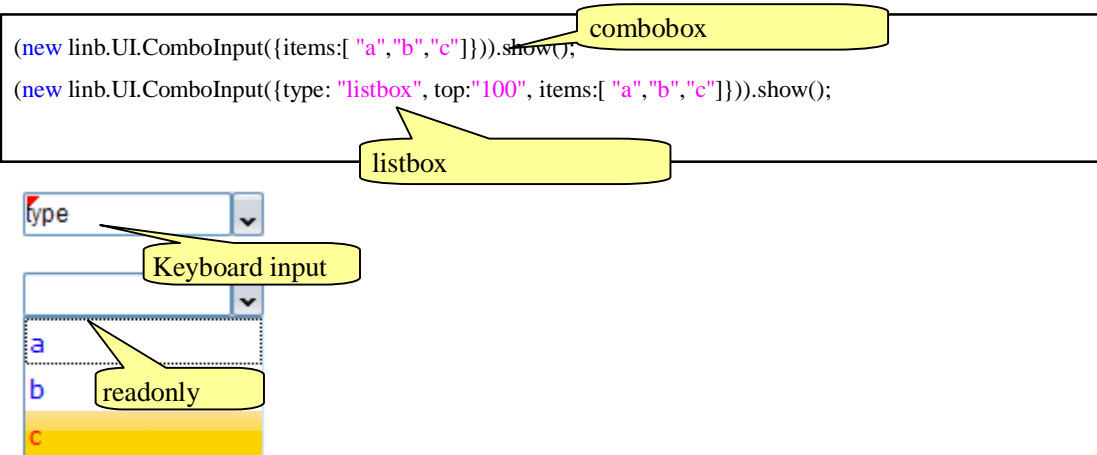
`chapter2\Input\index.html` is an overall example for Input.

### 3.6.9. linb.UI.ComboInput

`linb.UI.ComboInput` is an advanced Input.

#### 3.6.9.1. Pop list for selection

When type property was set to “combobox”, “listbox” or “helpinput”, click the command button will trigger to pop a list window for selection.



#### 3.6.9.2. combobox, listbox and helpinput

There's an `items` property in `linb.UI.ComboInput` (And all list related controls have this property too). Usually, we set `items` as a simple single layer array (like `["ia", "ib", "ic"]`). The lib? will convert this simple array to inner format:

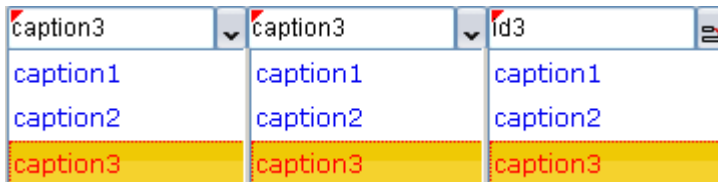
```
[
  {
    id: "ia",
    caption: "ia"
  },
  {
    id: "ib",
    caption: "ib"
  },
  {
    id: "ic",
    caption: "ic"
  }
]
```

- 1) combobox: Not readonly. The pop List shows “caption”; Input box shows “caption”; getValue returns “caption”.
- 2) listbox: Readonly. The pop List shows “caption”; Input box shows “caption”; getValue returns “id”.
- 3) helpinput: Not readonly. The pop List shows “caption”; Input box shows “id”; getValue returns “id”.

#### Input:

```
var items=[
  {
    id : "id1",
    caption : "caption1"
  },{
    id : "id2",
    caption : "caption2"
  },{
    id : "id3",
    caption : "caption3"
  }
];
linb.create('ComboInput',{position:'relative',items:items}).show();
linb.create('ComboInput',{position:'relative',items:items,type:'listbox'}).show();
linb.create('ComboInput',{position:'relative',items:items,type:'helpinput'}).show();
```

#### Output:



### 3.6.9.3. Date Picker

Sets type property to “date”.

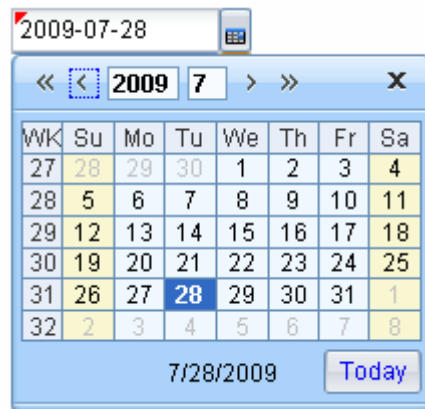
#### Input:

```
var ctrl=linb.create('ComboInput')
.setType('date')
.setValue(new Date)
.show();

__asynRun(function(){
  alert("The value is a timestamp string:"+ctrl.getValue());
  alert("You can convert it to date object:"+new Date(parseInt(ctrl.getValue())));
});
```

Date object or  
timestamp string

#### Output:



### 3.6.9.4. Time Picker

Sets type property to “time”.

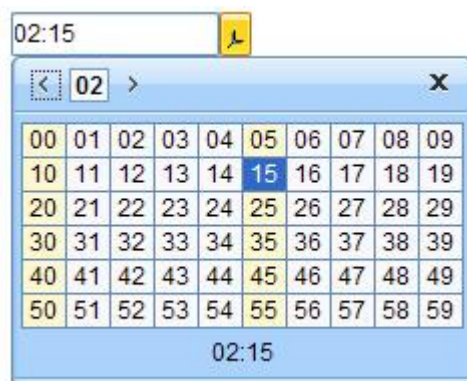
**Input:**

```
var ctrl=linb.create('ComboInput')
    .setType('time')
    .setValue('2:15')
    .show();

linb.alert("The value is a string : "+ctrl.getValue());
```

Sets string

**Output:**



### 3.6.9.5. Color Picker

Sets type property to “color”.

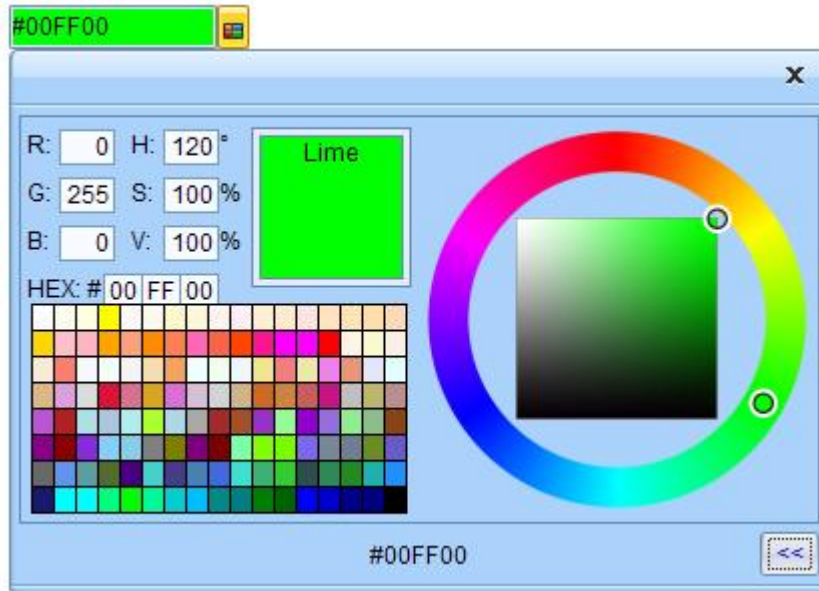
**Input:**

```
var ctrl=linb.create('ComboInput')
.setType('color')
.setValue('#00ff00')
.show();

linb.alert("The value is a string : "+ctrl.getValue());
```

Sets string

**Output:**



### 3.6.9.6. File Picker

Sets type property to “upload”.

**Input:**

```
var ctrl=linb.create('ComboInput')
.setType('upload')
.show();
```

**Output:**



Note: use getUploadObj function to get the file’s handler

```
ctrl.getUploadObj)
```

### 3.6.9.7. Getter

Sets type property to “getter”.

**Input:**

```
var ctrl=linb.create('ComboInput')
.setType('getter')
.beforeComboPop(function(profile){
    profile.boxing().setUIValue(_id())
})
.show();
```

Sets value in beforeComboPop event

Output:



### 3.6.9.8. Custom Pop Window

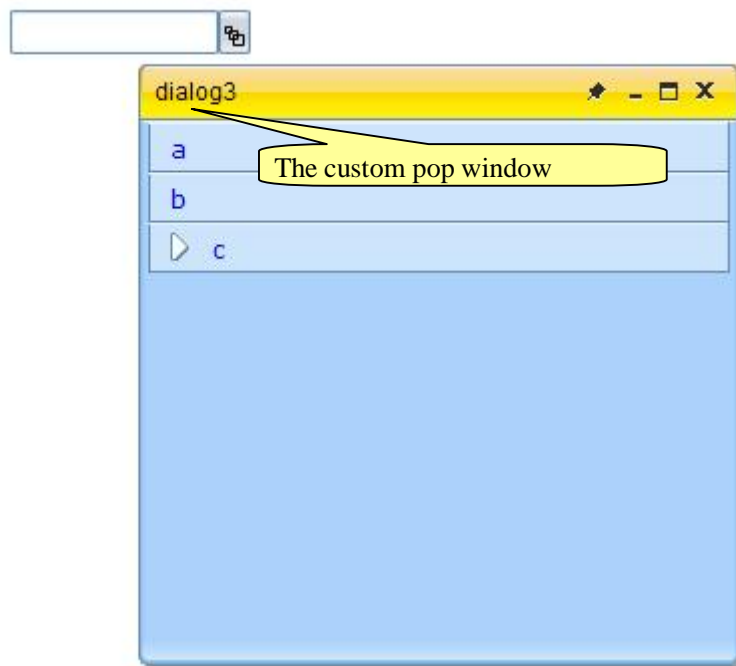
Sets type property to “cmdbox”, or “popbox”.

Input:

```
var ctrl=linb.create('ComboInput')
.setType('popbox')
.beforeComboPop(function(profile){
    var dlg=new linb.UI.Dialog, tb;
    dlg.append(tb=new linb.UI.TreeBar({items:["a","b",{id:"c",sub:["c1","c2","c3"]}]}));
    tb.onItemSelected(function(profile,item){
        ctrl.setUIValue(item.id);
        dlg.destroy();
    });
    dlg.show(null,true,100,100)
})
.show();
```

Shows custom pop window in beforeComboPop event

Output:





### 3.6.9.9. Command Buttons

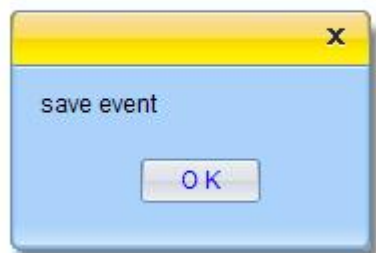
You can use `commandBtn` property to add an command button into `ComboInput` control. The following types are available for `commandBtn` property:

- | “none”: no command button
- | “save”: It’s a save button
- | “add” : It’s a add button
- | “remove” : It’s a remove button
- | “delete” : It’s a delete button
- | “custom” : custom button ( sets `imageClass` or `imagePos` to custom it)

**Input:**

```
(new linb.UI.ComboInput).setPosition('relative').setCommandBtn('none').show();
(new linb.UI.ComboInput).setPosition('relative').setCommandBtn('save').onCommand(function(){ linb.alert('save event'); }).show();
(new linb.UI.ComboInput).setPosition('relative').setCommandBtn('add').onCommand(function(){ linb.alert('add event'); }).show();
(new linb.UI.ComboInput).setPosition('relative').setCommandBtn('remove').onCommand(function(){ linb.alert('remove event'); }).show();
(new linb.UI.ComboInput).setPosition('relative').setCommandBtn('delete').onCommand(function(){ linb.alert('delete event'); }).show();
(new linb.UI.ComboInput).setPosition('relative').setCommandBtn('save').onCommand(function(){ linb.alert('save event'); }).show();
```

**Output:**



#### NOTE

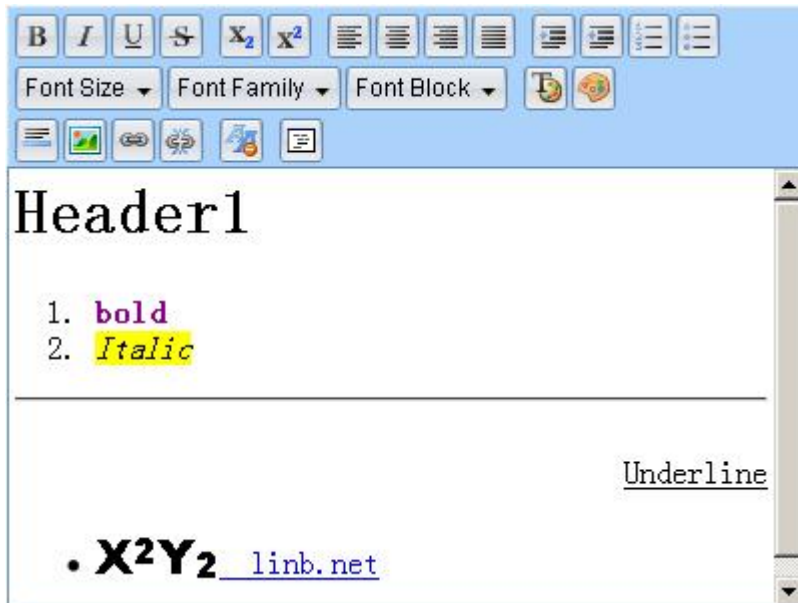
`chapter2\ComboInput\index.html` is an overall example for `ComboInput`.

### 3.6.10. RichEditor

**Input:**

```
(new linb.UI.RichEditor()).show();
```

**Output:**



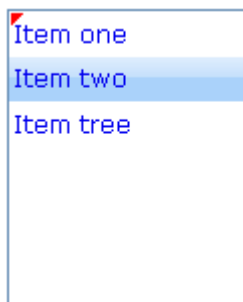
## 3.7. List related

This section relates to the following controls: `linb.UI.List`, `linb.UI.RadioButton` and `linb.UI.IconList` and `linb.UI.Gallery`.

### 3.7.1. A Simple one

```
linb.create("List")
.setItems(["Item one", "Item two", "Item tree"])
.onItemSelected(function(profile, item){
    linb.message(item.id);
})
.show();
```

onItemSelected event



### 3.7.2. A little bit complicated

```

var renderer=function(o){
    return '<span style="width:40px">'+o.col1+'</span>' + ' <span style="width:60px">'+o.col2+'</span>' +
    '<span style="width:40px">'+o.col3+'</span>';
};

linb.create("List")
.setWidth(160)
.setItems([[
    id:"a",
    col1:'Name',
    col2:'Gender',
    col3:'Age',
    renderer:renderer,
    itemStyle:'border-bottom:solid 1px #C8E1FA;font-weight:bold;',
    ],
    [
    id:"b",
    col1:'Jack',
    col2:'Male',
    col3:'23',
    renderer:renderer
    ],
    [
    id:"c",
    col1:'Jenny',
    col2:'Female',
    col3:'32',
    renderer:renderer
    ]
    ])
.beforeUIValueSet(function (profile, ov, nv){
    return nv!="a"
})
.show();

```

Gives a render function

Extra variables

For the header item

#### Result:

Name	Gender	Age
Jack	Male	23
Jenny	Female	32

A Grid List

The above special render function applies to any control's caption property (e.g. linb.UI.Button, linb.UI.Label); and any control's sub item caption property (e.g. linb.UI.List, linb.UI.TreeBar) .

```
linb.create("SCheckBox")
.setCaption("caption")
.setRenderer(function(prop){return prop.caption+" "+this.key})
.show();
```



### 3.7.3. RadioBox

linb.UI.RadioBox is derived from linb.UI.List.

**Input:**

```
linb.create("RadioBox")
.setItems(["a","b","c"])
.onItemSelected(function(profile,item){
    linb.message(item.id);
})
.show();
```

**Output:**



### 3.7.4. IconList and Gallery

Both are derived from linb.UI.List.

**Input:**

```
linb.create("IconList")
.setItems([{id:'a',image:'img/a.gif'},{id:'b',image:'img/b.gif'},{id:'c',image:'img/c.gif'}])
.onItemSelected(function(profile,item){
    linb.message(item.id);
})
.show();
```

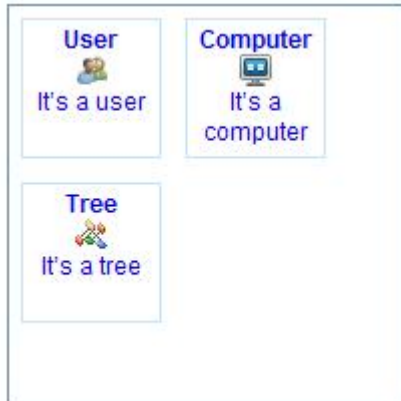
**Output:**



**Input:**

```
linb.create("Gallery")
.setItemWidth(64).setItemHeight(64)
.setItems([
  {id:'a',image:'img/a.gif',caption:'User',comment:'It's a user'},
  {id:'b',image:'img/b.gif',caption:'Computer',comment:'It's a computer'},
  {id:'c',image:'img/c.gif',caption:'Tree',comment:'It's a tree'}
])
.onItemSelected(function(profile,item){
  linb.message(item.id);
})
.show();
```

Item's height

**Output:**

### 3.7.5. Item selection

You can use “**setUIValue**” function to select a item in List, or use “**fireItemClickEvent**” function to get the same result. “fireItemClickEvent” function will trigger “onItemSelected” event, “setUIValue” won’t.

```
var ctrl=linb.create("List")
.setItems(["Item one", "Item two", "Item tree"])
.onItemSelected(function(profile,item){
  linb.message(item.id);
})
.show();

_.asynRun(function(){
  ctrl.fireItemClickEvent("Item two");
},1000);

_.asynRun(function(){
  ctrl.setUIValue("Item one");
},2000);
```

Trigger onItemSelected event

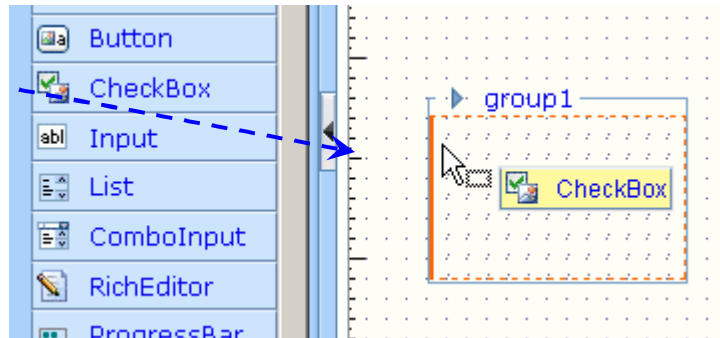
### 3.7.6. Container related

This section relates to the following controls: linb.UI.Group, linb.UI.Pane , linb.UI.Panel,

linb.UI.Block.

linb.UI.Dialog, linb.UI.Layout and linb.UI.Tabs /Stacks/ButtonViews are container controls too, we will **give an example?** these controls in separate sections.

Container is those controls that can have child controls. In jsLinb Designer, you can drag a child control and drop it into a container control. Just like this,



#### Input 1:

```
(new linb.UI.Group)
.append(new linb.UI.SButton)
.show();
```

append

#### Input 2:

```
var con = new linb.UI.Group;
con.show();
(new linb.UI.SButton).show(con);
```

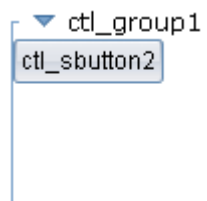
show

#### Input 3:

```
linb.create({
  key: "linb.UI.Group",
  children: [{key: "linb.UI.SButton"}]}
}).show();
```

In children object

#### Output:



### 3.7.7. Pane and Panel

linb.UI.Pane is a single node control. It's derived from linb.UI.Div. linb.UI.Panel has a border and a

title bar.

**Input:**

```
(new linb.UI.Pane)
.append(new linb.UI.SButton)
.show()
```

You can't see output, It's transparent

**Input:**

```
(new linb.UI.Panel)
.setDock("none")
.append(new linb.UI.SButton)
.show()
```

Sets dock to 'none'

**Output:**

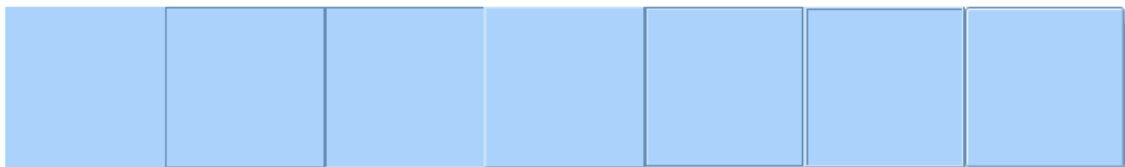


## 3.7.8. Block

**Input:**

```
linb.create("Block",{position:'relative',borderType:'none'}).show()
linb.create("Block",{position:'relative',borderType:'flat'}).show()
linb.create("Block",{position:'relative',borderType:'inset'}).show()
linb.create("Block",{position:'relative',borderType:'outset'}).show()
linb.create("Block",{position:'relative',borderType:'groove'}).show()
linb.create("Block",{position:'relative',borderType:'ridge'}).show()
linb.create("Block",{position:'relative',borderType:'none',border:true,shadow:true,resizer:true}).show()
```

**Output:**

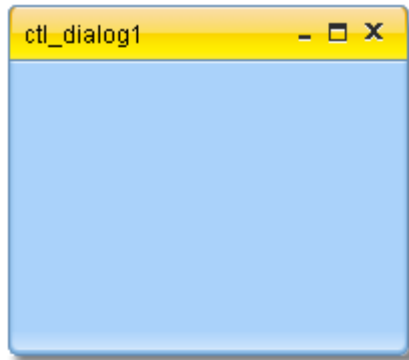


## 3.8. Dialog related

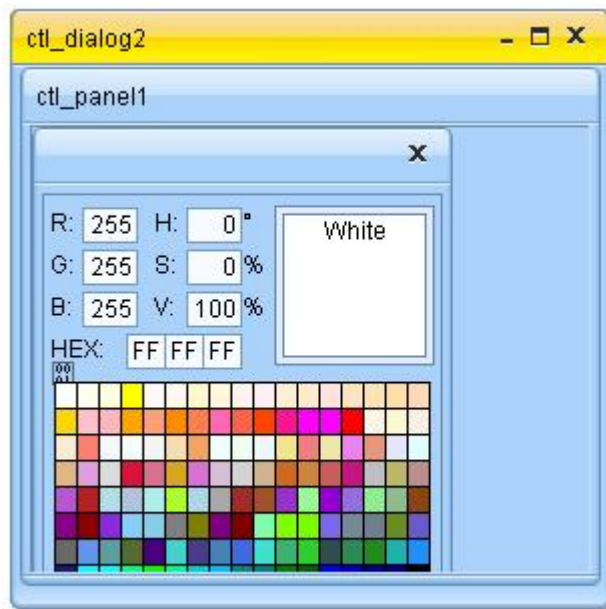
### 3.8.1. Normal state

**Input:**

```
(new linbUI.Dialog).show()
```

**Output:****Input:**

```
var dlg = (new linb.UI.Dialog).show();  
var panel;  
_.asynRun(function(){  
    dlg.append(panel=new linb.UI.Panel)  
},1000);  
_.asynRun(function(){  
    panel.append(new linb.UI.ColorPicker)  
},2000);
```

**Output:**

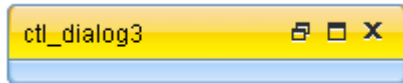
### 3.8.2. Min and Max status

**Input:**



```
var dlg = (new linb.UI.Dialog).setStatus("min").show();
_.asynRun(function(){
  dlg.setStatus("normal");
},1000);
_.asynRun(function(){
  dlg.setStatus("max");
},2000);
```

**Output:**



### 3.8.3. Modal Mode

**Input:**

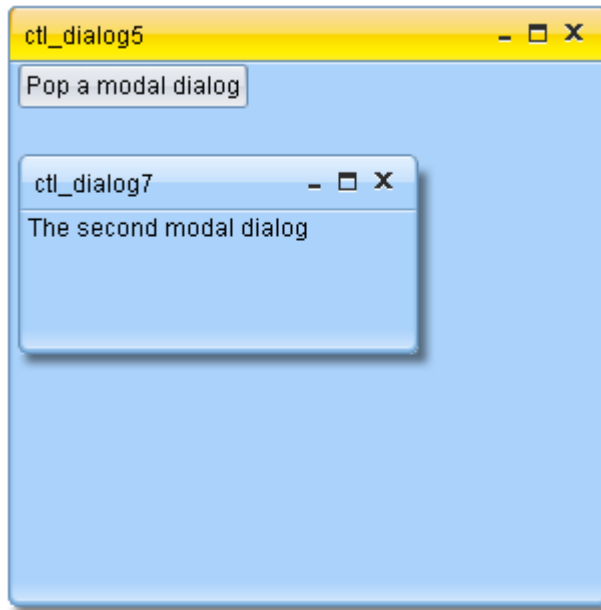
```
var dlg = (new linb.UI.Dialog).show();
dlg.append(panel=new linb.UI.SButton({
  caption: "Pop a modal dialog"
}),
{onClick:function(){
  linb.create("Dialog",{
    width:200,
    height:100,
    html:"The second modal dialog"
  }).showModal(dlg);
}}
))

(new linb.UI.Dialog)
.setHtml("The first modal dialog")
.show(null,true);
```

Annotations:

- Sets caption (points to `caption: "Pop a modal dialog"`)
- onClick event (points to `onClick:function(){`)
- Parent is dlg (points to `).showModal(dlg);`)
- Parent is html body (points to `.setHtml("The first modal dialog")`)

**Output:**



### 3.9. Layout Control

#### Input:

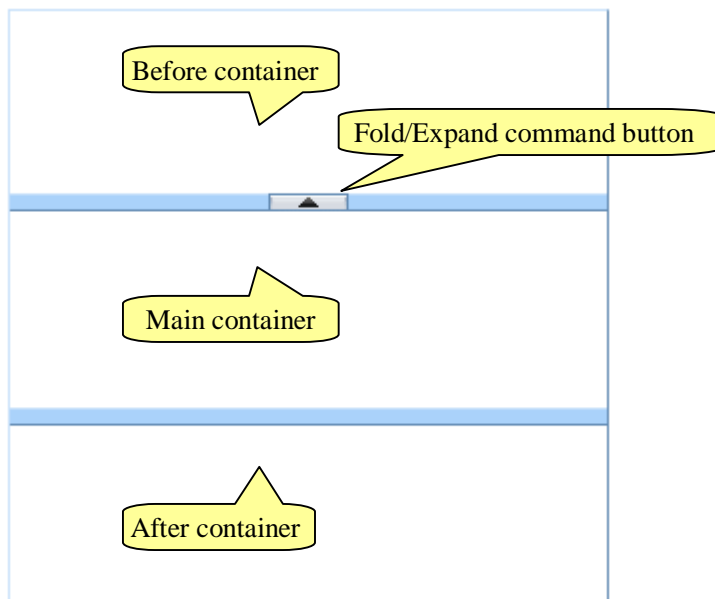
```
var block=linb.create("Block").setWidth(300).setHeight(300);
var layout=linb.create("Layout",{items:[
  {id:'before',
    pos:'before',
    size:100,
    cmd:true
  }, {id:'after',pos:'after',size:100}
]});
block.append(layout).show();
```

Append into a block

Size to 100

Has a command button

#### Output:



**Input:**

```

var block=linb.create("Block").setWidth(400).setHeight(100);
var layout=linb.create("Layout",{items:[
  {id:'before',
   pos:'before',
   size:100,
   cmd:true,
   folded:true,
   max:120,
   min:80
}, {
  id:'after',
  pos:'after',
  cmd:true,
  locked:true,
  size:50
}, {
  id:'after2',
  pos:'after',
  size:50
}],type: 'horizontal'});
block.append(layout).show();

```

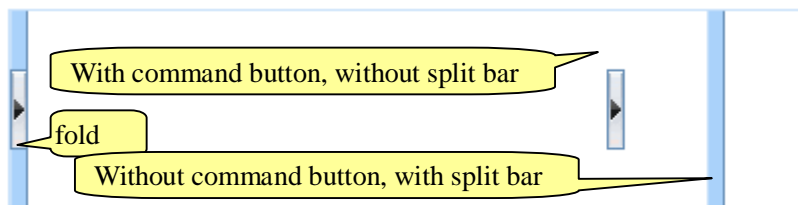
Default status is fold

Max size is 120

Min size is 80

Size locked

horizontal

**Output:****NOTE**

`chapter2\Layout\index.html` is an overall example for Layout.

## 3.10. Multi-pages Controls

Three multi-pages controls: `linb.UI.Tabs`, `linb.UI.Stacks` and `linb.UI.ButtonViews`.

**Input:**

```

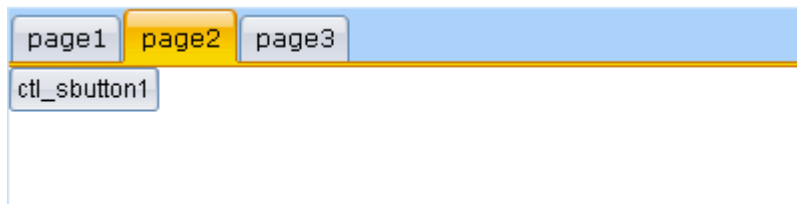
var block=linb.create("Block").setWidth(400).setHeight(100);
var pages=linb.create("Tabs",{
  items:["page1","page2","page3"],
  value:"page2"
});
block.append(pages).show();

_._asynRun(function(){
  pages.append(new linb.UI.SButton,"page2")
},1000);
    
```

Annotations:

- 3 pages (points to items array)
- The default page (points to value: "page2")
- Append to a block (points to block.append(pages).show();)
- Adds a SButton to 2th page (points to pages.append(new linb.UI.SButton, "page2")

**Output:**



### 3.10.1. noPanel property

For linb.UI.Tabs and linb.UI.ButtonViews, when “noPanel” property was set to true, they no longer are the container control. So, don’t append any children control to tabs in this case.

**Input:**

```

var block=linb.create("Block").setWidth(400).setHeight(300).show();
var items=["page1","page2","page3"];
linb.create("Tabs",{
  items:items,
  value:"page2",
  position:'relative',
  width:'auto',
  height:'auto',
  dock:'none',
  noPanel:true
}).show(block);

linb.create("ButtonViews",{
  items:items,
  value:"page2",
  position:'relative',
  width:'auto',
  height:32,
  barSize:30,
  dock:'none',
  noPanel:true
}).show(block);
    
```

Annotations:

- Set position to 'relative' (points to position:'relative')
- Auto width (points to width:'auto')
- Auto height (points to height:'auto')
- No container (points to noPanel:true in Tabs)
- Set height to buttonview (points to height:32)
- No container (points to noPanel:true in ButtonViews)

**Output:**

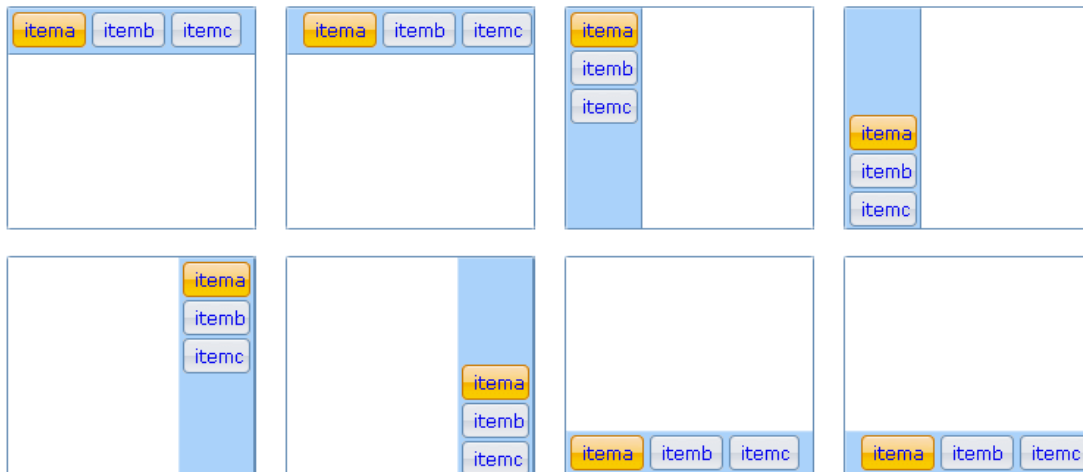


### 3.10.2. ButtonViews types

There are three properties used to define the ButtonViews' layout:

- l **barLocation**: Used to set the location of the command button bar.  
In 'top', 'bottom', 'left', 'right'.
- l **barHAlign**: Used to set command buttons horizontal alignment  
In 'left', 'right'. Only for **barLocation** is 'top' or 'bottom'
- l **barVAlign**: Used to set command buttons vertical alignment  
In 'left', 'right'. Only for **barLocation** is 'left' or 'right'

The below picture shows all the eight possible ButtonViews layouts:



In [chapter2\ButtonViews\index.html](#)

#### NOTE

[chapter2\ButtonViews\index.html](#) is an overall example for ButtonViews.

### 3.10.3. Page selection

You can use “**setUIValue**” function to select a page, or use “**fireItemClickEvent**” function to get the same result. “**fireItemClickEvent**” function will trigger “**onItemSelected**” event, “**setUIValue**” won't.

**Input:**

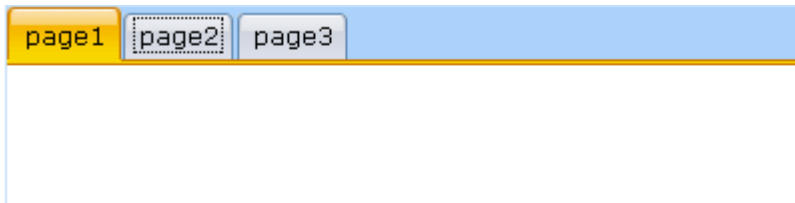
```
var block=linb.create("Block").setWidth(400).setHeight(100).show();
var pages=linb.create("Tabs",{
  items:["page1","page2","page3"]
})
.onItemSelected(function(profile,item){
  linb.message(item.id);
})
.show(block);

_.asRun(function(){
  pages.fireItemClickEvent("page2");
},1000);

_.asRun(function(){
  pages.setUIValue("page1");
},2000);
```

Trigger onItemSelected event

**Output:**



### 3.10.4. Pages

#### 3.10.4.1. Close and options Button

Each page can hold a “close” button and a “options” button. Click this button will close the page.

**Input:**

```

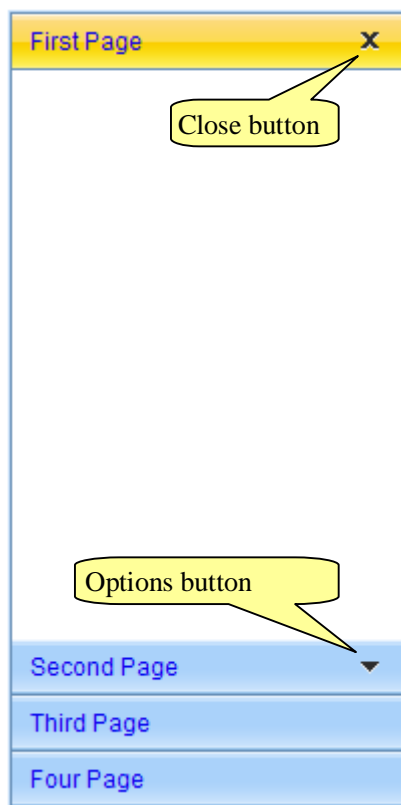
var block=linb.create("Block").setWidth(200).setHeight(400).show(), stacks;
block.append(stacks=new linb.UI.Stacks({
  value:'a',
  items:[{
    id:'a',
    caption:'First Page',
    closeBtn:true
  },{
    id:'b',
    caption:'Second Page',
    optBtn:true
  },{
    id:'c',
    caption:'Third Page'
  },{
    id:'d',
    caption:'Fourth Page'
  }]
}));
stacks.onShowOptions(function(profile,item){
  linb.message(" You clicked "+item.caption)
});

```

Close button

Options button

Click options to trigger onShowOptions event

**Output:**

Two events can be fired when “close” button was clicked:

- l beforePageClose: Fired before user clicked the close button on a page. If returns false, the page won't be closed.
- l afterPageClose: Fired after user clicked the close button on a page.

### 3.10.4.2. Add/Remove Pages

#### Input:

```

var block=linb.create("Block").setWidth(400).setHeight(100).show(), tabs;
block.append(tabs=new linb.UI.Tabs({
  value:'a',
  items:[{
    id:'a',
    caption:'First Page'
  },{
    id:'b',
    caption:'Second Page'
  }]
}));
_asyRun(function(){
  tabs.insertItems([
    id:'c',
    caption:'Third Page'
  ],{
    id:'d',
    caption:'Fourth Page'
  });
},500);
_asyRun(function(){
  tabs.insertItems('Fifth Page');
},1000);
_asyRun(function(){
  tabs.removeItem('d');
},1500);
_asyRun(function(){
  tabs.removeItem(['b','c']);
},2000);

```

Close button

Adds two pages

Adds one more

Removes this page

Removes two more



## 3.10.5. Dynamic content loading

### 3.10.5.1. onIniPanelView

```
var block=linb.create("Block").setWidth(400).setHeight(100).show(),
tabs=new linb.UITabs({
  value:'a',
  items:[{
    id:'a',
    caption:'First Page'
  },{
    id:'b',
    caption:'Second Page'
  },{
    id:'c',
    caption:'Third Page'
  }]
});
tabs.onIniPanelView(function(profile,item){
  profile.boxing().getPanel(item.id).append(new linb.UI.SButton)
});
block.append(tabs);
```

### 3.10.5.2. beforeUIValueSet/afterUIValueSet

It's a fine-grained mechanism.

```

var block=linb.create("Block").setWidth(400).setHeight(100).show(), tabs;
block.append(tabs=new linb.UITabs({
  value:'a',
  items:[{
    id:'a',
    caption:'First Page'
  },{
    id:'b',
    caption:'Second Page'
  },{
    id:'c',
    caption:'Third Page'
  }]
}));
tabs.beforeUIValueSet(function(profile,ovalue,value){
  if(value=='b')
    return false;
});
tabs.afterUIValueSet(function(profile,ovalue,value){
  if(value=='c'){
    var item=profile.getItemById(value);
    if(!item.$ini){
      profile.boxing().append(new linb.UISButton);
      item.$ini=true;
    }
  }
});

```

## 3.11. Menus and toolbars

### 3.11.1. Pop Menu

**Input:**

```

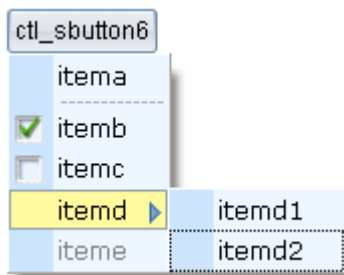
var pm=linb.create('PopupMenu')
.setItems([
    {"id":"itema", "caption":"itema", "tips":"item a"},
    {"type":"split"},
    {"id":"itemb", "type":"checkbox", value:true, "caption":"itemb", "tips":"item b"},
    {"id":"itemc", "caption":"itemc", "type":"checkbox", "tips":"item c"},
    {"id":"itemd", "caption":"itemd", "tips":"item d", sub:[
        {"id":"itemd1", "caption":"itemd1"},
        {"id":"itemd2", "caption":"itemd2"}
    ]},
    {"id":"iteme", "caption":"iteme", "tips":"item d", disabled:true}
])
.onMenuSelected(function(profile,item){
    linb.message(item.id + (item.type=="checkbox"? " : " + item.value: ""))
});

linb.create('SButton')
.onClick(function(profile){
    pm.pop(profile.getRoot())
})
.show();
    
```

Annotations in the original image:

- Checkbox type (points to `"checkbox"` in `itemb` and `itemc`)
- Sub pop menu (points to the `sub` array in `itemd`)
- event (points to the `onMenuSelected` function)
- Disabled it (points to `disabled:true` in `iteme`)
- For position (points to `pm.pop(profile.getRoot())`)

Output:



### 3.11.2. MenuBar

Input:

```

var pm=linb.create('MenuBar')
.setItems([
  {
    "id": "file", "caption": "File",
    "sub": [
      {
        "id": "newproject",
        "caption": "New Project"
      },
      {
        "id": "openproject", "caption": "Open Project",
        "add": "Ctrl+Alt+O",
        "image": "img/b.gif",
        "sub": ["option 1", "option 2"]
      },
      {
        "id": "closeproject", "caption": "Close Project"
      }
    ],
    "type": "split",
    {
      "id": "save", "caption": "Save",
      "image": "img/a.gif",
    },
    {
      "id": "saveall", "caption": "Save All",
      "add": "Ctrl+Alt+S",
      "image": "img/c.gif"
    }
  },
  {
    "id": "tools", "caption": "Tools",
    "sub": [
      {
        "id": "command", "caption": "Command Window"
      },
      {
        "id": "spy", "caption": "Components Spy"
      }
    ]
  },
  {
    "id": "build", "caption": "Build",
    disabled: true,
    "sub": [
      {
        "id": "debug",
        "caption": "Debug"
      }
    ]
  }
]).show()
    
```

Pop menu data

Extra data

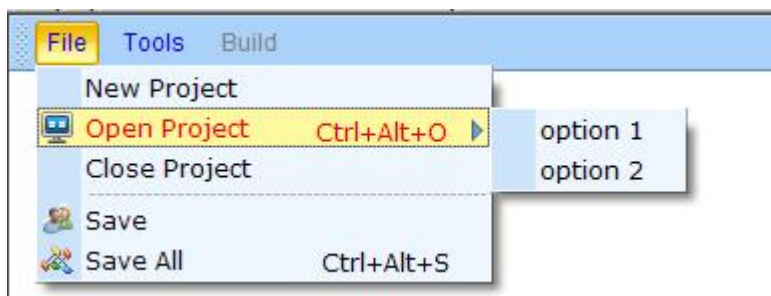
Sub pop menu

A split

An icon

Disabled it

**Output:**



### 3.11.3. Toolbars

**Input:**

```

linb.create('ToolBar',{items:[{
  "id" : "align",
  "sub" : [
    {"id" : "left","caption" : "left"},
    {"id" : "center","caption" : "center"},
    {type:'split'},
    {"id" : "right","caption" : "center"}
  ]
},{
  "id" : "code",
  "sub" : [{
    "id" : "format","caption" : "format",
    label:"label",
    image:"img/a.gif",
    "dropButton" : true
  }]
}]
})
.onClick(function(profile,group,item){
  linb.message(group.id + " : " + item.id)
})
.show();

```

Diagram annotations for the code above:

- Button group data (points to `"id" : "align"`)
- Button data (points to `{ "id" : "left", "caption" : "left" }`)
- A split (points to `{ type: 'split' }`)
- With a label (points to `label: "label"`)
- With an icon (points to `image: "img/a.gif"`)
- A drop (points to `"dropButton" : true`)
- Group object (points to the outer `sub` array)
- Button (points to the inner `sub` object)

Output:



## 3.12. TreeBar and TreeView

### 3.12.1. Three selection mode

All controls derived from `linb.UI.absList` have three options mode.

#### 3.12.1.1. No-selection

Input:

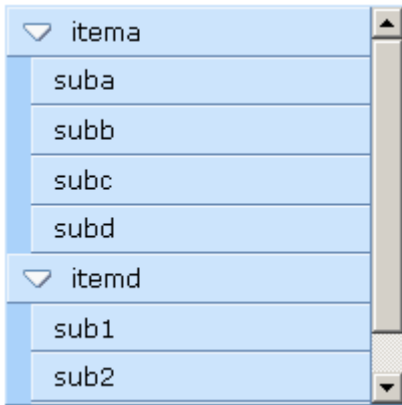
```

var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{items:[{ id : "itema", sub : ["suba","subb","subc","subd"]},
{ id : "itemd", sub : ["sub1","sub2","sub3"]}]})
.setSelMode("none")
.onItemSelected(function(profile,item){
  linb.message(item.id);
}).show(block);

```

Diagram annotation: Sets to 'none' (points to `.setSelMode("none")`)

Output:



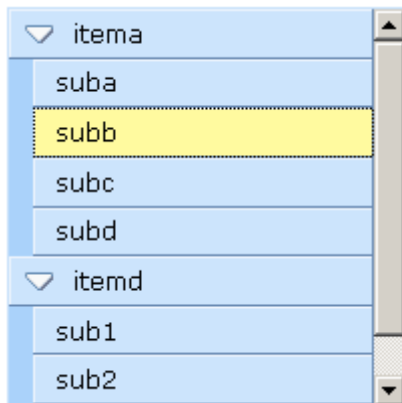
### 3.12.1.2. Single-selection

**Input:**

```
var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{items:[{ id : "itema", sub : ["suba","subb","subc","subd"]},
{id : "itemd", sub : ["sub1","sub2","sub3"]}]})
.setSelMode("single")
.onItemSelected(function(profile,item){
    linb.message(item.id);
}).show(block);
```

Sets to single

**Output:**



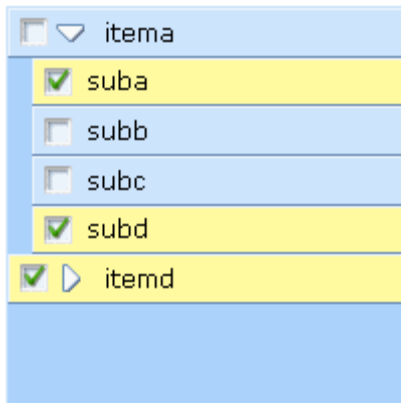
### 3.12.1.3. Multi-selection

**Input:**

```
var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{items:[{ id : "itema", sub : ["suba","subb","subc","subd"]},
{id : "itemd", sub : ["sub1","sub2","sub3"]}]})
.setSelMode("multi")
.onItemSelected(function(profile,item){
    linb.message(item.id);
}).show(block);
```

Sets to 'multi'

### Output:



## 3.12.2. Group Item

### Input:

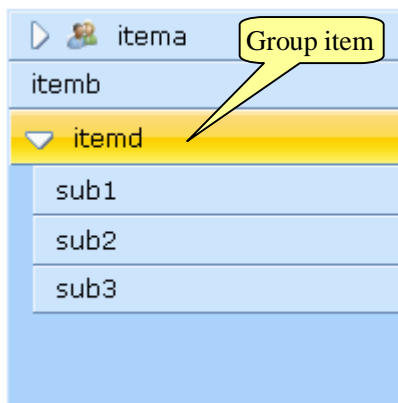
```
var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{items:[{
  id:"itema",
  image:"img/a.gif",
  sub:["suba","subb","subc","subd"]
},
{id:"itemb"},
{
  id:"itemd",
  group:true,
  sub:["sub1","sub2","sub3"]
}
]}).show(block);
```

With an icon

Sub items

It's a group

### Output:



Group item

## 3.12.3. Expand all nodes by default

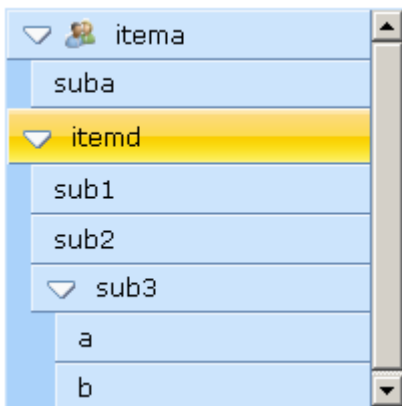
### Input:

```

var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{
  iniFold:false,
  items:[{
    id : "itema",
    image : "img/a.gif",
    sub : ["suba"]
  },
  {
    id : "itemd",
    group:true,
    sub : ["sub1","sub2",{
      id:"sub3",sub:["a","b"]
    }]
  }
]).show(block);

```

Sets iniFold property to false

**Output:****3.12.4. Mutex Expand**

```

var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{
  singleOpen:true,
  items:[{
    id : "itema",
    image : "img/a.gif",
    sub : ["suba"]
  },
  {
    id : "itemd",
    group:true,
    sub : ["sub1","sub2",{
      id:"sub3",sub:["a","b"]
    }]
  }
]).show(block);

```

Mutex Expand



### 3.12.5. Dynamic Destruction

```
var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{
  dynDestory:true,
  items:[{
    id : "itema",
    image : "img/a.gif",
    sub : ["suba"]
  },
  {
    id : "itemd",
    group:true,
    sub : ["sub1","sub2",{
      id:"sub3",sub:["a","b"]
    }]
  }
]
}).show(block);
```

Dynamic Destruction

### 3.12.6. Dynamically loading

Input:

```
var block=new linb.UI.Block({width:200,height:200}).show();
linb.create("TreeBar",{
  singleOpen:true,
  dynDestory:true,
  items:[{
    id : "itema",
    sub : true
  },
  {
    id : "itemb",
    sub : true
  }
]
})
.onGetContent(function(profile,item,callback){
  if(item.id=="itema"){
    var rnd=_();
    callback([rnd+"-a",rnd+"-b",rnd+"-c"]);
  }
  if(item.id=="itemb")
    return ["itembsub1","itembsub2","itembsub3"];
})
.show(block);
```

Mutex Expand

Dynamic Destruction

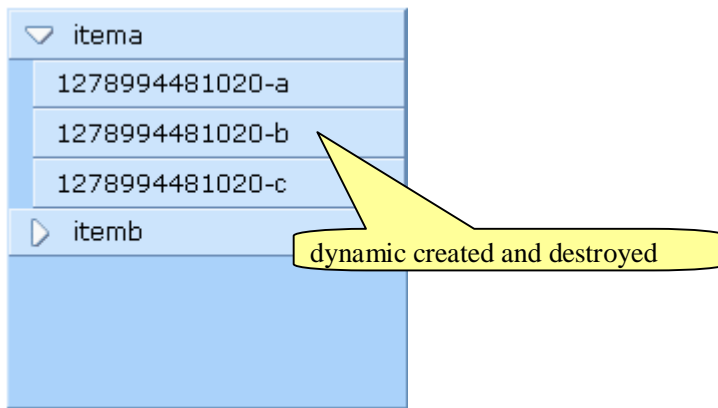
Wants to load children dynamically

Takes time stamp as a random string

Asynchronous or synchronous callback

Can also be returned directly

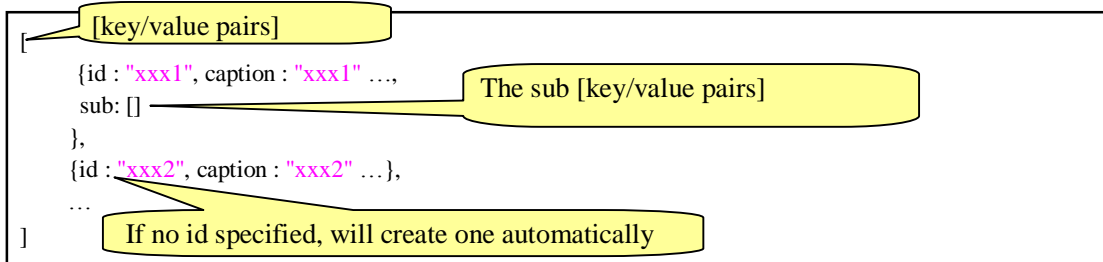
Output:



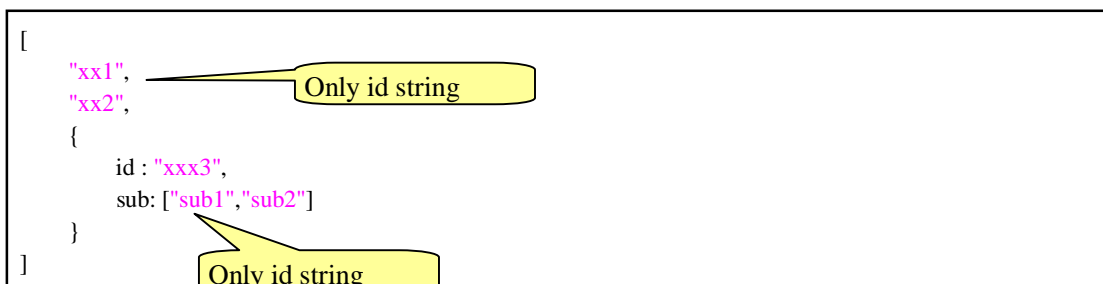
### 3.13. TreeGrid

#### 3.13.1. Header and Rows

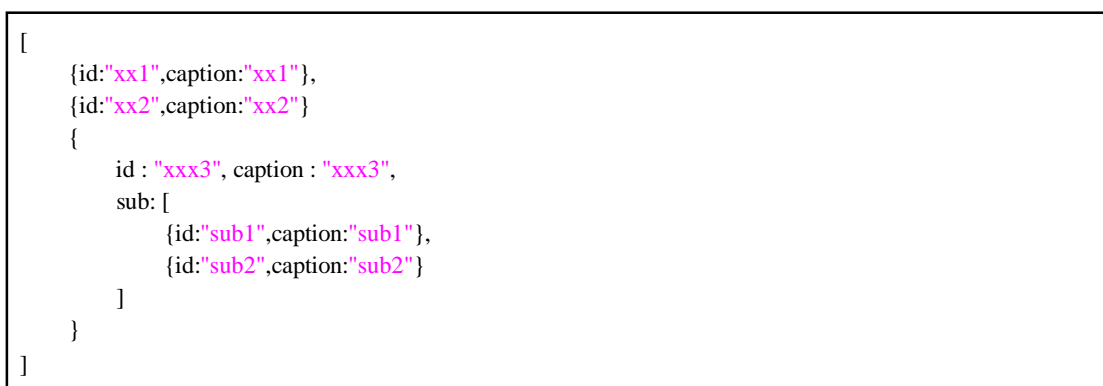
The header property and rows property in TreeGrid are Array of key/value pairs, like,



It can be written as a simplified format,



When call setHeader/setRows, the simplified format can be convert to,



### 3.13.1.1. Sets standard format

```
var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false);
tg.setHeader([
  {id:"col1", caption:"Name"},
  {id:"col2", caption:"Age", width:40}
]).setRows([
  {id:"row1",cells:[{
    value:'Jack',caption:'Jack'
  },{
    value:23,caption:'23'
  }]},
  {id:"row2",cells:[{
    value:'John',caption:'John'
  },{
    value:32,caption:'32'
  }]}
]).show(block);
```

Name	Age	
Jack	23	
John	32	

### 3.13.1.2. Sets simplified format

```
var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false);
tg.setHeader(["Name", "Age"]);
tg.setRows([[ 'Jack', 23], [ 'John', 32]]);
tg.show(block);
```

### 3.13.2. getHeader

Calls getHeader function to return the header data. There are three format,

- l getHeader(): returns memory data;
- l getHeader("data"): returns the standard format data;
- l getHeader("min"): returns the simplified format data;

```

var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.show(block);
linb.log(tg.getHeader());
linb.log(tg.getHeader("data"));
linb.log(tg.getHeader("min"));

```

Comparing these three formats



### 3.13.3. getRows

Calls `getRows` function to return the rows data. Similarly, there are three format,

- l `getRows ()`: returns memory data;
- l `getRows ("data")`: returns the standard format data;
- l `getRows ("min")`: returns the simplified format data;

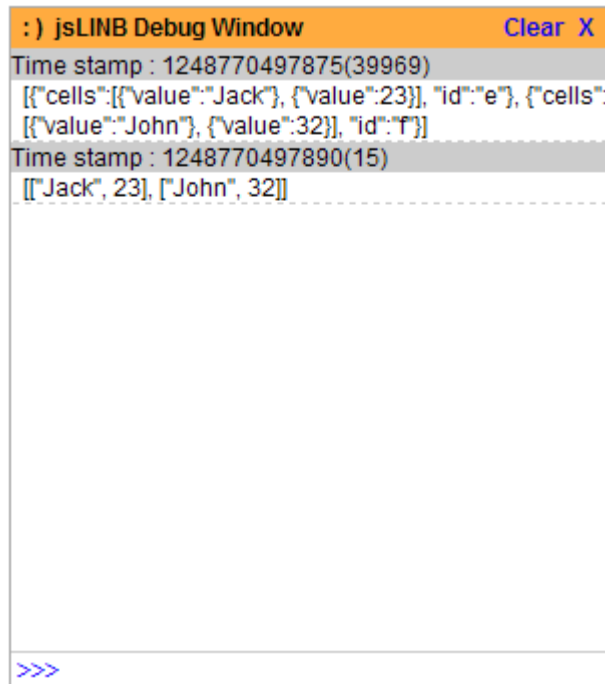
```

var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.show(block);
linb.log(tg.getRows("data"));
linb.log(tg.getRows("min"));
//console.log(tg.getRows());

```

Comparing these three formats

There is circular reference in memory data,  
can't be directly serialized



The rows memory data in firebug:



### 3.13.4. Active Modes

There are three active modes for TreeGrid:

- ! non-active appearance : activeMode is "none";
- ! the row-active appearance: activeMode is "row" ;
- ! the cell-active appearance: activeMode is "cell";

### 3.13.4.1. non-active appearance

Input:

```
var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setActiveMode("none")
.show(block)

.afterRowActive (function(profile,row){
    linb.message(row.id);
})
.afterCellActive (function(profile,cell){
    linb.message(cell.value);
})
```

No row handler

Sets to 'none'

Does not trigger events

Output:

Name	Age
Jack	23
John	32

non-active appearance

### 3.13.4.2. row-active appearance

```
var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setActiveMode("row")
.show(block)

.afterRowActive (function(profile,row){
    linb.message(row.id);
})
.afterCellActive (function(profile,cell){
    linb.message(cell.value);
})
```

No row handler

Sets to "row"

Will be fired

Name	Age
Jack	23
John	32

non-active appearance

### 3.13.4.3. cell-active appearance

```

var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setActiveMode("cell")
.show(block)

.afterRowActive (function(profile,row){
    linb.message(row.id);
})
.afterCellActive (function(profile,cell){
    linb.message(cell.value);
})

```

No row handler

Sets to "row"

Will be fired

Name	
Jack	23
John	32

non-active appearance

### 3.13.5. Selection Mode

There are five selection modes for TreeGrid:

- ! Non-selection: activeMode is "none", or selMode is 'none'
- ! Single row selection: activeMode is "row", and selMode is 'single'
- ! Multi-rows selection: activeMode is "row", and selMode is 'multi'
- ! Single cell selection: activeMode is "cell", and selMode is 'single'
- ! Multi-cells selection: activeMode is "cell", and selMode is 'multi'

#### 3.13.5.1. Non-selection

Input:

```

var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setSelMode("none")
.show(block)

.afterUIValueSet(function(profile,ovalue,value){
    linb.message(value);
});

```

Non-selection

Won't be fired

**Output:**

Name	Age	
Jack	23	
John	32	

It's active appearance, not the selection

**Input:**

```
var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack, 23], [John, 32]])
.setActiveMode("none")
.setSelMode("none")
.show(block)

.afterUIValueSet(function(profile,ovalue,value){
    linb.message(value);
});
```

Non-active

Non-selection

Won't be fired

**Output:**

Name	Age	
Jack	23	
John	32	

Non-active appearance, non-selection

### 3.13.5.2. Single row selection

**Input:**

```
var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setHeader(["Name", "Age"])
.setRows([[Jack, 23], [John, 32]])
.setSelMode("single")
.show(block)

.afterUIValueSet(function(profile,ovalue,value){
    linb.message(value);
});
```

Sets to 'single' mode

Will be fired

**Output:**

Name	Age	
Jack	23	
John	32	

Selection appearance



### 3.13.5.3. Multi-row selection

#### Input:

```
var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(24)
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setSelMode("multi")
.show(block)

.afterUIValueSet(function(profile,ovalue,value){
    linb.message(value);
});
```

Row handler's width

Sets to 'multi' mode

Will be fired

#### Output:

	Name	Age
<input checked="" type="checkbox"/>	Jack	23
<input checked="" type="checkbox"/>	John	32

### 3.13.5.4. Single cell selection

#### Input:

```
var block=new linb.UI.Block({ width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setActiveMode("cell")
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setSelMode("single")
.show(block)

.afterUIValueSet(function(profile,ovalue,value){
    linb.message(value);
});
```

Sets to 'cell' mode

Sets to 'single' mode

#### Output:

	Name	Age
	Jack	23
	John	32

### 3.13.5.5. Multi-cells selection

#### Input:

```

var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setActiveMode("cell")
.setHeader(["Name", "Age"])
.setRows([[Jack', 23], [John', 32]])
.setSelMode("multi")
.show(block)

.afterUIValueSet(function(profile,ovalue,value){
    linb.message(value);
});
    
```

Sets to 'multi' mode

**Output:**

Name	Age	
Jack	23	
John	32	

### 3.13.6. The Tree Grid

**Input:**

```

var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(20)
.setHeader([
    {id:"col1", caption:"Name"},
    {id:"col2", caption:"Age", width:40}
]).setRows([
    {id:"row1",cells:[Jack',23]},
    {id:"row2",cells:[John',32],
      sub:[{id:"row21",cells:[Tom',24]},
           {id:"row22",cells:[Bob',25]}
    ]}
]).show(block)
    
```

Row header is a must for tree grid

A row has sub rows

**Output:**

	Name	Age	
	Jack	23	
▼	John	32	
	Tom	24	
	Bob	25	

No Indentation here

**Input:**

```

var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(20)
.setGridHandlerCaption("Name")
.setRowHandlerWidth(80)
.setHeader([
  {id:"col2",caption:"Age",width:40}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23]},
  {id:"row2",caption:'John',cells:[32],
    sub:[{id:"row21",caption:'Tom',cells:[24]},
          {id:"row22",caption:'Bob',cells:[25]}
    ]
  }
]).show(block)

```

Grid handler caption

Row's caption

Output:

Name	Age
Jack	23
John	22
Tom	24
Bob	25

Indentation in row handler

### 3.13.7. Column config

#### 3.13.7.1. The first column

In order to show the first column, you have to set rowHandler to [true].

**Input:**

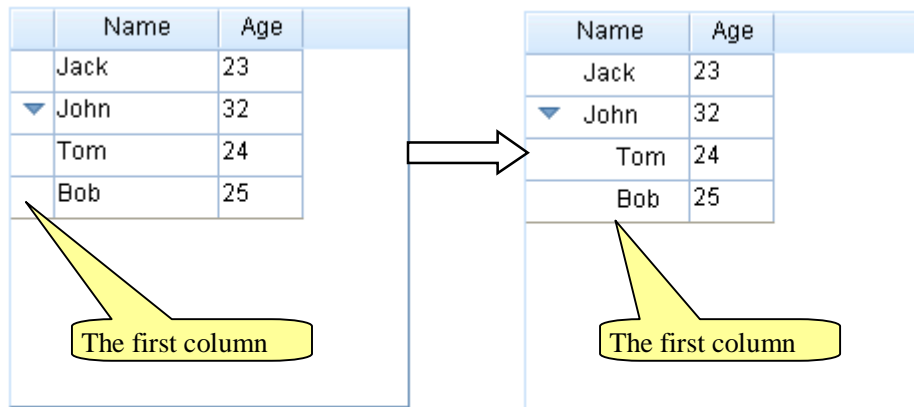
```

var block=new linb.UI.Block({width:200,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23]},
  {id:"row2",caption:'John',cells:[32],
    sub:[{id:"row21",caption:'Tom',cells:[24]},
          {id:"row22",caption:'Bob',cells:[25]}
    ]
  }
]).show(block)

```

Sets row handler's width

**Output:**



上一节中缩进的例子

### 3.13.7.2. Column width

Input:

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40},
  {id:"col2",caption:"Part-time",width:90}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)
._asRun(function(){
  tg.updateHeader("col2",{width:70});
},1000);
```

The first column's width

Column's width

Modify column width dynamically

Output:

Name	Age	Part-time
Jack	23	true
John	32	false

### 3.13.7.3. Drag&Drop to modify column width

“colResizer” property in TreeGrid determines whether the column width can be modified with Drag&Drop. Each column can include a “colResizer” property too. The “colResizer” property in column has higher priority than in TreeGrid.

In jsLinb, "fine-grained Setting has higher priority than coarse-grained" is a base rule.

Input:

```

var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColResizer(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40},
  {id:"col2",caption:"Part-time",width:90,colResizer:true}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)
    
```

coarse-grained setting

fine-grained setting

**Output:**

Name	Age	Part-time
Jack	23	true
John	32	false

Only this column

### 3.13.7.4. Drag&Drop to modify column position

**Input:**

```

var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColMovable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40},
  {id:"col2",caption:"Part-time",width:90,colMovable:true}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)
    
```

coarse-grained setting

fine-grained setting

**Output:**

Name	Age	Part-time
Jack	23	true
John	32	false

### 3.13.7.5. Default Sorting

**Input:**

```

var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40},
  {id:"col2",caption:"Part-time",width:90,colSortable:true}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)

```

coarse-grained setting

fine-grained setting

**Output:**

Name	Age	Part-time <input checked="" type="checkbox"/>	
Jack	23	true	
John	32	fa	

Sorting icon

### 3.13.7.6. Custom Sorting

```

var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40},
  {id:"col2",caption:"Part-time",width:90,sortby:function(x,y){return -1}}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)

```

Custom sorting function

### 3.13.7.7. Hide columns

**Input:**

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColHidable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40,colHidable:true},
  {id:"col2",caption:"Part-time",width:90,colHidable:true}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)
```

Global setting

These 2 columns  
can be hidden

Output:

Name	Age	Part-time
Jack	23	<input checked="" type="checkbox"/>
John	32	<input type="checkbox"/>

### 3.13.7.8. Setting Cell Types in column header

Input:

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40,type:"number"},
  {id:"col2",caption:"Part-time",width:90,type:"checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)
```

Number

Checkbox type

Output:

Name	Age	Part-time
Jack	23	true
John	32	false

Name	Age	Part-time
Jack	23	<input checked="" type="checkbox"/>
John	32	<input type="checkbox"/>

### 3.13.7.9. column header style

Input:

```
var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1", caption:"Age", width:40, type: "number"},
  {id:"col2", caption:"Part-time", width:90, type: "checkbox", headerStyle: "font-weight:bold;" }
]).show(block)
```

Sets bold

**Output:**

Naem	Age	Part-time	
------	-----	-----------	--


### 3.13.7.10. column header icon

**Input:**

```
var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1", caption:"Age", width:40, type: "number"},
  {id:"col2", caption:"Part-time", width:90, type: "checkbox", renderer: function(h){
    return "<img style='vertical-align:middle' src='img/a.gif'> " + h.caption;
  }}
]).show(block)
```

Renderer function

**Output:**

Name	Age	 Part-time	
------	-----	---	--



### 3.13.7.11. Update column header dynamically

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40,type:"number"},
  {id:"col2",caption:"Part-time",width:90,type:"checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)

_.asynRun(function(){
  tg.updateHeader('col2','Full-time')
},1000)

_.asynRun(function(){
  tg.updateHeader('col2',{caption:'Part-time',width:40,headerStyle:'font-weight:bold',colResizer:false,
colSortable:false,colMovable:true,colHidable:true})
},2000)
```

Updates caption only

Those properties are updatable

## 3.13.8. Row config

### 3.13.8.1. Row height

Input:

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40,type:"number"},
  {id:"col2",caption:"Full-time",width:90,type:"checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true],height:120},
  {id:"row2",caption:'John',cells:[32,false]}
]).show(block)
```

Sets row height

Output:

Name	Age	Full-time	
Jack	23	<input checked="" type="checkbox"/>	
John	32	<input type="checkbox"/>	

### 3.13.8.2. Drag&Drop to modify row height

Input:

```
var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setRowResizer(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1", caption:"Age", width:40},
  {id:"col2", caption:"Full-time", width:90}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23, true]},
  {id:"row2",caption:'John',cells:[32, false],rowResizer:true}
]).show(block)
```

Global disabled rowResizer

This row has rowResizer

Output:

Name	Age	Full-time	
Jack	23	true	
John	32	false	

### 3.13.8.3. Setting cell type in row

Input:

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40,type:"number"},
  {id:"col2",caption:"Part-time",width:90,type:"checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false],type:"label"}
]).show(block)
```

Sets all cells in this row type to 'label'

Output:

Name	Age	Part-time
Jack	23	<input checked="" type="checkbox"/>
John	32	<input type="checkbox"/>

Name	Age	Part-time
Jack	23	<input checked="" type="checkbox"/>
John	32	false

### 3.13.8.4. Row style

Input:

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1",caption:"Age",width:40,type:"number"},
  {id:"col2",caption:"Full-time",width:90,type:"checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23,true]},
  {id:"row2",caption:'John',cells:[32,false],rowStyle:"background-color:#00ff00"}
]).show(block)
```

Sets styles

Output:

Name	Age	Full-time
Jack	23	<input checked="" type="checkbox"/>
John	32	<input type="checkbox"/>

### 3.13.8.5. Row numbers

Input:

```

var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setRowNumbered(true)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1", caption:"Age", width:40, type: "number"},
  {id:"col2", caption:"Full-time", width:90, type: "checkbox", width:90, type: "checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23]},
  {id:"row2",caption:'John',cells:[32],
    sub:[{id:"row21",caption:'Tom',cells:[24]},
          {id:"row22",caption:'Bob',cells:[25]}
    ]
  })
]).show(block)

```

To show row numbers

**Output:**

	Name	Age	Full-time
1	Jack		
2	John	32	<input type="checkbox"/>
2.1	Tom	24	<input type="checkbox"/>
2.2	Bob	25	<input type="checkbox"/>

Default format line numbers

**3.13.8.6. Custom row numbers****Input:**

```

var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setRowNumbered(true)
.setGridHandlerCaption("姓名")
.setHeader([
  {id:"col1", caption:"Age", width:40, type: "number"},
  {id:"col2", caption:"Full-time", width:90, type: "checkbox", width:90, type: "checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23]},
  {id:"row2",caption:'John',cells:[32],
    sub:[{id:"row21",caption:'Tom',cells:[24]},
          {id:"row22",caption:'Bob',cells:[25]}
    ]
  })
])
.setCustomFunction('getNumberedStr',function(no){
  var a=no.split('.');
  a[0]={1:'T',2:'IT'}[a[0]];
  return a.join('-')
})
).show(block)

```

Custom function

Output:

Name	Age	Full-time
I Jack	23	<input type="checkbox"/>
▼ II John	32	<input type="checkbox"/>
II-1 Tom	24	<input type="checkbox"/>
II-2 Bob	25	<input type="checkbox"/>

### 3.13.8.7. Alternate Row Colors

Input:

```
var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setAltRowsBg (true)
.setGridHandlerCaption("Name")
.setHeader([
  {id:"col1", caption:"Age", width:40, type: "number"},
  {id:"col2", caption:"Full-time", width:90, type: "checkbox"}
]).setRows([
  {id:"row1",caption:'Jack',cells:[23]},
  {id:"row2",caption:'John',cells:[32],
    sub:[{id:"row21",caption:'Tom',cells:[24]},
          {id:"row22",caption:'Bob',cells:[25]}
    ]}
]).show(block)
```

Sets alternate bg color

Output:

Name	Age	Full-time
Jack	23	<input type="checkbox"/>
▼ John	32	<input type="checkbox"/>
Tom	24	<input type="checkbox"/>
Bob	25	<input type="checkbox"/>

### 3.13.8.8. Group

Input:

```

var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setGridHandlerCaption("Name")
.setHeader([
    {id:"col1", caption:"Age", width:40, type: "number"},
    {id:"col2", caption:"Full-time", width:90, type: "checkbox"}
]).setRows([
    {id:"row1",caption:'Jack',cells:[23]},
    {id:"row2",caption:'John',cells:[32],group:true,
        sub:[{id:"row21",caption:'Tom',cells:[24]},
            {id:"row22",caption:'Bob',cells:[25]}
        ]
    }
]).show(block)
    
```

It's a group row

Name	Age	Full-time
Jack	22	
▼ John		
Tom	24	<input type="checkbox"/>
Bob	25	<input type="checkbox"/>

Group row

### 3.13.8.9. Preview and Summary region

Input:

```

var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setGridHandlerCaption("Name")
.setHeader([
    {id:"col1", caption:"Age", width:40, type: "number"},
    {id:"col2", caption:"Full-time", width:90, type: "checkbox"}
]).setRows([
    {id:"row1",caption:'Jack',cells:[23], preview: '<strong>Attention:</strong>',summary: '<em>Jack is the right one</em>' },
    {id:"row2",caption:'John',cells:[32], preview: 'John is OK',
        sub:[{id:"row21",caption:'Tom',cells:[24]},
            {id:"row22",caption:'Bob',cells:[25]}
        ]
    }
]).show(block)
    
```

preview

summary

Name	Age	Full-time
<b>Attention:</b>		<b>preview</b>
Jack	23	<input type="checkbox"/>
Jack is at the right one		
John is OK		
▶ John	32	<b>summary</b>

### 3.13.8.10. Update row dynamically

Input:

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80).setGridHandlerCaption("Name")
.setHeader([
{id:"col1", caption:"Age", width:40, type: "number"},{id:"col2", caption:"Full-time", width:90, type:
"checkbox"}
]).setRows([
{id:"row1",caption:'Jack',cells:[23, true]},
{id:"row2",caption:'John',cells:[32],
sub:[{id:"row21",caption:'Tom',cells:[24]},
{id:"row22",caption:'Box',cells:[25]}
]}
]).show(block)

_.asynRun(function(){
    tg.updateRow('row2', 'Jerry')
},1000)

_.asynRun(function(){
    tg.updateRow('row2',{caption:'Group', height:30, rowStyle:'background-color:#00ff00;', rowResizer:false,
group:true, preview:'preview', summary:'summary'})
},2000)

_.asynRun(function(){
    tg.updateRow('row1', {sub:[{ value:"Kate",cells:[24,true]}]})
},3000)
```

Updates row caption only

These properties are updatable

Updates all sub rows

Output:

Name	Age	Full-time
▶ Jack	23	<input checked="" type="checkbox"/>
preview		
▼ Group		
Tom	24	<input type="checkbox"/>
Box	25	<input type="checkbox"/>
summary		

### 3.13.9. Cell config

#### 3.13.9.1. Cell types

These types are support:

- | 'label': readonly text;
- | 'button': the button;
- | 'input': single line input;
- | 'textarea': multi lines input;
- | 'number': number only input;
- | 'currency': currency only input;
- | 'progress': the progress appearance;
- | 'combobox': combo input;
- | 'listbox': readonly combo input;
- | 'getter': for getting data;
- | 'helpinput': help data input;
- | 'cmdbox': command box input;
- | 'popbox': pop box input;
- | 'time': time input;
- | 'date': date input;
- | 'color': color input;

**Input:**



```

var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
    {id:"col1", caption:"Age", width:40, type: "number"},
    {id:"col2", caption:"Full-time", width:90, type: "label"}
]).setRows([
    {id:"row1",caption:'Jack',cells:[23, true]},
    {id:"row2",caption:'John',cells:[32, {value:false, type: "checkbox"}] }
]).show(block)
    
```

Setting in column header data

Setting in cell (has priority)

**Output:**

Name	Age	Full-time
Jack	23	true
John	32	<input type="checkbox"/>

### 3.13.9.2. Cell style

**Input:**

```

var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80)
.setColSortable(false)
.setGridHandlerCaption("Name")
.setHeader([
    {id:"col1", caption:"Age", width:40, type: "number"},
    {id:"col2", caption:"Full-time", width:90, type: "checkbox", cellStyle: "background-color:#00ff00;"}
]).setRows([
    {id:"row1",caption:'Jack', cells:[23, true]},
    {id:"row2",caption:'John', cellStyle: "background-color:#0000ff;",cells:[
        32,
        {value:false, cellStyle: "background-color:#ff0000;"}
    ] }
]).show(block)
    
```

**Output:**

Setting in row	Setting in column header
Age	Full-time
Jack	<input checked="" type="checkbox"/>
John	<input type="checkbox"/>
Setting in cell	

### 3.13.9.3. Update cell dynamically

**Input:**

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandlerWidth(80).setGridHandlerCaption("Name")
.setHeader([
{id:"col1", caption:"Age", width:40, type: "number"},{id:"col2", caption:"Full-time", width:90, type:
"checkbox"}
]).setRows([
{id:"row1",caption:'Jack',cells:[23, true]},
{id:"row2",caption:'John',cells:[32]}
]).show(block)

_.asRun(function(){
  tg.updateCellByRowCol('row2','col1', 18)
},1000)

_.asRun(function(){
  tg.updateCellByRowCol('row2','col1',{value:18,cellStyle:'background-color:#00ff00;'})
},2000)

_.asRun(function(){
  tg.updateCellByRowCol ('row2','col1', {type:"listbox",value:"20", editorListItems:["20","30","40"],
editable:true})
},3000)
```

Updates value only

These properties are updatable

Updates cell type

### 3.13.10. Editable

“editable” property in TreeGrid determines whether the TreeGrid is editable or not . Each column / row / cell has this property too. Those setting follow “Fine-grained priority principle”.

- ! TreeGrid’s editable =>false; cell’s editable=>true: only this cell is editable
- ! TreeGrid’s editable =>false; column header’s editable=>true: only this column is editable
- ! TreeGrid’s editable =>false; row’s editable=>true: only this row is editable
- ! TreeGrid’s editable =>true; cell’s editable=>true: only this cell is uneditable
- ! TreeGrid’s editable =>true; column header’s editable=>false: only this column is uneditable
- ! TreeGrid’s editable =>true; row’s editable=> false: only this row is uneditable

It should be noted that, cells in Row handler are uneditabe; cells with ‘label’ or ‘button’ type are uneditable.

### 3.13.10.1.Editable TreeGrid

**Input:**

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(true)
.setHeader([
{id:"col1",caption:"Name",width:60,type:'input'},
{id:"col2",caption:"Age",width:40,type:"number"},
{id:"col3",caption:"Gender",width:40,type:"listbox",editorListItems:[{id:'male',caption:'Male'},{id:'female',caption:'Female'}]}
]).setRows([
[Jack',23,{value:'male',caption:'Male'}],
[John',25,{value:'female',caption:'Female'}]
]).show(block)
```

Sets editable

List for editor

Value and caption

**Output:**

Name	Age	Gender
Jack	23	Male
John	25	Female

Name	Age	Gender
Jack	23	Male
John	25.00	Female

Name	Age	Gender
Jack	23	Male
John	25	Fen
		Male
		Female

### 3.13.10.2.Editable column

**Input:**

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(false)
.setHeader([
{id:"col1",caption:"Name",width:60,type:'input'},
{id:"col2",caption:"Age",width:40,type:"number"},
{id:"col3",caption:"Gender",width:40,type:"listbox",editorListItems:[{id:'male',caption:'Male'},{id:'female',caption:'Female'}]}
]).setRows([
[Jack',23,{value:'male',caption:'Male'}],
[John',25,{value:'female',caption:'Female'}]
]).show(block)
```

Sets uneditable

This column is editable

**Output:**

Name	Age	Gender
Jack	23	Male
John	25	Fem
		Male
		Female

### 3.13.10.3. Editable row

Input:

```
var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(false)
.setHeader([
{id:"col1", caption:"Name", width:60, type: 'input'},
{id:"col2", caption:"Age", width:40, type: "number"},
{id:"col3", caption:"Gender", width:40, type: "listbox", editorListItems:[{id:'male',caption:'Male'},{id:'female',caption:'Female'}]}
]).setRows([
[Jack',23, { value:'male',caption:'Male'}],
{cells:[John',25, { value:'female',caption:'Female'}],editable:true}
]).show(block)
```

Sets uneditable

This row is editable

Output:

Name	Age	Gender
Jack	23	Male
John	25.00	Female

### 3.13.10.4. Editable cell

Input:

```
var block=new linb.UI.Block({ width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(false)
.setHeader([
{id:"col1", caption:"Name", width:60, type: 'input'},
{id:"col2", caption:"Age", width:40, type: "number"},
{id:"col3", caption:"Gender", width:40, type: "listbox", editorListItems:[{id:'male',caption:'Male'},{id:'female',caption:'Female'}]}
]).setRows([
[Jack',23, { value:'male',caption:'Male'}],
[John',25, { value:'female',caption:'Female', editable:true} ]
]).show(block)
```

Sets uneditable

Only this cell is editable

Output:

Name	Age	Gender	
Jack	23	Male	
John	25	Fem	
		Male	
		Female	

### 3.13.10.5. The Editor

When a cell is set to editable, “active this cell” will show a corresponding editor. There are the following editors for different cell types.

- | ‘label’: readonly; no editor
- | ‘button’: readonly; no editor
- | ‘input’: normal linb.UI.Input control
- | ‘textarea’: multi lines linb.UI.Input control
- | ‘number’: number only linb.UI.Input control
- | ‘currency’: currency only linb.UI.Input control
- | ‘progress’: linb.UI.ComboInput control, spin
- | ‘combobox’: linb.UI.ComboInput control, combobox
- | ‘listbox’: linb.UI.ComboInput control, listbox
- | ‘getter’: linb.UI.ComboInput control, getter
- | ‘helpinput’: linb.UI.ComboInput control, helpinput
- | ‘cmdbox’: linb.UI.ComboInput control, cmdbox
- | ‘popbox’: linb.UI.ComboInput control, popbox
- | ‘time’: linb.UI.ComboInput control, time
- | ‘date’: linb.UI.ComboInput control, date
- | ‘color’: linb.UI.ComboInput control, color

**Input:**

```
var block=new linb.UI.Block({ width:300,height:340}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(true)
.setHeader(["Type", "Cell UI"]).setRows([
  { cells:['label',{ type:'label',value:'label'}]},
  { cells:['button',{ type:'button',value:'button'}]},
  { cells:['input',{ type:'input',value:'input'}]},
  { cells:['textarea',{ type:'textarea',value:'textarea'}]},
  { cells:['number',{ type:'number',value:'1.23'}]},
  { cells:['currency',{ type:'number',value:'21.23'}]},
  { cells:['progress',{ type:'progress',value:'0.85'}]},
  { cells:['combobox',{ type:'combobox',value:'combobox'}]},
  { cells:['listbox',{ type:'listbox',value:'listbox'}]},
  { cells:['getter',{ type:'getter',value:'getter'}]},
  { cells:['helpinput',{ type:'helpinput',value:'helpinput'}]},
  { cells:['cmdbox',{ type:'cmdbox',value:'cmdbox'}]},
  { cells:['popbox',{ type:'popbox',value:'popbox'}]},
  { cells:['time',{ type:'time',value:'12:08'}]},
  { cells:['date',{ type:'date',value:(new Date).getTime()}]},
  { cells:['color',{ type:'color',value:'#00ff00'}]}
]).show(block)
```

#### Output:

Type	Cell UI
label	label
button	button
input	input
textarea	textarea
number	1.23
progress	85%
combobox	combobox
listbox	listbox
getter	getter
helpinput	helpinput
cmdbox	cmdbox
popbox	popbox
timepicker	12:08
datepicker	7/29/2009
colorpicker	#00FF00

### 3.13.10.6. Custom the editor

#### Input:

```

var block=new linb.UI.Block({width:300,height:340}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(true)
.setHeader(["Type", "Cell UI "]).setRows([
  {cells:['email',{type:'email',value:'a@b.com'}]},
  {cells:['popwnd',{type:'popwnd',value:'value'}]}
])
.beforeIniEditor(function(profile, cell, cellNode){
  var t=cell.type;
  if(t=='email'){
    var editor = new linb.UI.Input({valueFormat:"^[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,4}$"});
    return editor;

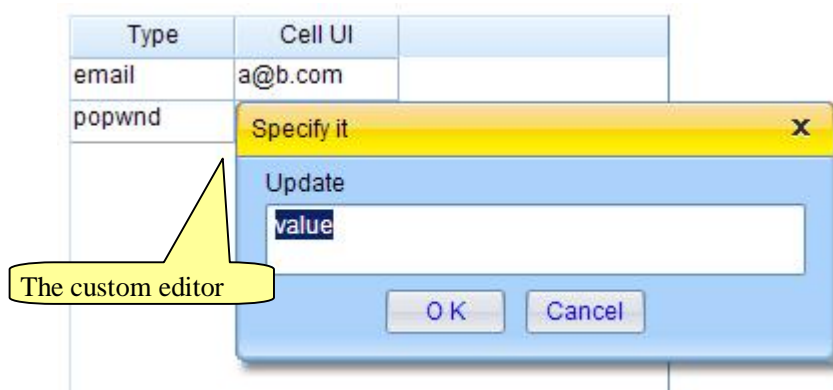
    if(t=='popwnd'){
      var dlg=linb.prompt('Specify it','Update',cell.value, function(value){
        if(cell.value!=value)
          profile.boxing().updateCell(cell, value);
      });
      dlg.getRoot().cssPos(cellNode.offset());
      return false;
    }
  })
}).show(block);

```

Return the custom editor  
Linb.U.Input or CombInput

Return false for advanced custom editor

**Output:**



### 3.13.11. Add/Remove rows

**Input:**

```
var block=new linb.UI.Block({width:240,height:200}).show();
var tg=new linb.UI.TreeGrid;
tg.setRowHandler(false)
.setEditable(true)
.setHeader([
{id:"col1",caption:"Name",width:60,type:'input'},
{id:"col2",caption:"Age",width:40,type:"number"}
]).setRows([
{id:'row1',cells:['Jack',23]},
{id:'row2',cells:['John',25]}
]).show(block);
```

```
_.asRun(function(){
  tg.insertRows([[]])
},1000);
```

Adds a empty row

```
_.asRun(function(){
  tg.insertRows(["Tom",30])
},2000);
```

Adds a new row

```
_.asRun(function(){
  tg.insertRows([id:'row3',cells:['Jerry',19]],["Mark",31])
},3000);
```

Adds two rows

```
_.asRun(function(){
  tg.removeRows('row1')
},4000);
```

Removes a row by id

```
_.asRun(function(){
  tg.removeRows(['row2','row3'])
},5000);
```

Removes two row by ids

```
_.asRun(function(){
  tg.insertRows([id:'row4',cells:['Jack',23]],null,null,true)
},6000);
```

Adds a row to the top

```
_.asRun(function(){
  tg.insertRows(['John',23],null,'row1',false)
},7000);
```

Adds a row next to 'row1'

#### NOTE

**chapter2/TGDynamic\index.html** is an overall example for ThreeGrid

**chapter2/TreeGrid.Paging\index.html** is another example for multi pages

## 3.14. Other standard controls

### 3.14.1. ProgressBar

Input:

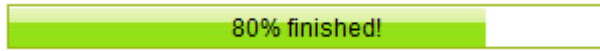


```
linb.create('ProgressBar')
.setCaptionTpl("{value}% finished!")
.setValue(80)
.show();
```

Sets text display template

percentage

Output:



### 3.14.2. Slider

Input:

```
linb.create('Slider')
.setPosition('relative')
.setSteps(100)
.setValue("20:50")
.show()

linb.create('Slider')
.setSteps(10)
.setType("vertical")
.setIsRange(false)
.setValue(2)
.setHeight(200)
.show();
```

Sets step to 100

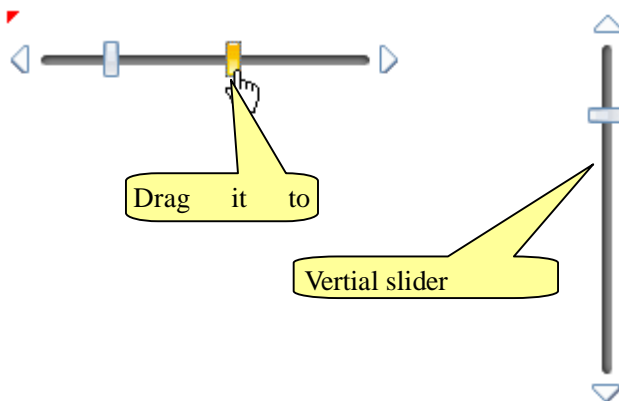
Sets the locations for the two slider

Sets step to 10

Vertical slider

A single slider

Output:



### 3.14.3. Image

```
linb.create("Image")
.setSrc("img/a.gif")
.afterLoad(function(){
    linb.message("The picture is loaded.");
})
.show();
```

The image was loaded successful

```
linb.create("Image")
.setSrc("img/b.gif")
.beforeLoad(function(){
    return false;
})
.show()
```

Cancel loading process

### 3.14.4. PageBar

Input:

```
var onclick=function(profile,page){
    profile.boxing().setPage(page);
};
// a PageBar
linb.create("PageBar")
.setValue("1:5:12")
.onClick(onclick)
.show();
// another PageBar
linb.create("PageBar")
.setValue("1:5:12")
.setTop(100)
.setCaption("")
.setPrevMark("<<")
.setNextMark(">>")
.setTextTpl("[ * ]")
.onClick(onclick)
.show();
```

Set current page

1.Min page; 2.current page; 3.max page

onClick event

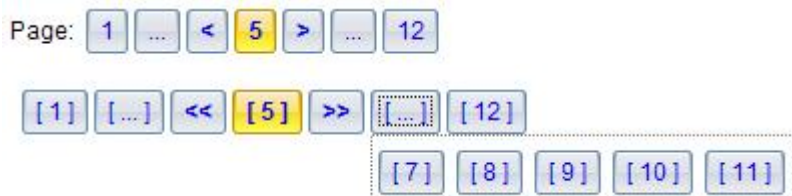
Caption label

Prev command label

Next command label

Page label templates, \* is the variable value

Output:



## Chapter 4. Data exchanging(Ajax)

jsLinb is a client-side solution, it can work with any backend (php, .Net, Java, python) or static HTML pages. Client-side and backend is completely decoupled. Client-side does not need to care what kind of technique is used in the backend. Client-side sends request to, and gets response from a given backend service(e.g. JSON service, REST service) .

There are three IO class in jsLinb:

- I linb.Ajax: An AJAX wrapper for xmlhttp object. It's features:
  - n Can only access the same domain by default;
  - n Works both synchronous and asynchronous;
  - n Works both 'get' and 'post' methods;
  - n Returns string.
- I linb.SAjax: An AJAX wrapper for "script tag". It's features:
  - n Cross domain;
  - n Asynchronous only;
  - n Can not post data;
  - n Returned content is packaged as javascript's Object

linb.SAjax send request data includes a "callback" parameter (default is "linb.SAjax.\$response"), and a "id" parameter ( the uniquely identify).

**Server's return data must be the following format:**

```
linb.SAjax.$response({id: "12483145855311"/*,data*/})
```

- I linb.IAjax: An AJAX wrapper for "iframe". It's features:
  - n Cross domain;
  - n Asynchronous only;
  - n Can update file;
  - n Works both 'get' and 'post' methods;
  - n Returned content is packaged as javascript's Object

linb.IAjax send request data includes an "id" parameter ( the uniquely identify).

**Server's return data must be the following format:**

```
<script type='text' id='json'>{"id": "12483161278402"/*,data*/}]]</script>
<script type='text/javascript'>
window.name=document.getElementById('json').innerHTML;
</script>
```

**"linb.request" function can choose an appropriate class from linb.Ajax, linb.SAjax or linb.IAjax automatically, according to requested domain, 'GET/POST' method and other information.**

### NOTE

Examples in this chapter works only as a http url, do not double-click directly to open.

## 4.1. Fiddler

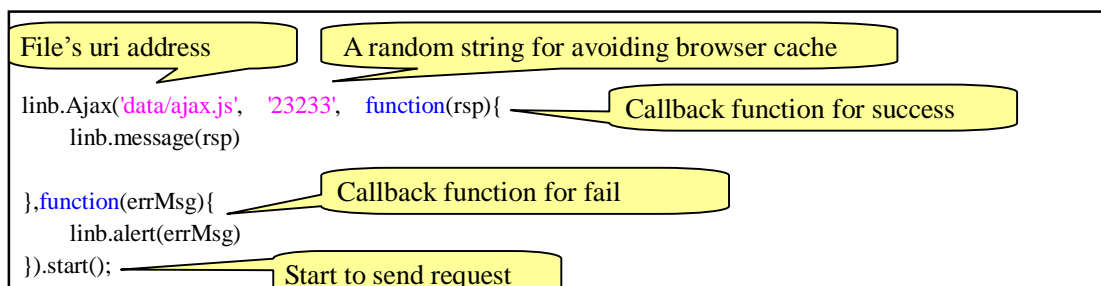
In order to understand the data exchanges process better, you need a tool like Fiddler to monitor network traffic.

Go to <http://www.fiddler2.com/fiddler2/> to get Fiddler.

Fiddler can configure IE proxy automatically, but if you are in firefox, chrome or opera, you need to configure **the** proxy by manual (Fiddler proxy: 127.0.0.1:8888). Of course, you can find some firefox proxy plug-ins to help you.

## 4.2. To get the contents of the file

linb.Ajax can get file contents easily.



In Fiddler:



## 4.3. Synchronous data exchange

Only `linb.Ajax` support synchronous data exchanging.

```

var url="chapter3/request.php";
linb.Ajax(url, {
    key:'test',
    para:{p1:'para 1'}
},function(rsp){
    linb.log(rsp);
},function(errMsg){
    linb.alert(errMsg)
}, null, {
    asy:false
}).start();

```

Request data

synchronous

**In fiddler:****The request:**

```
GET /jsLinb2.2/cases/chapter3/request.php?%7B%22key%22%3A%22test%22%2C%20%22para%22%3A%7B%22p1%22%3A%22para%201%22%7D%7D HTTP/1.1
```

**The response:**

Transformer	Headers	TextView	ImageView	HexView	WebView	JSON	Auth	Caching	Privacy	Raw
<pre>{   "data": [     {       "p1": "para 1",       "p2": "server_set",       "time": "2009-07-23 03:05:39",       "rand": "03-05-397jaso7bqm0f8op0rw"     }   ] }</pre>										

This is an asynchronous request:

```

var url="chapter3/request.php";
linb.Ajax(url, {
    key:'test',
    para:{p1:'para 1'}
},function(rsp){
    linb.log(rsp);
},function(errMsg){
    linb.alert(errMsg)
}).start();

```

## 4.4. Cross-domain

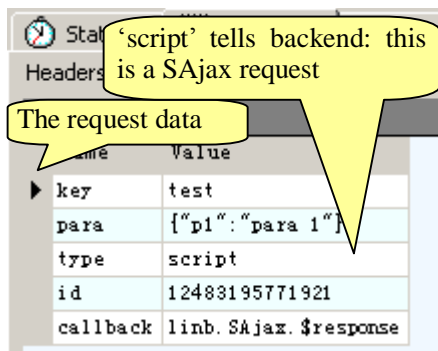
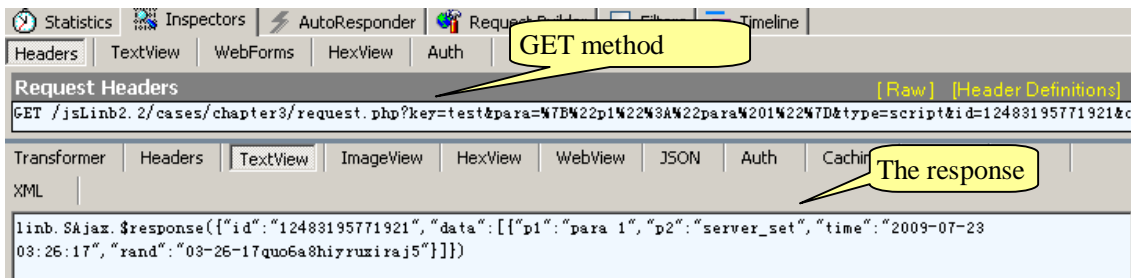
linb.SAjax and linb.IAjax can be used for calling Cross Domain Web Services. But only linb.IAjax can post data and upload file.

### 4.4.1. To monitor SAjax

**Code:**

```
var url="chapter3/request.php";
linb.SAjax(url, {
  key:'test',
  para:{p1:'para 1'}
},function(rsp){
  linb.log(rsp);
},function(errMsg){
  linb.alert(errMsg)
}).start();
```

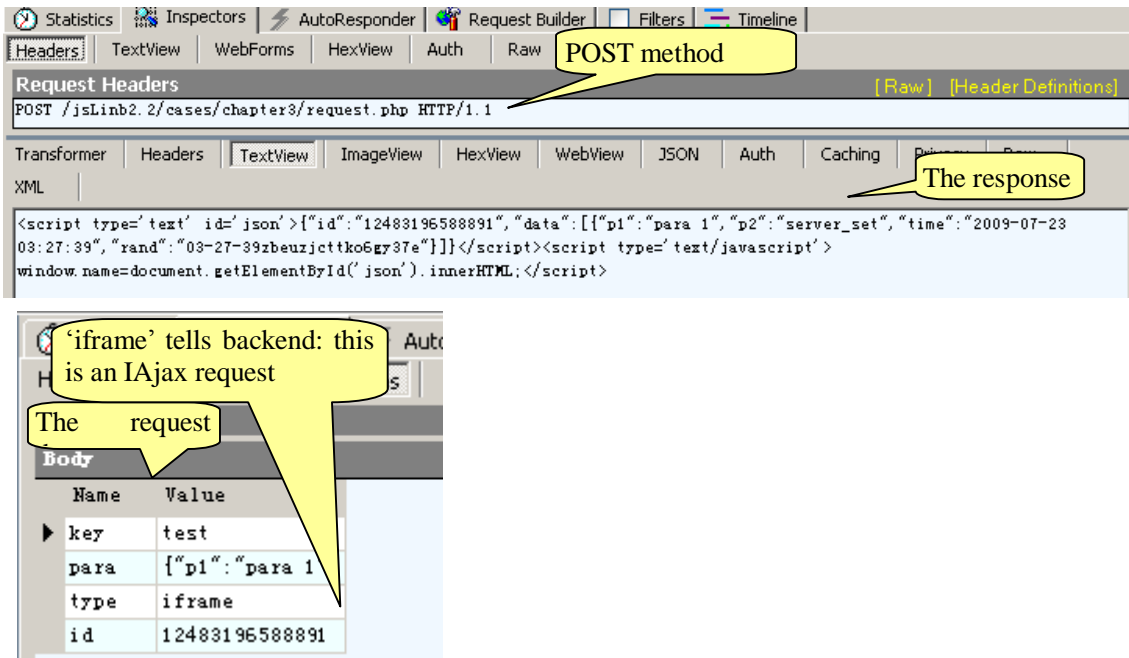
Request data

**In Fiddler:****4.4.2. To monitor IAjax****Code:**

```
var url="chapter3/request.php";
linb.IAjax(url, {
  key:'test',
  para:{p1:'para 1'}
},function(rsp){
  linb.log(rsp);
},function(errMsg){
  linb.alert(errMsg)
}).start();
```

Request data

**In Fiddler:**



By default, IAajax use “POST” method, you can specify method in options.

```
var url="chapter3/request.php";
linb.IAjax(url, {
    key:'test',
    para:{p1:'para 1'}
},function(rsp){
    linb.log(rsp);
},function(errMsg){
    linb.alert(errMsg)
},null,{
    method: 'get'
}).start();
```

Switch to GET method

## 4.5. File Upload

Only linb.UI.IAjax can upload file.

This code in this section is in "chapter3/upload/".

### 4.5.1. Selecting upload file with ComboInput

Sets ComboInput's type property to “upload”:

Select your file:

ComboInput

chapter3/upload/index.html

## 4.5.2. Upload by IAjax

```

Class('App', 'linb.Com',{
  Instance:{
    iniComponents:function(){
      // [[code created by jsLinb UI Builder
      var host=this, children=[], append=function(child){ children.push(child.get(0));

      append((new linb.UI.SLabel)
        .setHost(host,"slabel1")
        .setLeft(40)
        .setTop(44)
        .setCaption("Select your file: ")
      );

      append((new linb.UI.ComboInput)
        .setHost(host,"upload")
        .setLeft(140)
        .setTop(40)
        .setWidth(140)
        .setReadOnly(true)
        .setType("upload")
        .setValue("Select a file ...")
      );

      append((new linb.UI.SButton)
        .setHost(host,"sbutton3")
        .setLeft(290)
        .setTop(40)
        .setCaption("Upload it")
        .onClick("_sbutton3_onclick")
      );

      return children;
      // ]]code created by jsLinb UI Builder
    },
    _sbutton3_onclick:function (profile, e, src, value) {
      var file=this.upload.getUploadObj();
      if(file){
        linb.IAjax('./request.php',{key:'upload',para:{},file:file},function(rsp){
          linb.alert(rsp.data.message);
        },function(errMsg){
          linb.alert(errMsg)
        }).start();
      }
    }
  }
});

```

Created by Designer

Upload control

Getting file content

IAjax upload

Successful return

## 4.6. A request wrapper for real application

In practical applications, you can choose `linb.Ajax`, `linb.SAjax` and `linb.IAjax` according to the



actual situation. Usually, we will wrap a common function or class to handle all data interaction with the backend service. This is an example wrapper. Just for your reference.

```
request=function(service,
  requestData,
  onOK,
  onStart,
  onEnd,
  file
){
  _tryF(onStart);
  linb.observableRun(function(threadid){
    var options;
    if(file){
      requestData.file=file;
      options={method:'post'};
    }
    linb.request(service, requestData, function(rsp){
      if(rsp){
        if(!rsp.error)
          _tryF(onOK, [rsp]);
        else
          linb.pop(_serialize(rsp.error));
      }else{
        linb.pop(_serialize(rsp));
      }
      _tryF(onEnd);
    },function(rsp){
      linb.pop(_serialize(rsp));
      _tryF(onEnd);
    }, threadid,options)
  });
};
```

Service url address

Request data (key/value pairs)

Callback for successful call

File to upload

Callback for onStart and onEnd

Success

Fail

Fail

Fail

## 4.7. XML Data

If the server returns xml data, we can use linb.XML to convert the XML data into JSON data.

```
linb.Ajax('data/ajax.xml', "", function(rsp){
  alert (rsp)
  var obj = linb.XML.xml2json(linb.XML.parseXML(rsp));
  linb.pop(obj.message);
},function(errMsg){
  linb.alert(errMsg)
}).start();
```

XML to JSON

## 4.8. An overall example

The following is an overall example for data exchanging.

**how to use linb.absIO to interact with service**

	get	post	post file	cross domain	
				get	post
<b>linb.Ajax</b>	yes	yes	no	no	no
<b>linb.SAjax</b>	yes	no	no	yes (best for "return big data")	no
<b>linb.IAjax</b>	yes	yes	yes	yes	yes

Request data:

```
{
  key:'test',
  para:{
    p1:'client_set'
  }
}
```

use linb.Ajax 'get'   use linb.Ajax 'post'

use linb.SAjax   use linb.IAjax 'get'   use linb.IAjax 'post'

use linb.request function

Response data:

```
{"p1":"client_set", "p2":"server_set", "time":"2009-07-23 03:48:55", "rand":"03-48-55hjww63a7kenm3h7wr"}
```

Chapter3/io/index.html

## Chapter 5. Distributed UI

Sometimes, especially in larger applications, we maybe save a large “not frequently used” UI Class into a separate file. This file will not be loaded at the beginning.

When the application needs to show the UI, the program will automatically load code from the "separate file". It is so called "distributed UI". This "distributed UI" file can be in your server, or in different domain remote servers.

### 5.1. Shows dialog from a remote file

There's a file "Module3.js" in folder "[cookbook](#)\chapter4\distributed\App\js\", "Module3.js" includes a Class named "App.Module3". Let's try to call it.

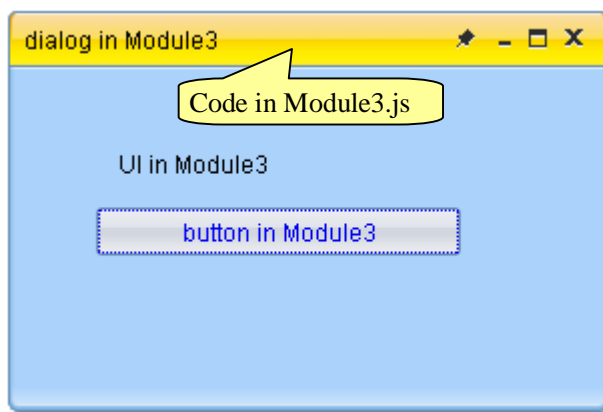
**Input:**

```
Namespace("App");  
linb.include("App.Module3",  
  linb.getPath("chapter4/distributed/App/js/", "Module3.js"),  
  function(){  
    var ins=new App.Module3();  
    ins.show();  
  },function(){  
    linb.alert("fail");  
  }  
);
```

Annotations:

- Namespace is "App"
- Dynamic asynchronous remote file loading
- Create instance and show it

**Output:**



And try to load code and create UI from a difference domain.

```

Namespace("App");
linb.include("App.Module3",
"http://www.linb.net/Samples/linb/app/distributed/App/js/Module3.js",
function(){
    var ins=new App.Module3();
    ins.show();
    },function(){
        linb.alert("fail");
    }
);

```

A difference domain

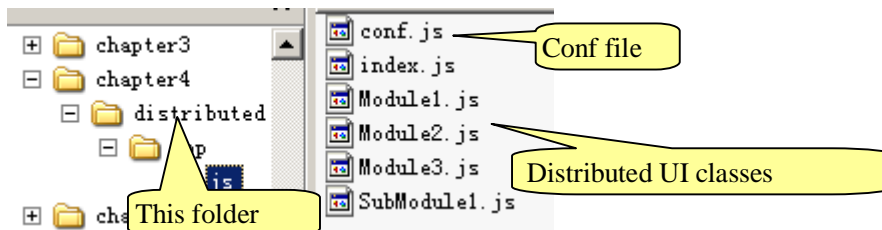
## 5.2.linb.Com and linb.ComFactory

In fact, most of the actual business applications will not load code from a foreign domain. From another perspective, most of "Distributed UI" files are put in the application directory.

In this case, we can use linb.Com and linb.ComFactory to load those "distributed UI". In order to use this approach, all those Classes must be derived from the linb.Com, named according to specified rules, and put into the specified directory.

linb.ComFactory implements a management mechanism for the linb.Com. It can follow a specified rule (finding file path from the class name) to load code from a remote file.

There 's an overall example in "chapter4/distributed", we can browse it for detail.



### 5.2.1. linb.ComFactory config

In conf.js:

```

CONF={ComFactoryProfile:
{
  module1:{
    cls:'App.Module1',
    children:{
      tag_SubModule1:'submodule1'
    }
  },
  submodule1:{
    cls:'App.SubModule1'
  }
}}

```

module1 is an "Aoo.Module1" Class

module1 has a child submodule1

submodule1 is a "Aoo.SubModule1" Class

Loading this configuration to linb.ComFactory:

```
linb.ComFactory.setProfile(CONF.ComFactoryProfile);
```

## 5.2.2. linb.Com.Load

In file index.html,

```
linb.Com.load ('App');
```

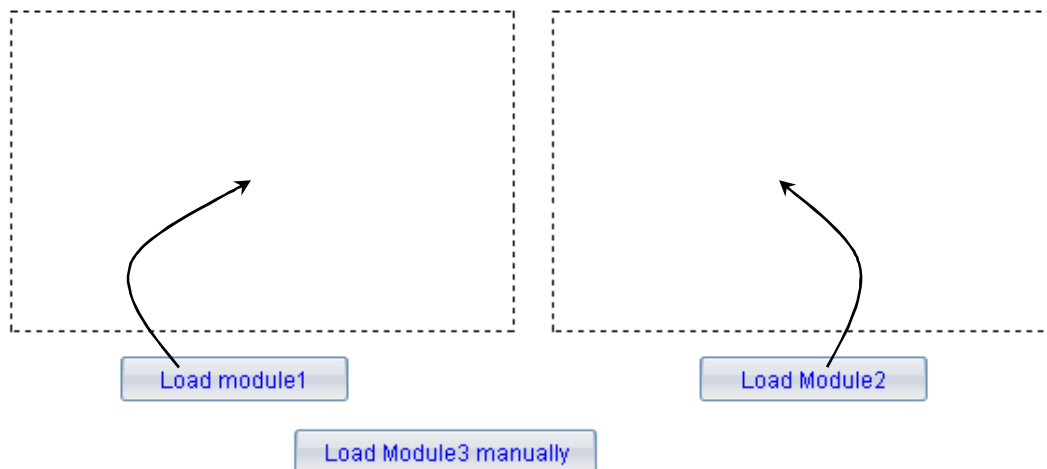
To load and show the first UI Class

The above code will try to find file named "index.js" from "distributed/App/js/", create an instance (new App), and show the instance to DOM.

### Output:

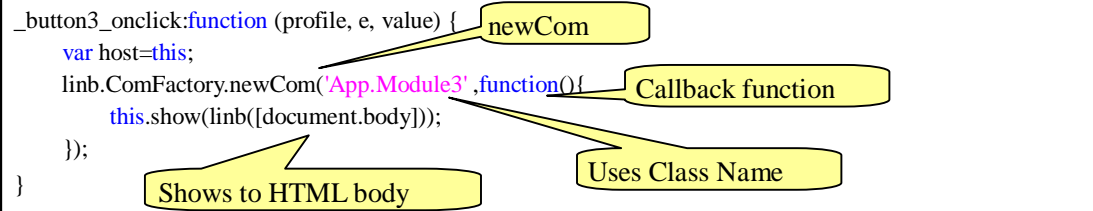
#### Loading code from outside dynamically!

Get Module code from out file on the fly, and append module UI to the current page



## 5.2.3. newCom and getCom

In index.js, onClick event for “Load module3 manually” button is:

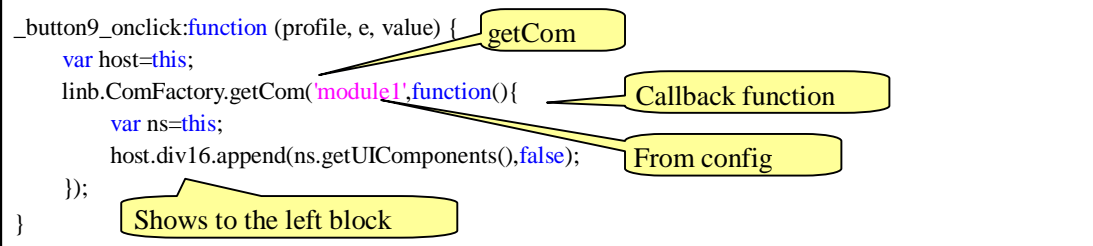


[linb.ComFactory.newCom(“App.Module3”.. ], will:

- ! find file “Module3.js” in “distributed/App/js/”
- ! load code from file “Module3.js” ;
- ! create new instance,;
- ! call the callback function.

Note: newCom use “Class Name” to load code.

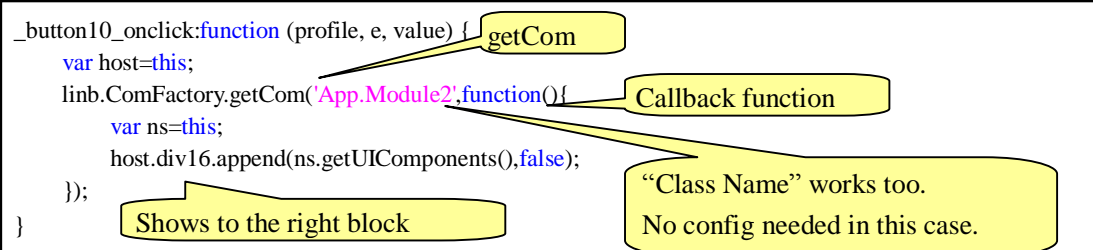
onClick event for “Load module1” button is:



[linb.ComFactory.newCom(“module1”.. ], will:

- ! find config from linb.ComFactory
- ! find file “Module1.js” in “distributed/App/js/”
- ! load code from file “Module1.js” ;
- ! create new instance,;
- ! call the callback function.

onClick event for “Load module2” button is:



By default, the instance created by "getCom" is singleton, and will be cached in inb.ComFactory.

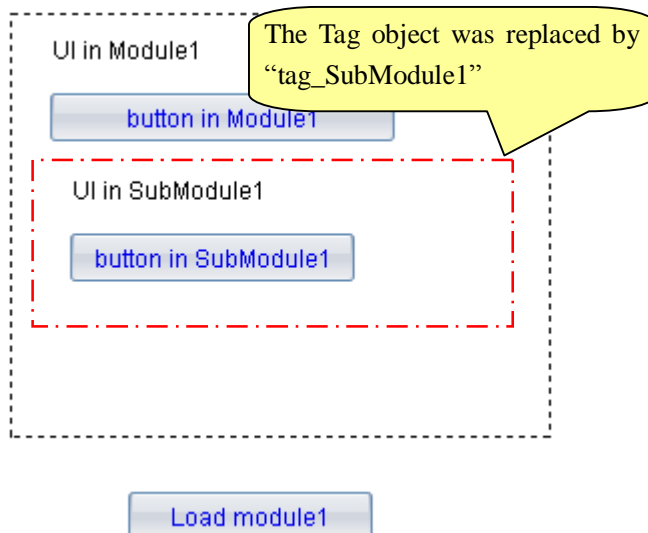
## 5.2.4. linb.UI.Tag

There's a linb.UI.Tag object in file Module1.js:

```
host.panelMain.append((new linb.UI.Tag)
    .host(host, "tag2")
    .setLeft(20)
    .setTop(70)
    .setWidth(218)
    .setHeight(98)
    .setTagKey("tag_SubModule1")
);
```

Module name in configuration

Here, this Tag object configures size and position properties for module “tag\_SubModule1”. When the instance of Module1 was created, according to the Tag object's info, system will load the “tag\_SubModule1” automatically, and set size and position properties to it. Then, system will replace the Tag object with “tag\_SubModule1” object, and destroy the Tag object.



## 5.2.5. Destroy com

Call com's **destroy()** function to destroy the Class instance;

Call **Class.destroy("class name")** to destroy the Class itself.

If you used “getCom(‘module name’)” to create an com instance, you have to call “linb.ComFactory.setCom ( ‘module name’, null )” to clear that cache.

## 5.2.6. If com exists in memory

If a com exists in memory already, we can call it directly:

```
linb('body').append(new App.Acom);
```

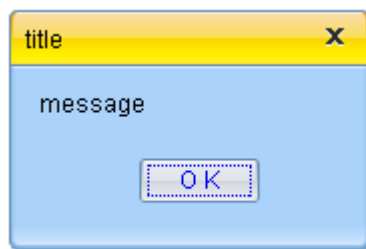
## Chapter 6. Some fundamental things

### 6.1. Pop-up window

#### 6.1.1. alert window

```
linb.alert('title','message',function(){  
    linb.message('You close this window!')  
}, 'OK', 50, 100);
```

Fired after user close the window



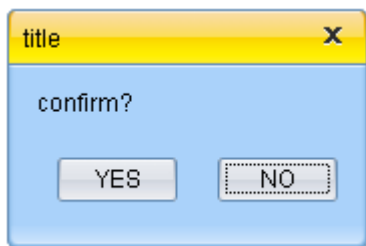
a

#### 6.1.2. confirm window

```
linb.confirm('title','confirm?',function(){  
    linb.message('You confirmed it')  
},  
function(type){  
    linb.message(" You didn't cofirm it -" + type)  
}, 'YES', 'NO', 50, 100);
```

Fired when user click "YES"

Fired when user click "NO" or click close button



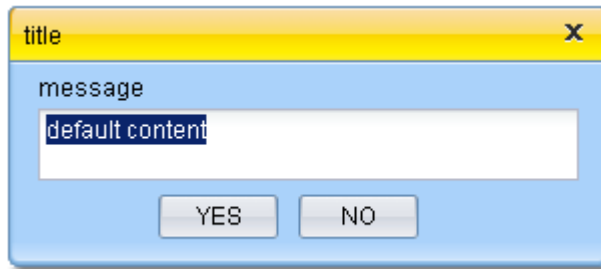


### 6.1.3. prompt window

```
linb.prompt('title', 'message', 'default content', function(msg){
    linb.message("You input - " + msg)
}, function(){
    linb.message(" You cancel it")
}, 'YES', 'NO', 50, 100);
```

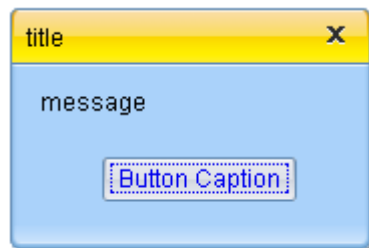
Fired when user click "YES"

Fired when user click "NO" or click close button



### 6.1.4. pop window

```
linb.pop('title', 'message', 'Button Caption', 50, 100);
```



## 6.2. Asynchronous execution

### 6.2.1. asyRun

\_.asyRun is a wrapper for setTimeout.

```
_.asyRun(function(arg1, arg2){
    linb.pop("this===linb:"+(this===linb), arg1+":"+arg2)
},
500,
['arg1', 'arg2'],
linb)
```

Function body

Delay 500 ms

parameters

scope

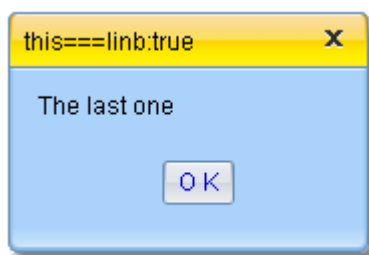
## 6.2.2. resetRun

`_asyRun` is a wrapper for `set timeout` too. But it has an unique id. When you set another function with the same id, the latter will cover the former.

```
_resetRun('key1',function(arg){
  linb.pop("this===linb:"+(this===linb), arg)
},500,['The previous one'],linb)

_resetRun('key1',function(arg){
  linb.pop("this===linb:"+(this===linb), arg)
},500,['The last one'],linb)
```

The latter one will be executed



## 6.3. Skin switcher

### 6.3.1. Switch skin for whole application

There are three system skins in jsLinb4.0: default, vista and aqua. You can use `linb.UI.setTheme` to switch the skin.

```
linb.UI.setTheme("vista")
```

Dynamically (no page refresh)

### 6.3.2. Change skin for a single control

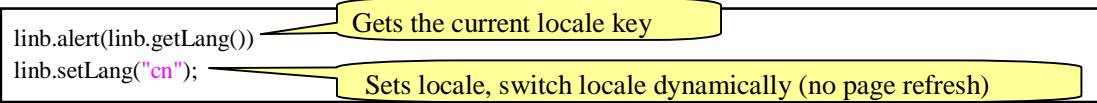
It's a fine-grained mechanism.

```
ctl_button.setTheme("custom")
```

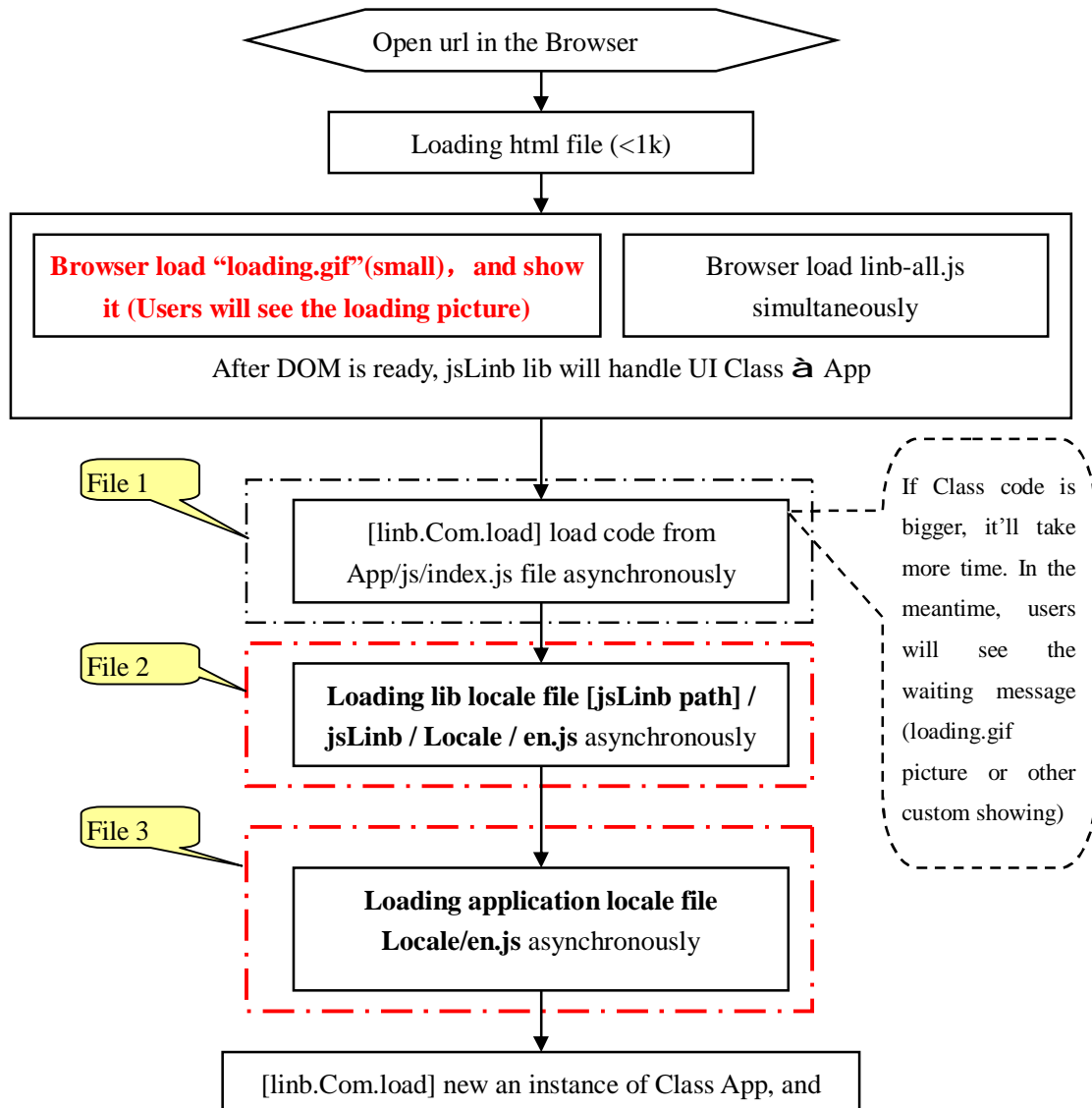
Only for "ctl\_button"

In this case, developer needs to define CSS class for this "custom".

## 6.4. Locale switcher



Example “chapter5\lang” loading process:

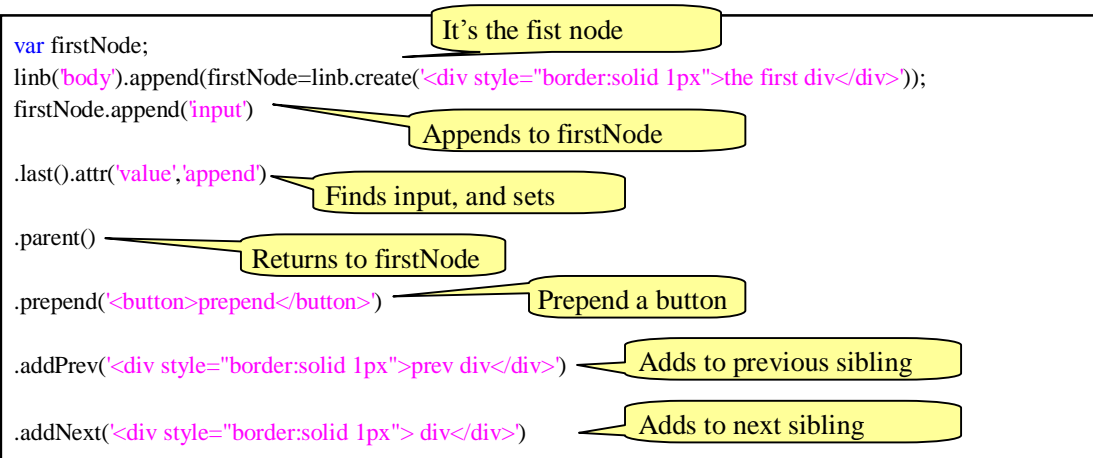


## 6.5. DOM Manipulation

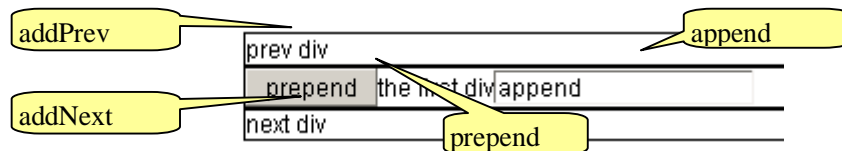
Class “linb.Dom” is a wrapper for cross-browser DOM Manipulation. It can: create / remove elements; manage elements’ attributes; manage elements’ CSS; manage elements’ events.

## 6.5.1. Node generation and insertion

Input:

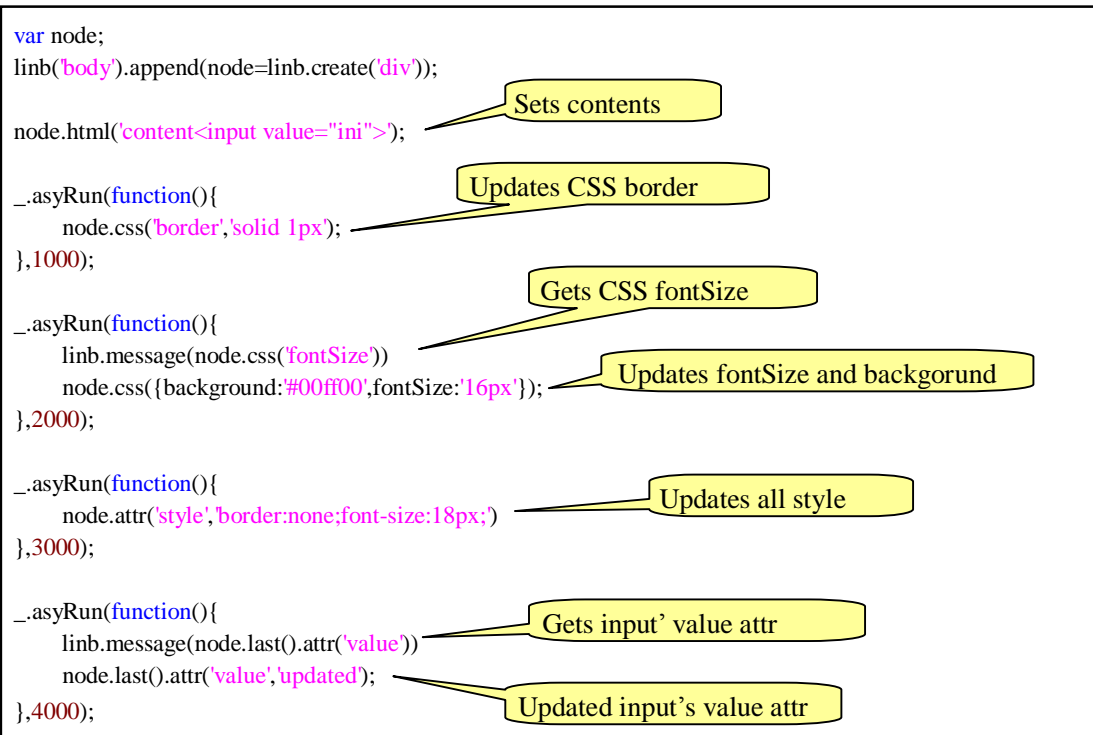


output:



## 6.5.2. Attributes and CSS

Input:



### 6.5.3. className

There are five function to handle CSS className:

- ! hasClass: Determines whether a specified class exists or not
- ! addClass: Adds classes to the current DOM nodes
- ! removeClass: Removes classes from the current DOM nodes
- ! replaceClass: Replaces classes for the current DOM nodes
- ! tagClass: Adds/Removes a tag to all classes of the current DOM nodes

```

var node;
linb('body').append(node=linb.create('div'));

_.asynRun(function(){
  node.addClass("cls1 cls2 cls3");
  linb.message(node.hasClass('cls2'));
  node.text(node.attr('className'));
},1000);

_.asynRun(function(){
  node.removeClass("cls2");
  node.text(node.attr('className'));
},2000);

_.asynRun(function(){
  node.replaceClass(/cls/g,"class");
  node.text(node.attr('className'));
},3000);

_.asynRun(function(){
  node.tagClass("-mouseover",true);
  node.text(node.attr('className'));
},4000);

_.asynRun(function(){
  node.tagClass("-mouseover",false);
  node.text(node.attr('className'));
},6000);

```

The diagram illustrates the following actions:

- Adds classes**: Points to `node.addClass("cls1 cls2 cls3");`
- Determines whether a class name exists or not**: Points to `linb.message(node.hasClass('cls2'));`
- Removes**: Points to `node.removeClass("cls2");`
- Modifies**: Points to `node.replaceClass(/cls/g,"class");`
- Adds tag**: Points to `node.tagClass("-mouseover",true);`
- Remove tag**: Points to `node.tagClass("-mouseover",false);`

### 6.5.4. Dom events

There are three groups of event functions are designed for a DOM event in jsLinb: [before-], [on-] and [after-].

- ! `linb(/**/).onClick([function], 'label')` => adds the [function] to [onclick] group;
- ! `linb(/**/).onClick([function])` => removes all event functions in [onclick] group, and adds the [function] to [onclick] group;
- ! `linb(/**/).onClick(null, 'label')` => removes the event function labeled with 'label' from the [onclick] group;
- ! `linb(/**/).onClick(null)` => removes all event functions in [onclick] group;

- I `linb(/**/).onClick(null,null,true) =>` removes all event functions in [beforeclick] group, [onclick] group and [afterclick] group;
- I `linb(/**/).onClick() =>` fire event, executes all event functions in [onclick] group in order. If any of those functions returns [false], the remaining functions will be ignored;
- I `linb(/**/).onClick(true) =>` fire event, executes all event functions in [beforeclick] group, [onclick] group and [afterclick] group in order;

```
var node;
linb('body').append(node=linb.create("<button>click me</button>"));

node.onClick(function(){
    alert('onClick');
    return false;
})
node.beforeClick(function(){
    alert('beforeClick');
})
node.afterClick(function(){
    alert('afterClick');
});
node.onClick(true);
_._.run(function(){
    node.onClick(null);
    node.onClick(true);
},2000);
```

Adds a onClick event

Adds a beforeClick event

Adds an afterClick event

Fires all click events. Since onClick returns false, afterClick will not be fired.

Removes onClick event;

Fires all click events. Since onClick was removed, afterClick will be fired this time.

## 6.5.5. Node Drag&Drop

Input:

```
var btn,div;
linb('body').append(btn=linb.create("<button>drag me</button>"))
.append(div=linb.create("<div style='position:absolute;left:100px;top:100px;border:solid 1px;width:150px; height:150px;display:block;'> drop here </button>"))

btn.draggable(true,{dragType:'icon','dragkey','dragdata'},
div.droppable(true,'dragkey')
.onDrop(function(){
    linb.alert(linb.DragDrop.getProfile().dragData);
});
```

Sets draggable

Sets droppable

onDrop event

Output:

drag me



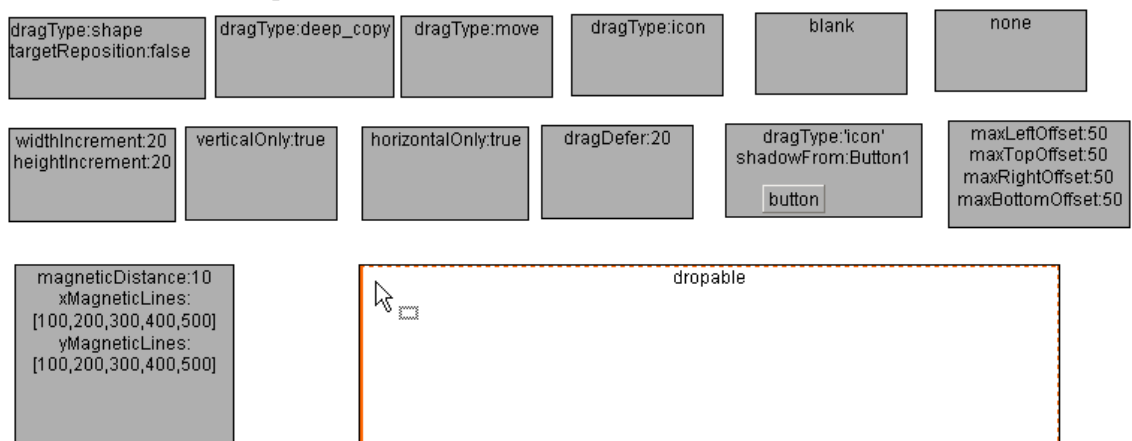
### 6.5.5.1. Drag&Drop profile

The “draggable” function’s second parameter is Drag&Drop profile object. It’s a key/value pairs. In dragging process, the Drag&Drop profile object can be got by `linb.DragDrop.getProfile()`. The profile object:

- | `dragType`: ‘move’, ‘copy’, ‘deep\_copy’, ‘shape’, ‘icon’, ‘blank’ or ‘none’, Default is ‘shape’;
- | `shadowFrom`: DOM element or `linb.Dom` Object. It’s valid when `dragType`==‘icon’;
- | `targetReposition`: Boolean, does dd reset the target position, Default is [true];
- | `dragIcon`: String, the drag icon image path, Default is [`linb.ini.path`+‘ondrag.gif’];
- | `magneticDistance`: Number, the magnetic distance, Default is 0;
- | `xMagneticLines`: Array of Number, the magnetic line values in horizontal dir, Default is [];
- | `yMagneticLines`: Array of Number, the magnetic line values in vertical dir, Default is [];
- | `widthIncrement`: Number, the width increment in horizontal dir, Default is 0;
- | `heightIncrement`: Number, the height increment in vertical dir, Default is 0;
- | `dragDefer`: Number, when [`linb.DragDrop.startDrag`] is called, the real drag action will be triggered after [`document.onmousemove`] runs [`dragDefer`] times, Default is 0;
- | `horizontalOnly`: Boolean, drag horizontal dir only, Default is [false];
- | `verticalOnly`: Boolean, drag vertical dir only, Default is [false];
- | `maxBottomOffset`: Number, the offset between [the restricted bottom] and [the current mouse Y], for mouse restricted region, Default is [null];
- | `maxLeftOffset`: Number, the offset between [the restricted left] and [the current mouse X], for mouse restricted region, Default is [null];
- | `maxRightOffset`: Number, the offset between [the restricted right] and [the current mouse X], for mouse restricted region, Default is [null];
- | `maxTopOffset`: Number, the offset between [the restricted top] and [the current mouse Y], for mouse restricted region, Default is [null];
- | `targetNode`: DOM element or `linb.Dom` Object, the drag target node;
- | `targetCSS`: Number, the drag target node’s CSS key/value Object, Default is [null];
- | `dragKey`: String, the drag key, Default is [null];
- | `dragData`: Object, the drag data, Default is [null];
- | `targetLeft`: Number, the drag target node’s CSS left, Default is [null];
- | `targetTop`: Number, the drag target node’s CSS top, Default is [null];
- | `targetWidth`: Number, the drag target node’s CSS width, Default is [null];

- | targetHeight: Number, the drag target node's CSS height, Default is [null];
- | targetOffsetParent: linb.Dom Object, the drag target node offsetParent node, Default is [null];
- | dragCursor: 'none', 'move', 'link', or 'add', the drag cursor key; [readonly]
- | x: Number, current X value of mouse; [readonly]
- | y: Number, current Y value of mouse; [readonly]
- | ox: Number, original X value of mouse; [readonly]
- | oy: Number, original Y value of mouse; [readonly]
- | curPos: {left:Number,top:Number}, current CSS pos of the dragging node [readonly]
- | offset: {x:Number,y:Number}, offset from now to origin [readonly]
- | isWorking: Boolean, is dd working or not? [readonly]
- | restrictedLeft: Number, the calculated restricted left value; [readonly]
- | restrictedRight: Number, the calculated restricted right value; [readonly]
- | restrictedTop: Number, the calculated restricted top value; [readonly]
- | restrictedBottom: Number, the calculated restricted bottom value; [readonly]
- | proxyNode: linb.Dom Object, the proxy Object; [readonly]
- | dropElement: String, the target drop element DOM id. [readonly]

There is an DD overall example in **chapter3/dd/ddProfile.html**.



### 6.5.5.2. Events in Drag&Drop

For that node in dragging,

- | onDragbegin
- | onDrag
- | onDragstop

For that droppable node,

- | onDragenter
- | onDragleave
- | onDragover
- | onDrop

**Input:**





```
var tpl=new linb.Template;  
tpl.setTemplate("<div [event]>{con}</div>")  
//tpl.setTemplate({root:"<div [event]>{con}</div>"})  
.setProperties({  
  con:'click me'  
})  
.setEvents({  
  root:{  
    onClick:function(){  
      linb.alert('Hi');  
    }  
  }  
})  
.show()
```

For event

Sets template

Sets properties

Sets events

Output:

click me



## 6.6.2. example 2

Input:

```

(tpl=new linb.Template)
.setTemplate({
  root : "<div style='width:200px;border:solid 1px;'><h3>{ head}</h3><ul>{ items}</ul><div
style='clear:bottom;'></div></div>"
  items: "<li [event] style='padding: 4px;border-top:dashed 1px;'><div><div><a href='{href}'><img
src='{src}'></a><p>{ price}</p></div></div><div><a
href='{href}'><h4>{ title}</h4></div><div>{ desc}</div></a></div></li>"
})
.setEvents({
  items:{
    onmouseover: function(profile,e,src){
      linb.use(src).css('backgroundColor','#EEE');
      //Tips
      var item=profile.getItem(src),
      tpl=new linb.Template({ "root": "<div style='text-align:center;border:solid
1px;background:#fff;'><h4>{ title}</h4></div><p>{ desc}</p>" },item),
      html=tpl.toHtml();
      linb.Tips.show(linb.Event.getPos(e),html);
    },
    onmouseout: function(profile,e,src){
      linb(src).css('backgroundColor','transparent');
      linb.Tips.hide();
    }
  }
})
.setProperties({
  head: "On sale products",
  items: [{ id: "a", href: "#", price: "$ 18.99", title: "product #0", desc: "product #0 is on sale now!" },
    { id: "b", href: "#", price: "$ 23.99", title: "product #1", desc: "product #1 is on sale now!" },
    { id: "c", href: "#", price: "$ 23.99", title: "product #2", desc: "product #2 is on sale now!" } ]
})
.show()

```

Sub template exists

Here, "root" key is a must

Sets events in "items"

Sets properties in "root"

Sets properties in "items"

Output:



### 6.6.3. A SButton based on linb.Template

“chapter5\SButton” is an example for creating a linb.UI.SButton like control based on linb.Template.

Output:

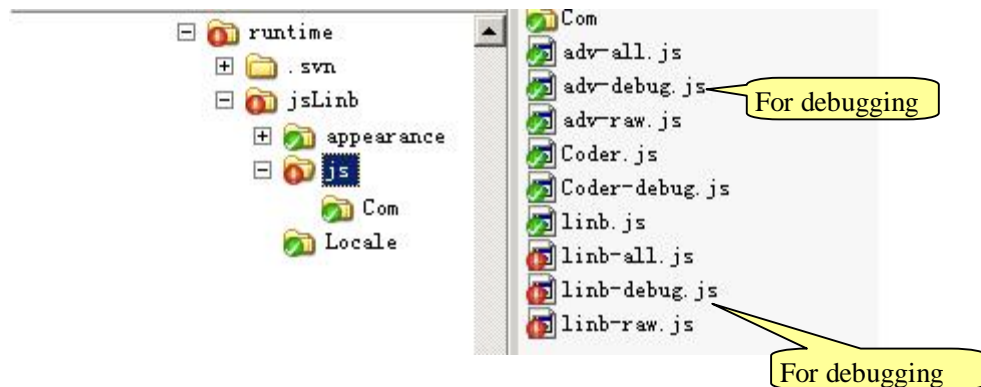
Template SButton 1    Template SButton 2    Template SButton 3



## 6.7. About Debugging

### 6.7.1. The code package for debugging

In folder “runtime/jsLinb/js/”, All files ending with “-debug.js” are for debugging purpose.



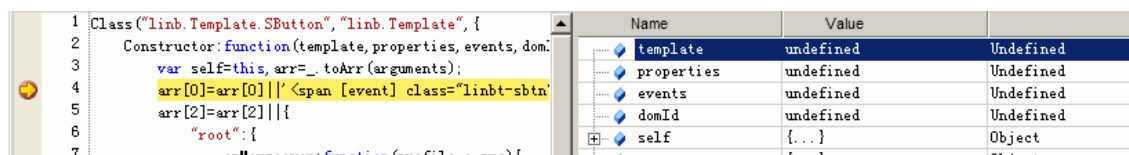
### 6.7.2. Debugging Tools

You can use Firebug in Firefox, developer tool in IE8, chrome or opera10 to debug JavaScript.

**FireBug:**



**Developer Tools in IE8:**



### 6.7.3. jsLinb Monitor Tools

jsLinb has a variable monitor tools. It's cross browser. Call `linb.log("xxx")` to show the monitor window:



## Chapter 7. Some typical issues

### 7.1. Layout

#### 7.1.1. Docking

Input:

```
linb.create('Block',{ dock:"top",
    height:80,html:'top'
}).show();
linb.create('Block',{ dock:"bottom",
    height:30,html:'bottom'
}).show();
linb.create('Block',{ dock:"left",
    width:150,html:'left'
}).show();
linb.create('Block',{ dock:"right",
    width:150,html:'right'
}).show();
linb.create('Block',{ dock:"fill",
    html:'main',
    dockMargin:{left:10,right:10,top:10,bottom:10}
}).show();
```

The diagram illustrates the docking of four blocks around a central 'main' area. The blocks are labeled 'top', 'bottom', 'left', and 'right'. The 'main' area is labeled 'The main area'. The 'dock' property is used to specify the docking location: 'top', 'bottom', 'left', 'right', and 'fill'. The 'dockMargin' property is used to specify the docking margin: {left:10,right:10,top:10,bottom:10}.

Output:



#### 7.1.2. linb.UI.Layout

Input:

```

var layout1=linb.create('Layout',{type:'vertical',
  items:[{
    pos:'before',
    id:'top',
    size:80
  },{
    pos:'after',
    id:'bottom',
    size:30
  }]
}).show();

linb.create('Layout',{type:'horizontal',
  items:[{
    pos:'before',
    id:'top',
    size:150
  },{
    pos:'after',
    id:'bottom',
    size:150
  }]
}).show(layout1);

```

Vertical layout

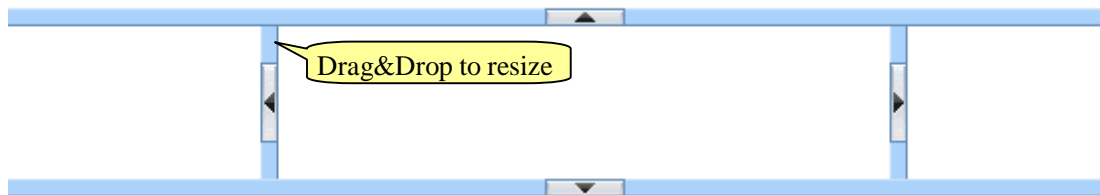
At top

At bottom

Horizontal layout

Left side

Right side



### 7.1.3. Relative Layout

**Input:**

```

linb.create('Pane',{position:'relative',
    width:"auto",height:80,html:"the top div"
})
.setCustomStyle({"KEY":"border:solid 1px #888"})
.show()

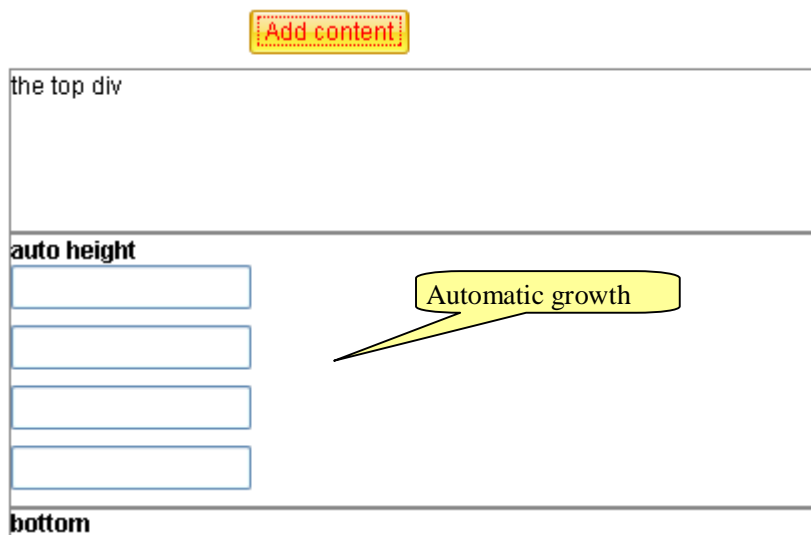
var pane=linb.create('Pane',{position:'relative',
    width:"auto",height:"auto",
    html:"<strong>auto height</strong>"
})
.setCustomStyle({"KEY":"border:solid 1px #888"})
.show()

linb.create('Pane',{position:'relative',
    width:"auto",height:100,
    html:"<strong>bottom</strong>"
})
.setCustomStyle({"KEY":"border:solid 1px #888"})
.show()

linb.create("SButton")
.setLeft(140)
.setTop(30)
.setCaption("Add content")
.onClick(function(){
    pane.append(linb.create("Pane").setPosition("relative").setHeight(30).append("Input"))
})
.show()

```

## Output:





## 7.2. UI Control's Drag&Drop

### 7.2.1. Drag&Drop control among containers

Input:

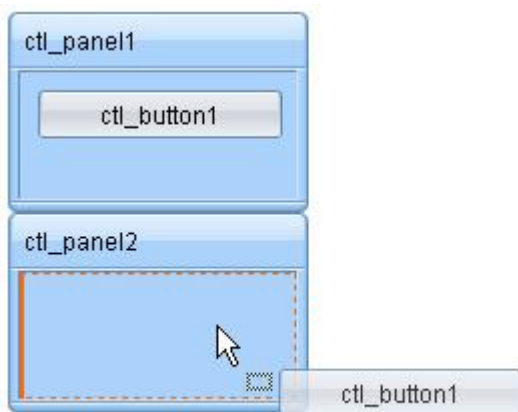
```
var panel1=linb.create('Panel',{position:'relative', dock:'none',width:150}).show();
var panel2=linb.create('Panel',{position:'relative', dock:'none',width:150}).show();
var btn=linb.create('Button',{left:10,top:10}).show(panel1);
var onDrop=function (profile, e, node, key, data) {
    var dd = linb.DragDrop.getProfile(), data = dd.dragData;
    if(data){
        var btn=linb.getObject(data);
        profile.boxing().append(btn.boxing());
    }
};
btn.draggable('iAny',btn.get(0).getId(),null,{shadowFrom:btn.getRoot()});
panel1.setDropKeys('iAny').onDrop(onDrop);
panel2.setDropKeys('iAny').onDrop(onDrop);
```

Sets onDrop function

Sets draggable

Sets droppable

Output:



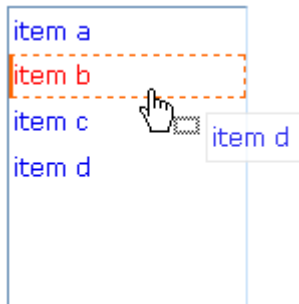
### 7.2.2. List sorting 1

Input:

```
linb.create("List",{
    items:["item a","item b","item c","item d"]
})
.setDragKey("list")
.setDropKeys("list")
.show()
```

Sets drag key and drop keys.

Output:



### 7.2.3. List sorting 2

Input:

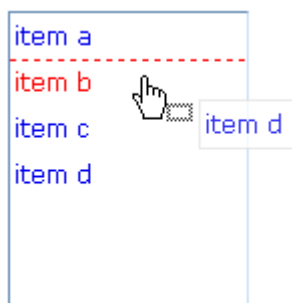
```
linb.create("List",{
  items:["item a","item b","item c","item d"]
})
.setDragKey("list")
.setDropKeys("list")
.onDropMarkShow(function(profile,e,src,key,data,item){
  if(item){
    linb.DragDrop.setDragIcon('move');
    linb.DragDrop.setDropFace(null);
    profile.getSubNodeById('ITEM',item.id).css('borderTop','dashed 1px');
    return false;
  }
})
.onDropMarkClear(function(profile,e,src,key,data,item){
  if(item){
    linb.DragDrop.setDragIcon('none');
    profile.getSubNodeById('ITEM',item.id).css('borderTop','');
    return false;
  }
})
.show()
```

Sets drag key and drop keys

Custom appearance

Restores appearance

Output:



## 7.3. Form

### 7.3.1. Form 1

**Input:**

```

Class.destroy('App');
Class('App', 'linb.Com',{
  Instance:{
    initComponents:function(){
      // [[code created by jsLinb UI Builder
      var host=this, children=[], append=function(child){ children.push(child.get(0));
      append((new linb.UI.SLabel)
        .setHost(host, "slabel1").setLeft(80).setTop(60).setWidth(44).setCaption("Name:") );
      append((new linb.UI.SLabel)
        .setHost(host, "slabel2").setLeft(80).setTop(90).setCaption("Age:").setWidth(44));
      append((new linb.UI.Input)
        .setHost(host, "iName").setLeft(130).setTop(60).setValueFormat("[^.*]").setValue("Jack"));
      append((new linb.UI.ComboInput)
        .setHost(host, "iAge").setLeft(130).setTop(90).setType("spin").setIncrement(1).setMin(20).s
      etMax(60).setValue("35"));
      append((new linb.UI.SCheckBox)
        .setHost(host, "cFull").setLeft(130).setTop(130).setCaption("Full time"));
      append((new linb.UI.SButton)
        .setHost(host, "submit").setLeft(130).setTop(170).setCaption("SUBMIT").onClick("_submit
      _onclick"));
      return children;
      // ]]code created by jsLinb UI Builder
    },
    _submit_onclick:function (profile, e, src, value) {
      if(!this.iName.checkValid()){
        linb.alert("You must specify Name");
        return;
      }
      var name=this.iName.updateValue().getValue(), age=this.iAge.updateValue().getValue(),
        full=this.cFull.updateValue().getValue();
      linb.alert(_serialize({ name:name,age:age,full:full}))
    }
  }
});
(new App).show();

```

Code created by Designer

event

Form validation

Collects data

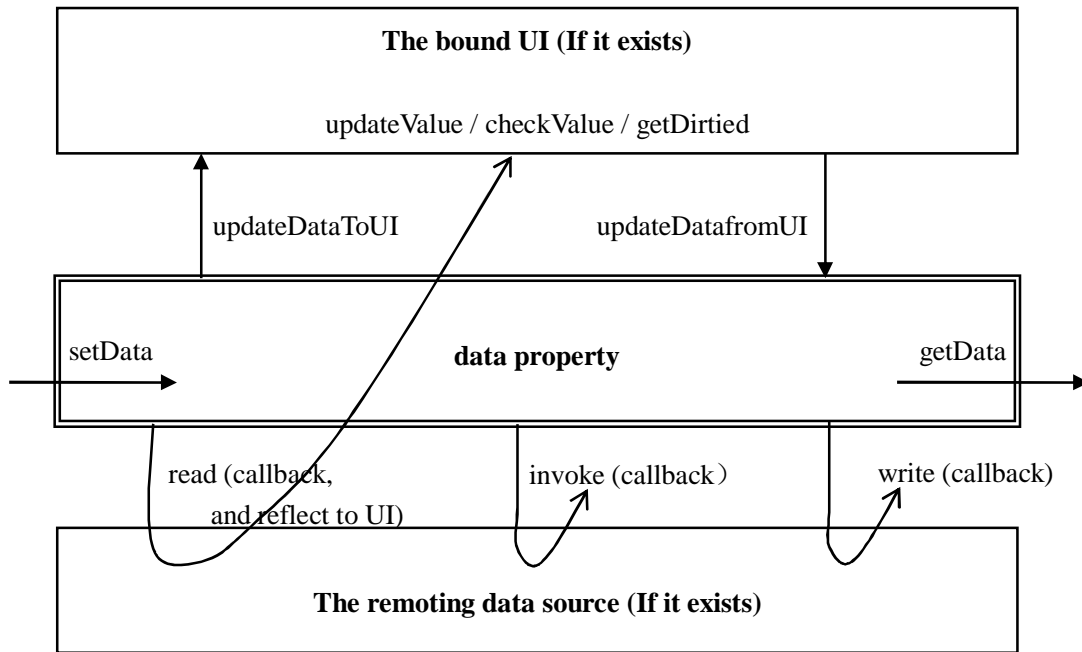
**Output:**



### 7.3.2. DataBinder

There are three types of data in DataBinder:

1. The inner data property:
  - setData function: to get the inner data property;
  - getData function: to set the inner data property;
2. The bound UI controls (if it exists):
  - updateValue function: to update the bound UI controls "UI value" to inner value, and removed dirty marks;
  - checkValue function: to check the bound UI controls;
  - getDirtied function: to get the dirtied values from the bound UI controls;
  - updateDataToUI function: to update the inner data to the bound UI controls;
  - updateDatafromUI function: to update the inner data from the bound UI controls;
3. The remoting data source (if it exists):
  - invoke function: invoke the remoting call;
  - write function: invoke the writing type remoting call;
  - read function: invoke the reading type remoting call; and update result data to the bound UI controls (updateDataToUI);



### Input:

```

Class.destroy('App');
Class('App', 'linb.Com',{
  Instance:{
    initComponents:function(){
      // [[code created by jsLinb UI Builder
      var host=this, children=[], append=function(child){ children.push(child);
      append((new linb.DataBinder).setHost(host, "binder").setName("binder"));
      append((new linb.UI.SLabel)
        .setHost(host, "slabel1").setLeft(80).setTop(60).setCaption("Name:").setWidth(44));
      append((new linb.UI.SLabel)
        .setHost(host, "slabel2").setLeft(80).setTop(90).setCaption("Age:").setWidth(44));
      append((new linb.UI.Input).setDataBinder("binder").setDataField("name")
        .setHost(host, "iName").setLeft(130).setTop(60).setValueFormat("[^.*]").setValue("Jack"));
      append((new linb.UI.ComboInput).setDataBinder("binder").setDataField("age")
        .setHost(host, "iAge").setLeft(130).setTop(90).setType("spin").setIncrement(1).setMin(20).setMax(60).setValue("35"));
      append((new linb.UI.SCheckBox).setDataBinder("binder").setDataField("isfull")
        .setHost(host, "cFull").setLeft(130).setTop(130).setCaption("Full time"));
      append((new linb.UI.SButton)
        .setHost(host, "submit").setLeft(130).setTop(170).setCaption("SUBMIT").onClick("_submit_onclick"));
      return children;
      // ]]code created by jsLinb UI Builder
    },
    _submit_onclick:function (profile, e, src, value){
      if(!this.binder.checkValid()){
        linb.alert('One or some invalid fields exists!');
        return;
      }
      linb.alert(_serialize(this.binder.updateDataFromUI().getData()))
    }
  }
});
(new App).show();
    
```

**Code created by Designer**

**Adds a DataBinder, sets name property**

**Sets dataBinder and dataField to each control**

**Form validation**

**Collects data**

## Output:

Name:

Age:

☒ Full time

×

{ "name": "Jack Lee", "age": "36", "isfull": true }

OK

Name:

Age:

☒ Full time

×

One or some invalid fields exits!

OK

## 7.4. Custom UI Styles

### 7.4.1. Custom only one instance only - 1

#### Input:

```
linb.CSS.remove("id","my_css");
linb.CSS.addStyleSheet(".linb-sbutton custom focus{ font-weight:bold;color:#ff0000;}", "my_css");
```

```
(new linb.UI.SButton)
.setCaption("Use setCustomClass ")
.setTheme ("custom")
.show();
```

Theme key words

Sets theme to this instance

Adds CSS. You should put those into a CSS file in your real application

#### Output:

Use setCustomStyle

## 7.4.2. Custom only one instance only - 2

Input:

```
(new linb.UI.SButton)
.setCaption("Use setCustomStyle")
.setCustomStyle({
  FOCUS:"font-weight:bold;color:#ff0000;"
})
.show();
```

Custom FOCUS node style

Output:

Use setCustomStyle

## 7.4.3. Custom only one instance only - 3

Input:

```
linb.CSS.remove("id","my_css");
linb.CSS.addStyleSheet(".my-class{ font-weight:bold;color:#ff0000;}", "my_css");

(new linb.UI.SButton)
.setCaption("Use setCustomClass ")
.setCustomClass({
  FOCUS:"my-class"
})
.show();
```

Adds CSS. You should put those into a CSS file in your real application

Custom FOCUS node className

Output:

Use setCustomClass

## 7.4.4. Custom only one instance only - 4

Input:

```
linb.CSS.remove("id","my_css");
linb.CSS.addStyleSheet("#myctrl1 .linb-sbutton-focus{font-weight:bold;color:#ff0000;}", "my_css");

(new linb.UI.SButton)
.setCaption("Use domId")
.setDomId("myctrl1")
.show();
```

Adds CSS. You should put those into a CSS file in your real application

Gives a domId

Output:

Use domId

## 7.4.5. Custom only one instance only - 5

**Input:**

```
(new linb.UI.SButton)
.setCaption("Use getSubNode and css ")
.onRender(function(profile){
  profile.getSubNode('FOCUS').css({
    fontWeight:'bold',
    color:'#ff0000'
  });
})
.show()
```

After it was rendered into DOM

**Output:**

Use getSubNode and css

## 7.4.6. Custom only one instance only - 6

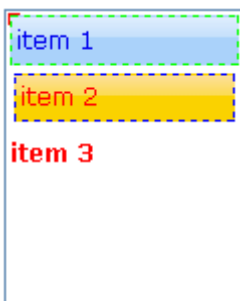
**Input:**

```
linb.CSS.remove("id","my_css");
linb.CSS.addStyleSheet(".my-listitem{font-weight:bold;color:#ff0000;}", "my_css");

linb.create('List',{items:[{
  id:"item 1",
  itemStyle:"border:dashed 1px #00ff00;margin:4px;"
},{
  id:"item 2",
  itemStyle:"border:dashed 1px #0000ff;margin:4px;"
},{
  id:"item 3",
  itemClass:"my-listitem"
}]}).show()
```

Adds CSS. You should put those into a CSS file in your real application

**Output:**





## 7.4.7. Custom style for an UI Class

**Input:**

```
linb.CSS.remove("id","my_css");
linb.CSS.addStyleSheet(".linb-sbutton-focus{font-weight:bold;color:#ff0000;}", "my_css");

(new linb.UI.SButton({position:'relative'})).show();
(new linb.UI.SButton({position:'relative'})).show();
(new linb.UI.SButton({position:'relative'})).show();
(new linb.UI.SButton({position:'relative'})).show();
(new linb.UI.SButton({position:'relative'})).show();
```

Adds CSS. You should put those into a CSS file in your real application

All instances were changed

**Output:**

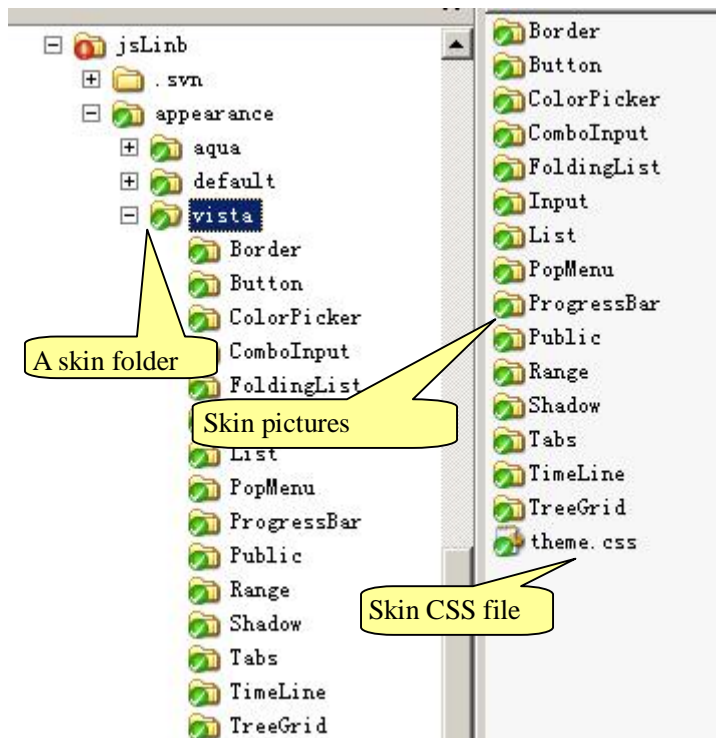
**sbutton7 sbutton8 sbutton9 sbutton10 sbutton11**

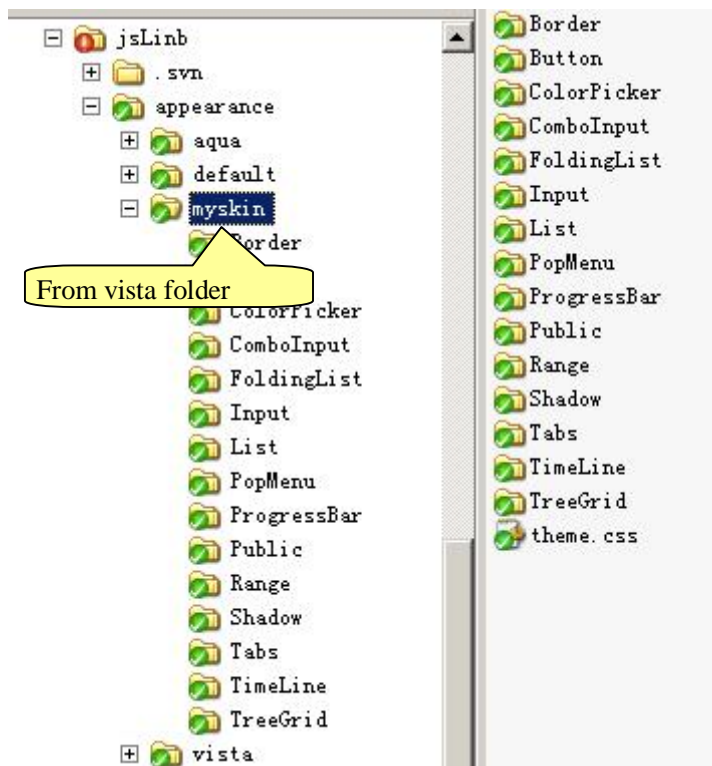
## 7.4.8. Custom style for all UI Class - skin

There are three system skins in jsLinb4.0: default, vista and aqua. You can use `linb.UI.setTheme` to switch the skin. You can also add your own custom skin easily. Only two steps:

### 7.4.8.1. First: Copy

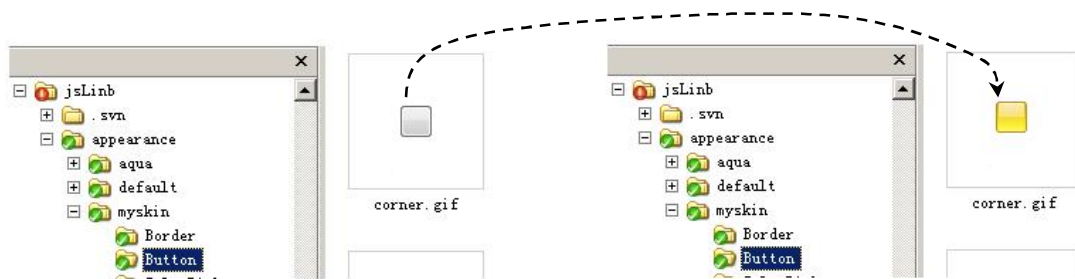
All skins are in “runtime/jsLinb/appearance”, you can create an new folder (e.g. ‘myskin’), and copy all directories and files in an existing skin folder to it.





### 7.4.8.2. Second: Little by little, modify pictures and CSS

For example, we modified corner.gif file in Button folder.



After that,

**Input:**

```
linb.create("Button").show();
_.asynRun(function(){
    linb.UI.setTheme('myskin')
},2000);
```

**Output:**



# The end

linb

2010-7-21

All rights reserved.