YAHOO!®

the yahoo user interface library

http://developer.yahoo.com/yui
http://yuiblog.com

YUI 2.4 Cheat Sheets

animation autocomplete
browser history manager
calendar charts color picker
connection manager container
css base css fonts css grids css
reset datasource datatable
dialog dom drag & drop
element event get imageloader
json logger menu module
overlay panel profiler rich text
editor selector simplediaolg
slider tabview treeview tooltip
yui compressor yui loader yui
test yahoo global object

Welcome to the Yahoo! User Interface Library (YUI)

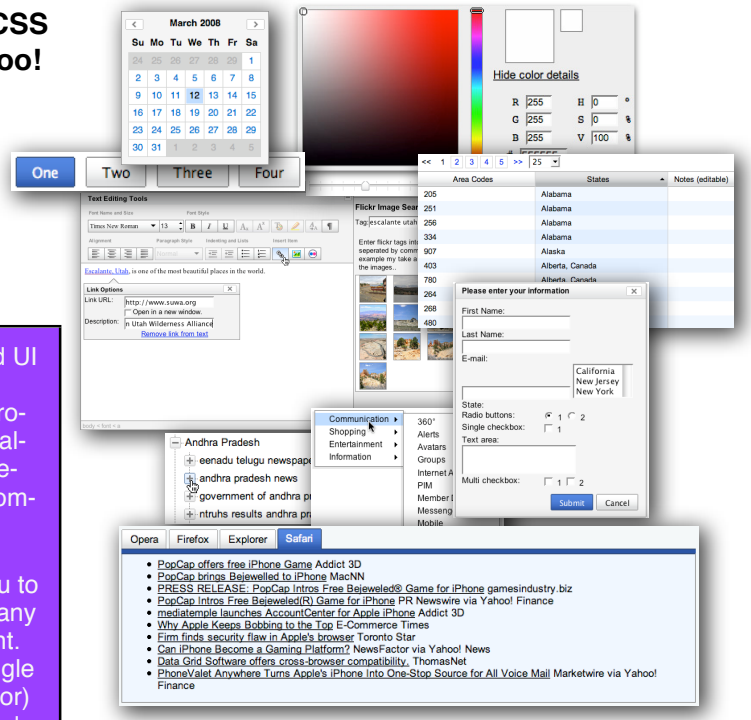
A free, open-source JavaScript and CSS library. The same one we use at Yahoo! every day. In February, 2006, Yahoo! opened its frontend library under a BSD license, making it freely available to all developers. Now the library that powers the world's most visited website can also power yours.

YUI provides a suite of JavaScript utilities and UI controls that help developers create rich, dynamic interfaces within the browser. It also provides four core CSS utilities that help to normalize your baseline styles and to build robust, semantic, CSS-driven page layouts utilizing a compact and flexible toolkit.

YUI is an á la carte library, one that allows you to pick and choose which features you want on any given page. The code is compact and efficient. If you enable server-side compression, no single YUI component (other than the Rich Text Editor) requires a download of more than 35KB — and most are much smaller than that.

Today's YUI consists of a CSS foundation, a core of three essential utilities (**YAHOO**, **Dom**, and **Event**), numerous other utilities that empower rich interactions (like Connection Manager, Drag & Drop, and Animation), and more than a dozen rich UI controls ranging from DHTML windowing to **AutoComplete** to a full **Rich Text Editor**.

YUI provides support for what we consider to be the "A-Grade" browsers: **Firefox**, **Opera**, **Safari** and **Internet Explorer**. You can count on all of our components working across the A-Grade; rare exceptions are documented as "known issues" and you can rely on Yahoo! to work with browser manufacturers to resolve known issues as promptly as possible.



Explore YUI:
YUI ships with nearly 200 examples. Download the library and explore the YUI core — YAHOO Global Object, the Dom Collection, and the Event Utility — and their examples. Then start exploring some of the other utilities like Animation, Connection Manager (for Ajax), and Drag & Drop.

Then take a look at the UI controls like AutoComplete and TabView and see how you can transform your web page with just a few lines of code and a few KB of library includes.

YUI Destinations:

YUI Website
http://developer.yahoo.com/yui

- Download YUI
- Read official documentation
- Explore searchable APIs
- Find user's guides for every component

YDN-JavaScript
http://tech.groups.yahoo.com/group/ydn-javascript/

- Ask questions
- Discuss YUI development and usage
- Share your implementation with other developers

SourceForge
http://sourceforge.net/projects/yui/

- Make feature requests
- File bug reports

YUIBlog
http://yuiblog.com

- Read articles and watch videos about YUI, JavaScript, CSS and more
- See YUI implementations in the wild
- Let others know about your own YUI implementations

Next:
We're working hard to improve YUI and better serve the YUI community. Please join YDN-JavaScript, follow the blog, and let the community know what you're doing with YUI.

YUI Library: Animation

20077-12-4 **v2.4**

Simple Use Case

```
myAnimObj = new YAHOO.util.Anim("myDiv", {width:
   {to: 100}, height: {to: 100}});
myAnimObj.animate();
```

Makes the HTML element whose id attribute is "myDiv" resize to a height and width of 100 pixels.

Constructor (YAHOO.util.Anim, ColorAnim, etc.)

```
YAHOO.util.Anim(str | element target, obj
   attributes[, num duration, obj easing]);
```

Arguments:
(1) **Element id or reference:** HTML ID or element reference for the element being animated.
(2) **Attributes object:** Defines the qualities being animated; see below.
(3) **Duration:** Approximate, in seconds.
(4) **Easing:** Reference to an easing effect, member of YAHOO.util.Easing.

Attributes Object

```
animAttributes = {
  animatedProperty: {
    by: 100, //start at current, change by this much
    to: 100, //start at current, go to this
    from: 100, //ignore current; start from this
    unit: 'em' //can be any legal numeric unit
  }
}
```

Note: Do not include `to` and `by` for the same animation property.

Animation Properties

Use Animation to apply gradual transitions to these properties*:

borderWidth	height
bottom	margin
fontSize	opacity
left	lineHeight
right	padding
top	width

*or to any other member of an element's style object that takes a numeric value

Dependencies

Animation requires the YAHOO Global Object, Dom Collection, and Event Utility.

Interesting Moments in Animation

Event	Fires...	Arguments
onStart	...when anim begins	
onTween	...on every frame	
onComplete	...when anim ends	[0] {frames: *total frames*, fps: *frames per second*, duration: *of animation in miliseconds*}

These are Custom Event members of YAHOO.util.Anim; use these by subscribing:
`myAnimInstance.onComplete.subscribe(myOnCompleteHandler);`

Using the Motion Subclass

Use the Motion subclass to define animations to/from a specific point, using (optional) bezier control points.

```
var attributes = {
  points: {
    to: [250, 450],
    control: [[100, 800], [-100, 200], [500, 500]]}};
var anim = new YAHOO.util.Motion(element,
   attributes, 1, YAHOO.util.Easing.easeIn);
```

Using the ColorAnim Subclass

Use the ColorAnim subclass to background, text or border colors.

```
var myAnim = new YAHOO.util.ColorAnim(element, {back
   groundColor: { to: '#dcdcdc' } });
myAnim.animate();
```

Using the Scroll Subclass

Use the Scroll subclass to animate horizontal or vertical scrolling of an overflowing page element.

```
var attributes = {
  scroll: { to: [220, 0] }
};
var anim = new YAHOO.util.Scroll(element,
   attributes, 1, YAHOO.util.Easing.easeOut);
```

Solutions

Subscribe to an API method:

```
myAnimObj = new YAHOO.util.Anim(element, {width:
   {to: 100}, height: {to: 100}});
myHandler = function(type, args) {
  someDiv.innerHTML = args[0].fps; //gets frames-
   per-second from the onComplete event}
myAnimObj.onComplete.subscribe(myHandler);
myAnimObj.animate();
```

YAHOO.util.Anim: Properties

attributes (obj)
currentFrame (int)
duration (num)
totalFrames (int)
useSeconds (b)

YAHOO.util.Anim: Methods

animate()
getEl()
getStartTime()
isAnimated()
stop(bFinish) if true, advances to last frame of animation

Easing Effects

Members of YAHOO.util.Easing

backBoth
backIn
backOut
bounceBoth
bounceIn
bounceOut
easeBoth
easeBothStrong
easeIn
easeInStrong
easeNone default; no easing
easeOut
easeOutStrong
elasticBoth
elasticIn
elasticOut

YUI Library: AutoComplete

2007-12-4c **v2.4**

Simple Use Case (AutoComplete)

Markup:

```
<div id="myAutoComplete">
  <input id="myInput" type="text">
  <div id="myContainer"></div>
</div>
```

Script:

```
var myAutoComp = new YAHOO.widget.AutoComplete ("myInput",
   "myContainer", myDataSource);
```

Instantiates a new AutoComplete object, `myAutoComp`, which queries an existing DataSource `myDataSource`.

Constructor (AutoComplete)

```
YAHOO.widget.AutoComplete(str | el ref input field, str |
   el ref suggestion container, obj DataSource instance[,
   obj configuration object]);
```

Arguments:
(1) **HTML element (string or object):** Text input or textarea element.
(2) **HTML element (string or object):** Suggestion container.
(3) **DataSource instance (obj):** An instantiated DataSource object; see below for DataSource types and constructor syntax.
(4) **Configuration object (object):** An optional object literal defines property values of an AutoComplete instance.

Constructors (DataSource Classes)

```
YAHOO.widget.DS_JSArray(arr js array[, obj configuration
   object]);
```

(1) **JS Array (array):** A JavaScript array of strings.
(2) **Configuration object (object):** An optional object literal defines property values of a DataSource instance.

```
YAHOO.widget.DS_JSFunction(str sURI, fn callback[, obj
   configuration object]);
```

(1) **JS Function (fn):** A JavaScript function which returns an array of strings.
(2) **Configuration object (object):** See above.

```
YAHOO.widget.DS_ScriptNode(str sURI, array schema[, obj
   configuration object]);
```

(1) **URI:** URI to the script location that will return data.
(2) **Schema (array):** Schema description of server response data.
(3) **Configuration object (object):** See above.

```
YAHOO.widget.DS_XHR(sr script uri, array schema[, obj
   configuration object]);
```

(1) **Script URI (string):** Server URI (local domains only – use a proxy for remote domains).
(2) **Schema (array):** Schema description of server response data.
(3) **Configuration object (object):** See above.

Interesting Moments (AutoComplete)

Event	Arguments (passed via *args* array)
textboxFocusEvent/ textboxBlurEvent	[0] AC instance
textboxKeyEvent	[0] AC instance; [1] keycode int
dataRequestEvent/ dataErrorEvent	[0] AC instance; [1] query string
dataReturnEvent	[0] AC instance; [1] query string; [2] results array
containerExpandEvent/ containerCollapseEvent	[0] AC instance
itemArrowToEvent/ itemArrowFromEvent	[0] AC instance; [1] element
itemMouseOverEvent/ itemMouseOutEvent	[0] AC instance; [1] element
itemSelectEvent	[0] AC instance; [1] element; [2] item data object or array
selectionEnforceEvent	[0] AC instance
unmatchedItemSelectEvent	[0] AC instance; [1] user selected string
typeAheadEvent	[0] AC instance; [1] query string; [2] prefill string

Subscribe to AutoComplete Custom Events on your AutoComplete instance:
`myAC.containerExpandEvent.subscribe(myFn[, myObj, bScope]);`

Simple Use Case (DataSource)

```
var myDataSource=new YAHOO.widget.DS_JSArray(["a","b"]);
```

Instantiates a new DataSource object, `myDataSource`, which is an array of strings that queries can be matched against.

Custom Formatting (AutoComplete)

The `formatResult` function gets passed (1) an array that holds result data and (2) the original query string. Override this function to return custom markup to populate each <li> element in the container.

```
myAC.formatResult = function(aResultItem, sQuery) {
   var sKey = aResultItem[0]; //the query match key
   // Additional data mapped by schema
   var attribute1 = aResultItem[1];
   var attribute2 = aResultItem[2];
   return (sKey + ": " + attribute1); }
```

Dependencies

AutoComplete requires the YAHOO Global Object, Dom, and Event; Connection Manager (for DS_XHR datasources), Animation (for animated opening of the suggestion container), Get and JSON (for DS_ScriptNode datasources) are optional.

YAHOO.widget. AutoComplete Properties:

animVert (b)
animHoriz (b)
animSpeed (int)
delimChar (char || array)
maxResultsDisplayed (int)
minQueryLength (int)
queryDelay (int)
autoHighlight (b)
highlightClassName (string)
prehighlightClass Name (string)
useShadow (b)
useIFrame (b)
forceSelection (b)
typeAhead (b)
allowBrowser Autocomplete (b)
alwaysShowContainer (b)

YAHOO.widget. DataSource Properties:

maxCacheEntries (int)
queryMatchCase (b)
queryMatchContains (b)
queryMatchSubset (b)

YAHOO.widget.DS_XHR Properties:

responseType (static constant) TYPE_FLAT, TYPE_JSON, or TYPE_XML
scriptQueryParam (string)
scriptQueryAppend (string)
responseStripAfter (string)
connTimeout (int)

YUI Library: Browser History Manager

2007-12-4 **v2.4**

Getting Started with Browser History Manager

1. Required Markup

The Browser History Manager requires the following in-page markup:

```
<iframe id="yui-history-iframe" src="asset"></iframe>
<input id="yui-history-field" type="hidden">
```

1. The asset loaded in the IFrame must be in the same domain as the page (use a relative path for the src attribute to make sure of that)
2. The asset loaded in the IFrame does not have to be an HTML document. It can be an image for example (if you use an image that you also happen to use in your page, you will avoid an unnecessary round-trip, which is always good for performance)
3. This markup should appear right after the opening `<body>` tag.

2. Module Registration and the `register` Method

Use the following code to register a module:

```
YAHOO.util.History.register(str module, str initial
state, fn callback[, obj associated object, b scope])
```

Arguments:

1. ***module***: Arbitrary, non empty string identifying the module.
2. ***Initial state***: Initial state of the module (corresponding to its *earliest* history entry). `YAHOO.util.History.getBookmarkedState` may be used to find out what this initial state is if the application was accessed via a bookmark.
3. ***callback***: Function that will be called whenever the Browser History Manager detects that the state of the specified module has changed. Use this function to update the module's UI accordingly.
4. ***associated object***: Object to which your callback will have access; often the callback's parent object.
5. ***scope***: Boolean – if true, the callback runs in the scope of the associated object.

3. Using the `onReady` Method

Once you've registered at least one module, you should use the Browser History Manager's `onReady` method. In your handler, you should initialize your module(s) based on their current state. Use the function `YAHOO.util.History.getCurrentState` to retrieve the current state of your module(s).

```
YAHOO.util.History.onReady(function () {
  var currentState =
   YAHOO.util.History.getCurrentState("module");
  // Update UI of module to match current state
});
```

4. Initializing the Browser History Manager

Before using the Browser History Manager, you must initialize it, passing in the id of the required HTML elements created in step 1:

```
YAHOO.util.History.initialize("yui-history-field",
   "yui-history-iframe");
```

Storing New History Entries: The `navigate` Method

Any registered module can create a new history entry at any time. Doing so creates a new "stop" to which the user can navigate to via the back/forward buttons and that can be bookmarked in the browser. You can create new history entries in your script using the `navigate` method.

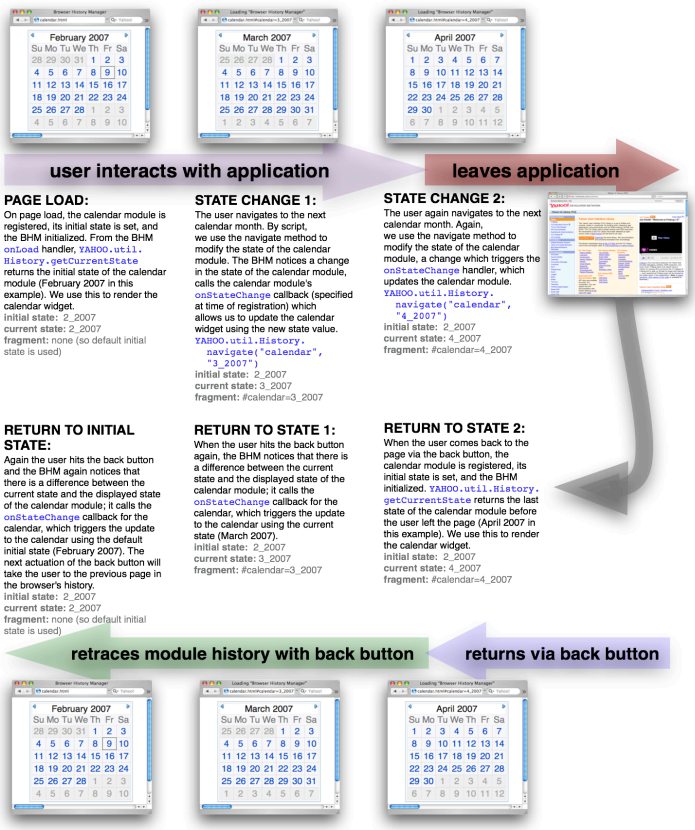
```
YAHOO.util.History.navigate(str module, str new state);
```

Arguments:

1. ***module***: Module identifier you used when you registered the module.
2. ***new state***: String representing the new state of the module.

Note: The navigate method returns a Boolean indicating whether the new state was successfully stored.
Note: The multiNavigate method allows you to change the state of several modules at once, creating a single history entry, whereas several calls to navigate would create several history entries.

A Sample Interaction



YAHOO.util.History Methods:

getBookmarkedState(str *module*) returns str *bookmarked state*
getCurrentState(str *module*) returns str *current state*
getQueryStringParameter(str *param name*[, str *query string*]) returns str *param value*
initialize(str *stateFieldId*, str *histFrameId*)
navigate(str *module*, str *state*) returns Boolean *success*
multiNavigate(arr *states*) returns Boolean *success*
register(str *module*, str *initial state*, fn *callback*[, *obj associated object*, *b scope*])

Dependencies

Browser History Manager requires the YAHOO Global Object and the Event Utility.

YUI Library: Button

2007-12-4 **v2.4**

Simple Use Cases: YAHOO.widget.Button

Create a Button instance using existing markup:

Markup:

```
<input type="button" id="mybutton" name="mybutton"
   value="Press Me!">
```

Script:

```
var oButton = new YAHOO.widget.Button("mybutton");
```

Create a Button instance using script alone:

```
var oButton =
        new YAHOO.widget.Button({ label:"Press Me!"});
```

Constructor: YAHOO.widget.Button

```
YAHOO.widget.Button(str|HTMLElement|obj element[, obj
   configuration object]);
```

Arguments:

(1) **element:** HTML ID or HTMLElement of existing markup to use when building Button. If neither, this is treated as the Configuration object.

(2) **configuration object:** JS object defining configuration properties for the Button instance. See Configuration section for full list.

Simple Use Cases: YAHOO.widget.ButtonGroup

Create a ButtonGroup instance using existing markup:

Markup:

```
<div id="mybuttongroup">
    <input type="radio" name="myfield" value="One">
    <input type="radio" name="myfield" value="Two">
    <input type="radio" name="myfield" value="Three">
</div>
```

Script:

```
var oButtonGroup = new
   YAHOO.widget.ButtonGroup("mybuttongroup");
```

Constructor: YAHOO.widget.ButtonGroup

```
YAHOO.widget.ButtonGroup(str|HTMLElement|obj element[,
   obj configuration object]);
```

Arguments:

(1) **element:** HTML ID or HTMLElement of existing markup to use when building ButtonGroup. If neither, this is treated as the Configuration object.

(2) **configuration object:** JS object defining configuration properties for the ButtonGroup instance. See Configuration section for full list.

Key Interesting Moments in Button

See online docs for a complete list of Button's Events.

focus	blur
click	

All Button events are Custom Events (see Event Utility docs); subscribe to these events using "addListener": (e.g. `oButton.addListener("click", fn);`).

Key Interesting Moments in ButtonGroup

See online docs for a complete list of ButtonGroup's Events.

Event:	Event Fields:
checkedButtonChange	type (s), prevValue (s), newValue (s)

All ButtonGroup events are Custom Events (see Event Utility docs); subscribe to these events using `addListener` (e.g. `oButtonGroup.addListener("checkedButtonChange", fn);`).

Key Button Configuration Options

See online docs for complete list of Buttion configuration options.

Option (type)	Default	Description
type (s)	"push"	String specifying the button's type. (Possible values are: "push," "link," "submit," "reset," "checkbox," "radio," "menu," and "split.")
label (s)	null	The button's text label or innerHTML.
name (s)	null	The name for the button.
value (o)	null	The value for the button.
checked (b)	false	Boolean indicating if the button is checked. (Applies only to buttons of type "radio" and "checkbox.")
disabled (b)	false	Boolean indicating if the button should be disabled. (Disabled buttons are dimmed and will not respond to user input or fire events.)
href (s)	null	The href for the button. (Applies only to buttons of type "link.")
menu (o)	null	HTMLElement id, array of YAHOO.widget.MenuItem configuration attributes, or YAHOO.widget.Menu instance. (Applies only to buttons of type "menu" and "split.")

Button options can be set in the constructor's second argument (eg, `{disabled: true}`) or at runtime via `set` (eg, `oButton.set("disabled", true);`).

Key ButtonGroup Configuration Options

See online docs for complete list of ButtonGroup configuration options.

Option (type)	Default	Description
name (s)	null	The name for the button group (will be applied to each button in the button group).
disabled (b)	false	Boolean indicating if the button group should be disabled. (Disabling the button group will disable each button in the button group.)
value (o)	null	Object specifying the value for the button.
checkedButton (o)	null	The checked button in the button group.

ButtonGroup options can be set in the constructor's second argument (eg, `{disabled: true}`) or at runtime via `set` (eg, `oButtonGroup.set("disabled", true);`).

YAHOO.widget.Button: Properties

CSS_CLASS_NAME
NODE_NAME

YAHOO.widget.Button: Methods

blur()
destroy()
focus()
getForm()
getMenu()
hasFocus()
isActive()
set([attr name], [attr value])
get([attr name])

YAHOO.widget. ButtonGroup: Properties

CSS_CLASS_NAME
NODE_NAME

YAHOO.widget. ButtonGroup: Methods

addButton(button)
addButtons([]buttons)
check(index)
destroy()
focus(index)
getButton(index)
getButtons()
getCount()
removeButton(index)
set([attr name], [attr value])
get([attr name])

Dependencies

Button requires the YAHOO Object, Event, Dom, and Element. **Optional:** Container Core and Menu.

YUI Library: Calendar and CalendarGroup

2007-12-4 **2.4**

Simple Use Case: YAHOO.widget.Calendar

Markup:

```
<div id="container"></div>
```

Script:

```
var myCal = new YAHOO.widget.Calendar("container");
myCal.render();
```

Creates a single page Calendar instance set to the current month.

Constructor: YAHOO.widget.Calendar, CalendarGroup

```
YAHOO.widget.Calendar([str calId,] str|HTMLElement
   container [,obj config]);

YAHOO.widget.CalendarGroup([str calId,] str|HTMLElement
   container [,obj config]);
```

calId: HTML ID for the new element created by the control to house the Calendar's DOM structure *(optional, as of 2.4.0)*. If not provided, the container's ID with a "_t" suffix is used.

container: HTML ID of (or a reference to) an existing but empty HTML element into which the new Calendar will be inserted.

config: Calendar configuration settings object *(optional)*.

Solutions

Render a single page calendar set displaying January 2008 with Spanish weekdays and month names:

```
myCal = new YAHOO.widget.Calendar("container",
   { pagedate: "1/2008",
     MONTHS_LONG:["Jenero","Febrero", … ,"Diciembre"],
     WEEKDAYS_SHORT:["Lu", "Ma", … , "Do"]});
myCal.render();
```

Add highlighting for Mexican holidays using CSS styles:

```
<style>
  .m1 .d1, .m1 .d6, .m2 .d5, .m2 .d14, .m2 .d24
      { background-color:yellow; }
</style>
```

There are two ways to configure options on your Calendar:

```
// 1. In the constructor, via an object literal:
myCal = new YAHOO.widget.Calendar("container",
                                  {pagedate: "5/2008"});
// 2. Via "setProperty" after rendering:
myCal.cfg.setProperty("pagedate", "5/2008");
```

Interesting Moments in Calendar, CalendarGroup

See online docs for complete list of Calendar and CalendarGroup events.

Event	Fires...	Arguments:
selectEvent	After a date is selected.	Array of date fields selected by the current action (e.g., `[[2008,8,1],[2008,8,2]]`)
deselectEvent	After a date has been deselected.	Array of date fields deselected by the current action (e.g., `[[2008,8,1],[2008,8,2]]`)
changePageEvent	When the Calendar navigates to a new month.	none
showEvent	When the Calendar's show() method is called.	none
hideEvent	When the Calendar's hide() method is called.	none

All Calendar events are Custom Events (see Event Utility docs); subscribe to these events using their subscribe method: `myCal.selectEvent.subscribe(fnMyHandler);`. Event-specific arguments, if present, will be passed in an array as the second argument to the event handler. Some events also have a "before" event which can be subscribed to

Calendar Options

Configure options using the constructor configuration object or *setProperty*, as described in "Solutions"

Calendar objects include several configurable options, including:

SHOW_WEEKDAYS HIDE_BLANK_WEEKS	SHOW_WEEK_ HEADER PAGES (CalendarGroup only)	MULTI_SELECT NAVIGATOR

Localizing Calendar and CalendarGroup

Calendar instances can be localized via configuration options, including:

MONTHS_SHORT MONTHS_LONG LOCALE_MONTHS	WEEKDAYS_1CHAR WEEKDAYS_SHORT LOCALE_WEEKDAYS	WEEKDAYS_MEDIUM WEEKDAYS_LONG

Applying localization properties requires the same syntax as any other properties in a Calendar's `cfg` object:

```
myCal = new YAHOO.widget.Calendar("calEl", "container");
myCal.cfg.setProperty("MONTHS_SHORT",
   ["Jan", "Feb", "Mär", "Apr", "Mai", "Jun", "Jul", "Aug",
    "Sep", "Okt", "Nov", "Dez"] );
myCal.cfg.setProperty("LOCALE_MONTHS", "short");
myCal.render();
```

Dependencies

Calendar requires the YAHOO Object, Event, and Dom.

YAHOO.widget.Calendar & CalendarGroup Properties

id (str)
cfg (Config)
the cal's configuration object
oDomContainer (el)
the cal's outer container

YAHOO.widget.Calendar & CalendarGroup Methods

Navigation:

addMonths(int)
addYears(int)
subtractMonths(int)
subtractYears(int)
setMonth(int)
setYear(int)
nextMonth()
nextYear()
previousMonth()
previousYear()

Rendering:

render()
renders current state to page
addRenderer
(s *dates*, fn *renderer*)
addWeekdayRenderer
(int *wkd*, fn *renderer*)
addMonthRenderer
(int *month*, fn *renderer*)
show()
hide()

Selection:

select(str *date*)
selectCell(int cellIdx)
deselect(str date)
deselectCell(int cellIdx)
deselectAll()
getSelectedDates()
returns array of JS date objects
clear()
removes all selected dates, resets month/year

Other:

reset()
resets calendar to original state
myCal.cfg.setProperty
(propName, propValue)
myCal.cfg.getProperty
(propName)

YUI Library: Charts

2007-12-4c **v2.4**

Simple Use Case: YAHOO.widget.LineChart

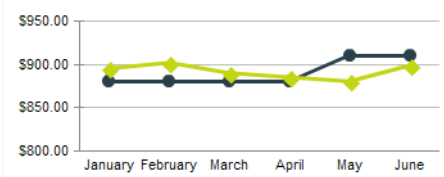
Markup:

```
<div id="myContainer">
  <-- For progressive enhancement, it's best to
   put the chart's data here in tabular or
   textual form to support viewers with Flash
   disabled. -->
</div>
```

Script:

```
var mySeriesDef = [
  {yField: "field1", displayName: "Series 1"},
  {yField: "field2", displayName: "Series 2"},
  ...];
var myDataSource =
  new YAHOO.util.DataSource([...]);
var myChart = new YAHOO.widget.LineChart(
  "myContainer", myDataSource, {} );
```

Creates a Chart instance from scratch.



Constructor: YAHOO.util.DataSource

```
YAHOO.util.DataSource(str|array|obj|HTMLFunction
  |HTMLTable live data[, obj config]);
```

Arguments:
(1) **live data:** Pointer to a set of data.
(2) **configuration object:** An optional object literal defines property values of a DataSource instance.

Constructor: YAHOO.widget.ColumnChart

```
YAHOO.widget.ColumnChart(str element, obj
   DataSource[, obj config]);
```

Arguments:
(1) **element:** HTML ID for a Chart container. May be empty or contain alternative content.
(2) **DataSource:** DataSource instance.
(3) **configuration object:** An optional object literal defines property values of a Chart instance.

Key Interesting Moments in Charts

See online docs for a complete list of Charts Events.

Event:	Arguments:
itemClickEvent, itemDoubleClickEvent, itemMouseOverEvent, itemMouseOutEvent	args.type (String) args.item (Object) args.index (Number) args.seriesIndex (Number) args.x (Number) args.y (Number)
itemDragStartEvent, itemDragEvent, itemDragUpdateEvent	args.type (String) args.item (Object) args.index (Number) args.seriesIndex (Number) args.x (Number) args.y (Number)

All Charts events are Custom Events (see Event Utility docs); subscribe to these events using "subscribe": (e.g. `myChart.subscribe("itemClickEvent", handler);`).

Key Charts Configuration Options

See online docs for complete list of Charts configuration options.

Option (type)	Default	Description
xField (s) yField (s)	null	The field used to access data to position items along the x or y axis.
request (s)	""	Request value to send to DataSource at instantiation for data to populate the chart.
series (a)	null	A series definition object.
dataTipFunction (s)	see docs	Object literal of pagination values.
xAxis (o) yAxis (o)	null	Custom axis objects.
polling (n)	null	The number of milliseconds between requests to the DataSource object for new data.
categoryNames (a)	null	If the DataSource does not contain a field that may be used with a category axis, an Array of Strings may be substituted.

Charts options can be set in the constructor's third argument (e.g., `{xField: "month"}`) or at runtime via set (e.g., `myChart.set("xField", "month");`).

Solutions

Specify a custom axis dimension if you don't want the chart to size the axis by default:

```
var axisWithMinimum = new
   YAHOO.widget.NumericAxis();
axisWithMinimum.minimum = 800;
myChart.set( "yAxis", axisWithMinimum );
```

YAHOO.widget.Axis

Properties

type
orientation
reverse
labelFunction
hideOverlappingLabels

YAHOO.widget.NumericAxis

Properties

minimum
maximum
majorUnit
minorUnit
snapToUnits
alwaysShowZero
scale

Note: Refer to online documentation for a full list of Axis properties.

YAHOO.widget.Series

Properties

type
displayName

YAHOO.widget.CartesianSeries

Properties

xField
yField

YAHOO.widget.PieSeries

Properties

dataField
categoryField

Note: Refer to online documentation for a full list of Series properties.

Dependencies

Charts require the YAHOO Global Object, Event Utility, Dom Collection, Element Utility, JSON Utility and DataSource Utility. **Note:** *On the client, Charts requires Flash Player 9.0.45 or later.*

YUI Library: Color Picker Control [beta]

2007-12-4b **v2.4**

Simple Use Case: YAHOO.widget.ColorPicker

Markup:

```
<div id="picker"></div>
```

Script:

```
var oPicker = new
     YAHOO.widget.ColorPicker("picker", {
          showhsvcontrols: true,
          showhexcontrols: true
     );
```

Creates a Color Picker instance from script; the Color Picker's DOM structure is created in the element whose ID is "picker".

Constructor: YAHOO.widget.ColorPicker

```
YAHOO.widget.ColorPicker(str ID|HTMLElement
   element[, obj config]);
```

Arguments:

(1) **Element:** HTML ID or HTMLElement for container element; this is the location in the DOM where the Color Picker Control will be inserted.

(2) **Configuration Object:** JS object defining configuration properties for the Color Picker instance. See Configuration Options section for full list.

Setting the Color Picker's Value

You can set the Color Picker's current value by script any time after instantiation using `setValue`:

```
oPicker.setValue([255, 255, 255], false);
```

The second argument is a boolean; if true, it suppresses the rgbChange event that would otherwise be fired when the value is changed.

Interesting Moment in Color Picker

This is the full list of Custom Events exposed in the Color Picker API.

Event:	Event Fields:
rgbChange	`newValue` (arr) and `oldValue` (arr), in both cases an array of RGB values; `type` (s), "rgbChange".

All Color Picker events are Custom Events (see Event Utility docs); subscribe to these events using "addListener" or "on": (e.g. `oPicker.on('rgbChange', fn);`).

Key Color Picker Configuration Options

See online docs for complete list of Color Picker configuration options.

Option (type)	Default	Description
showcontrols (b)	true	Hide/show the entire set of controls.
showhexcontrols (b)	true	Hide/show the hex controls.
showhexsummary (b)	true	Hide/show the hexadecimal summary information.
showhsvcontrols (b)	false	Hide/show the HSV controls.
showrgbcontrols (b)	true	Hide/show the RGB controls.
showwebsafe (b)	true	Hide/show the websafe swatch.
images (o)	Default images served by Y! servers	***Object Members:*** PICKER_THUMB: Image representing the draggable thumb for the color region slider. HUE_THUMB: Image representing the draggable thumb for the HSV slider.

Color Picker options can be set in the constructor's second argument (eg, `{showwebsafe: false}`) or at runtime via `set` (eg, `oPicker.set("showwebsafe", false);`).

Solutions

Listen for the `rgbChange` event and make use of the event's fields:

```
var oPicker = new
     YAHOO.widget.ColorPicker("container");

//a listener for logging RGB color changes;
//this will only be visible if logger is enabled:
var onRgbChange = function(o) {
   /*o is an object
       { newValue: (array of R, G, B values),
         prevValue: (array of R, G, B values),
         type: "rgbChange"
       }
   */
   YAHOO.log("The new color value is " + o.newValue,
   "info", "example");
}

//subscribe to the rgbChange event;
oPicker.on("rgbChange", onRgbChange);
```

YAHOO.widget.ColorPicker: Methods

getElement(id | key *element*)
returns the requested element; see API docs for keyset…pass in YUI_PICKER to return the host element for the instance

setValue(arr *RGB*[, b *silent*])
sets the currently selected color

YAHOO.widget.ColorPicker: Form Fields

When a form wrapping a Color Picker instance is submitted, the following form fields are included:

yui-picker-r	RGB red
yui-picker-g	RGB green
yui-picker-b	RGB bluee
yui-picker-h	HSV hue
yui-picker-s	HSV saturation
yui-picker-v	HSV value/ brightness
yui-picker-hex	Hex triplet for current color

Dependencies

Color Picker requires Yahoo, Dom, Event, Element, Drag & Drop, and Slider. For the richest interaction, the optional Animation Utility is highly recommended.

YUI Library: Connection Manager

2007-12-4 **v2.4**

Simple Use Case

```
var callback = {
  success: function(o) {
    document.getElementById('someEl').innerHTML =
      o.responseText;
  }
}
var connectionObject =
   YAHOO.util.Connect.asyncRequest('GET', 'file.php',
   callback);
```

Executes an asynchronous connection to `file.php`. If the HTTP status of the response indicates success, the full text of the HTTP response is placed in a page element whose ID attribute is "someEl".

Invocation (asyncRequest)

```
YAHOO.util.Connect.asyncRequest(str http method, str url[,
   obj callback object, str POST body]);
```

Arguments:
(1) **HTTP method (string):** GET, POST, HEAD, PUT, DELETE, etc. PUT and DELETE are not supported across all A-Grade browsers.
(2) **URL (string):** A url referencing a file that shares the same server DNS name as the current page URL.
(3) **Callback (object):** An object containing success and failure handlers and arguments and a scope control; see Callback Object detail for more.
(4) **POST body (string):** If you are POSTing data to the server, this string holds the POST message body.
Returns: **Transaction object.** `{ tId: int transaction id }` The transaction object allows you to interact (via Connection Manager) with your XHR instance; pass tId to CM methods such as `abort()`.

Callback Object: Members (All Optional)

1. **customevents:** Object containing any Custom Event handlers for transaction-level events (as alternatives to *success* and *failure* handlers below). Transaction-level Custom Events include *onStart*, *onComplete*, *onSuccess*, *onFailure*, *onAbort* and receive the same arguments as their global counterparts (see Global Custom Events, above right).
2. **success (fn):** The success method is called when an asyncRequest is replied to by the server with an HTTP in the 2xx range; use this function to process the response.
3. **failure (fn):** The failure method is called when asyncRequest gets an HTTP status of 400 or greater. Use this function to handle unexpected application/communications failures.
4. **argument (various):** The argument member can be an object, array, integer or string; it contains information to which your success and failure handlers need access.
5. **scope (obj)**: The object in whose scope your handlers should run.
6. **timeout (int):** Number of milliseconds CM should wait on a request before aborting and calling failure handler.
7. **upload (fn):** Handler to process file upload response.

Global Custom Events

These events fire for all transactions; subscribe via `YAHOO.util.Connect`; e.g.: `YAHOO.util.Connect.startEvent.subscribe(myFn);`

Event	Fires when...	Arguments
startEvent	transaction begins	transaction ID
completeEvent	transaction complete, but not yet reconciled as success or failure	transaction ID
successEvent	HTTP 2xx response received	Response object
failureEvent	HTTP 4xx, 5xx, or unknown response received	Response object
abortEvent	timeout/abort succeeds	transaction ID

Response Object

Your **success**, **failure**, and **upload** handlers are passed a single argument; that argument is an object with the following members:

tId	The transaction id.
status	The HTTP status code of the request.
statusText	The message associated with the HTTP status.
getResponse Header[]	Array collection of response headers and their corresponding values, indexed by header label.
getAllResponse Headers	String containing all available HTTP headers with name/value pairs delimited by "\n".
responseText	The server's full response as a string; for upload, the contents of the response's `<body>` tag.
responseXML	If a valid XML document was returned and parsed successfully by the XHR object, this will be the resulting DOM object.
argument	The arguments you defined in the Callback object's `argument` member.

Solutions

Roll up an existing form on the page, posting its data to the server:

```
YAHOO.util.Connect.setForm('formId');
var cObj = YAHOO.util.Connect.asyncRequest('POST',
   'formProcessor.php', callback);
```

Cancel a transaction in progress:

```
//if the transaction is created as follows...
var cObj = YAHOO.util.Connect.asyncRequest('GET',
   myServer.php', callback);
//...then you would attempt to abort it this way:
YAHOO.util.Connect.abort(cObj);
```

Connection Manager sets headers automatically for GET and POST transactions. If you need to **set a header manually**, use this syntax:

```
YAHOO.util.Connect.initHeader('SOAPAction', 'myAction');
```

Dependencies

Connection Manager requires the YAHOO Global Object and the Event Utility.

Key methods of YAHOO.util.Connect:

(o = Transaction object)

abort(o)
asyncRequest()
initHeader(s *label*, s *value*, [b *persistHeader*]) optional param persists header as a default for each subsequent transaction.
isCallInProgress(o)
setForm(str *formId* | o *form el ref*[, b *isUpload*, s *secureUri*]) optional params for file upload only; provide secureUri for iFrame only under SSL
setPollingInterval(int *i*)
setProgId(id)

HTTP Status Codes

2xx	Successful
3xx	Redirection
4xx	Client error
5xx	Server error

0	Communication failure
200	OK
400	Bad request
401	Unauthorized
403	Forbidden
404	Not found
408	Request timeout
410	Gone
500	Internal server error
502	Bad gateway
503	Service unavailable

YUI Library: CSS Reset, Base, Fonts, and Grids

2007-12-4 **v2.4**

Recommended Doctype and Render Mode

YUI works in both "Quirks" and "Standards" browser-rendering modes, but **we suggest using Standards mode by specifying this Doctype**:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

YUI CSS Reset + YUI CSS Base

YUI CSS Reset neutralizes browser CSS styles for HTML elements, creating a normalized platform. YUI CSS Base then rebuilds a consistent style foundation for common HTML elements.

YUI CSS Fonts: Setting Font Size and Family

Font-size: While still allowing users to zoom their font sizes, the YUI Fonts package renders all text at 13px by default. To preserve users' ability to zoom their fonts, **specify other sizes using percentages only** *(see chart in right column)*.

```
selector {font-size:123.1%;} /*16px*/
```

Font-family: The YUI Fonts package defines Arial as the default font and provides a degradation path through several alternates down to the generic family. Therefore, only specify a single font-family when you want a typeface other than Arial.

```
<style>selector {font-family:verdana;}</style>
```

Base Page Format

We find it useful to build a page in three stacked horizontal regions:

```
<body>
 <div id="doc"><!--doc, doc2, doc3, or doc4;
                   controls overall pg width-->
  <div id="hd"><!--header / masthead --></div>
  <div id="bd"><!--body--></div>
  <div id="ft"><!--footer--></div>
 </div>
</body>
```

Inside `#bd`, if two blocks (`.yui-b`) exist, mark one as primary by wrapping it in a container with the ID `yui-main`:

```
<div id="bd">
   <div id="yui-main">
      <div class"yui-b"><!--prim. block--></div>
   </div>
   <div class="yui-b"><!--sec. block--></div>
</div>
```

YUI CSS Grids: Nomenclature

#doc – #doc4	Define the overall width of the page.
.yui-t1 – .yui-t7	Choose the secondary column's width and orientation with one of seven *templates*.
.yui-g	Standard *grids* (`.yui-g`) instruct child *units* to share available space evenly. Nest inside other *grids* for additional subdivision.
.yui-gb – .yui-gf	Five special *grids* (.yui-gb ... .yui-gf.) are used when child *units* should occupy space unevenly and when dividing into three sections instead of the standard two. *(See chart in right column.)*
.yui-u	A *unit* inside a *grid*; generic; obeys parent *grid*.
.first	Overload the class attribute with "first" to indicate first of a series of *grids* or *units* to facilitate use of floats and margins.

YUI CSS Grids: Page Widths

750px, 950px, and 974px centered, and 100%-fluid page widths:

```
<div id="doc"> <-- 750 centered --></div>
<div id="doc2"><-- 950 centered --></div>
<div id="doc3"><-- 100% w/ 10px margin --></div>
<div id="doc4"><-- 974 centered --></div>
```

Customizing the Page Width: Divide your desired pixel width by 13 to find `em` width. Multiply that value by 0.9759 for IE via `*width`. For example, this is a 600px page-width:

```
   #custom-doc {
   margin:auto;text-align:left;
   width:46.15em; /* !IE */
   *width:45.04em;/*  IE */
   min-width:600px;}
```

Example: Nested 4-Column w/ "first" Identified

```
<div id="yui-main">
   <div class="yui-g">
      <div class="yui-g first">
         <div class="yui-u first"></div>
         <div class="yui-u"></div>
      </div>
      <div class="yui-g">
         <div class="yui-u first"></div>
         <div class="yui-u"></div>
      </div>
   </div>
</div>
```

Fonts Sizing Chart

For this PX	Use this percentage:
10	77
11	85
12	93 (was 92)
13	100
14	108 (was 107)
15	116 (was 114)
16	123.1 (was 122)
17	131 (was 129)
18	138.5 (was 136)
19	146.5 (was 144)
20	153.9 (was 152)
21	161.6 (was 159)
22	167
23	174
24	182
25	189
26	197

Grids: Nesting Grids (yui-g's)

`.yui-g`	1/2, 1/2
`.yui-gb`	1/3, 1/3, 1/3
`.yui-gc`	2/3, 1/3
`.yui-gd`	1/3, 2/3
`.yui-ge`	3/4, 1/4
`.yui-gf`	1/4, 3/4

Other configurations, such as ¼, ¼, ¼, ¼ can be rendered by nesting yui-g's inside other "yui-g" grids.

Grids: Templates (yui-t's)

`.yui-t1`	160 on left
`.yui-t2`	180 on left
`.yui-t3`	300 on left
`.yui-t4`	180 on right
`.yui-t5`	240 on right
`.yui-t6`	300 on right
`.yui-t7`	One full-width col.

YUI Library: DataTable

2007-12-4 **v2.4**

Simple Use Case: YAHOO.widget.DataTable

Markup (table is optional, can be empty container):

```
<div id="myContainer">
  <table>
    <thead><tr><th>…</th></tr></thead>
    <tbody><tr><td>…</td></tr></tbody>
  </table>
</div>
```

Script:

```
var myColumnDefs = [{key:"col1", label:"Col 1"},
  {key:"col2", label:"Col 2"}, ...];
var myDataSource =
  new YAHOO.util.DataSource([...]);
var myDataTable = new YAHOO.widget.DataTable(
  "myContainer", myColumnDefs, myDataSource);
```

Creates a DataTable instance from scratch or existing markup.

id	date	quantity	amount	title
po-0297	12/12/1978	12	$1.25	This Book Was Meant to Be Read Aloud
po-0167	3/24/1980	1	$4.00	A Book About Nothing
po-0783	1/3/1983		$12.12	The Meaning of Life
po-1482	3/11/1985	6	$3.50	Read Me Twice

Click to sort

Constructor: YAHOO.util.DataSource

```
YAHOO.util.DataSource(str|array|obj|HTMLFunction
  |HTMLTable live data[, obj config]);
```

Arguments:
(1) **live data:** Pointer to a set of data.
(2) **configuration object:** An optional object literal defines property values of a DataSource instance.

Constructor: YAHOO.widget.DataTable

```
YAHOO.widget.DataTable(str|HTMLElement el, array
  column defs, obj DataSource[, obj config]);
```

Arguments:
(1) **el:** HTML ID or HTMLElement for a DataTable container. May be emtpy or already contain <table> markup.
(2) **column defs:** An array of object literals defines Columns.
(3) **DataSource:** DataSource instance.
(4) **configuration object:** An optional object literal defines property values of a DataTable instance.

Key Interesting Moments in DataTable

Not all event types are available for all elements and units. See online docs for a complete list of DataTable Events.

Event:	oArgs Properties:
*element*ClickEvent, *element*DblclickEvent, *element*MousedownEvent, *element*MouseoutEvent, *element*MouseoverEvent	oArgs.event (HTMLEvent) oArgs.target (el) An *element* is a DOM element, such as button, cell, row, headerCell, headerCellLabel,e tc.
*unit*FormatEvent, *unit*HighlightEvent, *unit*SelectEvent, *unit*UnhighlightEvent, *unit*UnselectEvent, cellFormatEvent	oArgs.el (el) oArgs.record (YAHOO.widget.Record) When *unit* is a cell: oArgs.key (string) A *unit* is a cell or a row.
columnSortEvent	oArgs.column (YAHOO.widget.Column) oArgs.dir (string) "asc" || "desc"
editorRevertEvent, editorSaveEvent	oArgs.editor (object) oArgs.newData (object) oArgs.oldData (object)
refreshEvent	n/a
rowAddEvent	oArgs.record (YAHOO.widget.Record)
rowDeleteEvent	oArgs.oldData (object) oArgs.recordIndex (number) oArgs.trElIndex (number)
rowUpdateEvent	oArgs.record (YAHOO.widget.Record) oArgs.oldData (object)

All DataTable events are Custom Events (see Event Utility docs); subscribe to these events using "subscribe": (e.g. `myDataTable.subscribe("rowSelectEvent", fn);`).

Key DataTable Configuration Options

See online docs for complete list of DataTable configuration options.

Option (type)	Default	Description
caption (s) summary (s)	null	String values for caption element and summary attribute.
initialRequest (s)	""	Request value to send to DataSource at instantiation for data to populate the table.
paginated (b)	false	Whether or not the DataTable is paginated.
paginator (o)	see docs	Object literal of pagination values.
scrollable (b)	false	Whether or not the DataTable is scrollable.
selectionMode (s)	"standard"	A string value to define mode of row or cell selection.
sortedBy (o)	null	Object literal to define sorted columns.

DataTable options can be set in the constructor's second argument (e.g., `{paginated: true}`) or at runtime via set (e.g., `myTable.set("paginated", true);`).

YAHOO.widget.Column: Properties

abbr
children
className
editor
editorOptions
formatter
key
label
resizeable
sortable
sortOptions.defaultOrder
sortOptions.sortFunction
width

Note: Refer to online documentation for a full list of Column properties.

Dependencies

DataTable requires the YAHOO Global Object, Event Utility, Dom Collection, Element Utility, and DataSource Utility.

YUI Library: Dialog & SimpleDialog

2007-12-4 **v2.4**

Simple Use Case: YAHOO.widget.Dialog

Markup (optional, using HTML form in standard module format):

```
<div id="myDialog">
  <div class="bd">
    <form name="dlgForm" method="POST" action="
   post.php">
    <label for="firstname">First Name:</label>
   <input type="text" name="firstname" />
  </form></div>
</div>
```

Script:

```
//create the dialog:
var myDialog = new YAHOO.widget.Dialog("myDialog");
//set dialog to use form post on submit action:
myDialog.cfg.queueProperty("postmethod", "form");
//set up button handler:
var handleSubmit = function() {
   this.submit(); }; //default submit action
//set up button, link to handler
var myButtons = [ { text:"Submit",
   handler:handleSubmit, isDefault:true } ]);
//put buttons in configuration queue for processing
myDialog.cfg.queueProperty("buttons", myButtons);
mDialog.render(); //render dialog to page
myDialog.show(); //make dialog visible
```

Creates, renders and shows a panel using existing markup and all default Dialog settings.

Constructor: YAHOO.widget.Dialog & SimpleDialog

`YAHOO.widget.Dialog(str elId[, obj config]);`

Arguments:

(1) **Element ID:** HTML ID of the element being used to create the Dialog or SimpleDialog. If this element doesn't exist, it will be created.

(2) **Configuration Object:** JS object defining configuration properties for the Dialog. See Configuration section for full list.

The `postmethod` Property: Dialog & SimpleDialog

postmethod:	Characteristics:
`"none"`	Button handlers do all form processing.
`"form"`	Button handlers called, then form posted to url designated in form's target attribute.
`"async"`	Button handlers called, then form sent to url designated in form's target attribute using asynchronous XMLHttpRequest (via Connection Manager).

Key Interesting Moments in Dialog & SimpleDialog

See online docs for a complete list of Custom Events associated with Container controls.

Event	Arguments
beforeSubmitEvent	None.
cancelEvent	None.
submitEvent	None.

All events above are YUI Custom Events (see Event Utility docs); subscribe to these events using their subscribe method: `myDlg.hideEvent.subscribe(fnMyHandler);`.

Dialog/SimpleDialog Configuration Options

See online docs for complete list of Container options; see Simple Use Case (top left) for config. syntax.

Option (type)	Default	Description
text	null	Sets body text of SimpleDialog *(SimpleDialog only)*.
icon	"none"	Sets url for graphical icon. Six icons are provided: ICON_BLOCK, ICON_WARN, ICON_HELP, ICON_INFO, ICON_ALARM, and ICON_TIP. *(SimpleDialog only.)*
postmethod (s)	varies	Designates handling of form data; see box at bottom left. Default is "none" for SimpleDialog and "async" for Dialog.
buttons (a)	null	Array of button objects. Button objects contain three members: `text` label for button, `handler` function to process button click, and `isDefault` boolean specifying whether this is the default action on form submit.

See cheat sheet for Panel for additional configuration options; see online documentation for full list.

Solutions

Use `validate()` to **check form data** prior to submitting:

```
fnCheckEmail = function() {
  if (myDialog.getData().email.indexOf("@") > -1)
    {return true;} else {return false;} };
myDialog.validate = fnCheckEmail;
```

Set "success" handler for asynchronous post:

```
fnSuccess = function(o) {//function body};
myDialog.callback.success = fnSuccess;
```

Dependencies

Dialog requires the full Container package, the Yahoo Object, Dom Collection, and Event Utility. Animation, Button, Connection Manager and Drag And Drop are optional (though required for specific features).

YAHOO.widget.Dialog & SimpleDialog: Key Properties

body (el)
form (el)
callback (o) Connection Manager callback object for async transactions.
element (el) containing header, body & footer
footer (el)
header (el)
id (s) of the element

YAHOO.widget.Dialog & SimpleDialog: Methods

appendToBody(el *element*)
appendToFooter(el *element*)
appendToHeader(el *element*)
cancel() Executes cancel then hide().
getData() Returns obect of name/value pairs representing form data.
hide()
render([el *element*]) Argument required for Dialogs not built from existing markup. Dialog will not be in the DOM or visible until render is called.
setBody(str or el *content*)
setFooter(str or el *content*)
setHeader(str or el *content*)
submit() Executes submit followed by hide().
show()
getButtons()

YUI Library: Dom Collection

2007-12-4 **v2.4**

Methods Reference

Returns:	Method:
void	**addClass** (str | el ref | arr *el*, str *className*) Adds a class name to a given element or collection of elements
obj/array	**batch** (str | el ref | arr *el*, fn *method*, any *o*, b *overrideScope*) Returns the element(s) that have had the supplied method applied. The method will be provided the elements one at a time (method(el, o)).
str/array	**generateId** (str | el ref | arr *el*, str *prefix*) Generates a unique ID for the specified element.
obj/array	**get** (str | el ref | arr *el*) Returns an HTMLElement object or array of objects.
int	**getViewportHeight** () Returns the height of the client (viewport).
int	**getViewportWidth** () Returns the width of the client (viewport).
obj	**getAncestorByTagName** (str | *el ref el, str tag*) Returns the first HTMLElement ancestor of the element with the given tagName.
obj	**getAncestorByClassName** (str | *el ref el, str tag*) Returns the first HTMLElement ancestor of the element with the given className.
array	**getChildren** (str | *el ref el*) Returns the HTMLElement child nodes of the element.
array	**getElementsBy** (fn *method*, str *tag*, str | el ref *root*) Returns a array of HTMLElements that pass the test applied by supplied boolean method. For **optimized performance**, include a **tag** and/or **root node** when possible.
array	**getElementsByClassName** (str *className*, str *tag*, str | el ref *root*) Returns a array of HTMLElements with the given class. For **optimized performance**, include a **tag** and/or **root node** when possible.
obj	**getFirstChild** (str | *el ref el*) Returns the first HTMLElement childNode of the element.
obj	**getLastChild** (str | *el ref el*) Returns the last HTMLElement childNode of the element.
obj	**getNextSibling** (str | *el ref el*) Returns the next HTMLElement sibling of the element.
obj	**getPreviousSibling** (str | *el ref el*) Returns the previous HTMLElement sibling of the element.
obj	**getRegion** (str | el ref | arr *el*) Returns the region position of the given element.
str/array	**getStyle** (str | el ref | arr *el*, str *property*) Normalizes currentStyle and ComputedStyle.
int	**getX** (str | el ref | arr *el*) Gets the current X position of the element(s) based on page coordinates.
array	**getXY** (str | el ref | arr *el*) Gets the current position of the element(s) based on page coordinates.
int	**getY** (str | el ref | arr *el*) Gets current Y pos of the elem(s) based on page coordinates.

Methods Reference (continued)

Returns:	Method:
b/array	**hasClass** (str | el ref | arr *el*, str *className*) Determines whether the element(s) has the given className.
b/array	**inDocument** (str | el ref | arr *el*) Determines whether the element(s) is present in the current document.
obj	**insertAfter** (str | el ref *newNode, str| el ref refNode*) Inserts the newNode as the next HTMLElement sibling of the refNode.
obj	**insertBefore** (str | el ref *newNode, str| el ref refNode*) Inserts the newNode as the previous HTMLElement sibling of the refNode.
b	**isAncestor** (el ref *haystack*, el ref *needle*) Determines whether an HTMLElement is an ancestor of another HTMLElement in the DOM hierarchy.
void	**removeClass** (str | el ref | arr *el*, str *className*) Removes a class name from a given element or collection of elements.
void	**replaceClass** (str | el ref | arr *el*, str *oldClassName*, str *newClassName*) Replace a class with another class for a given element or collection of elements.
void	**setStyle** (str | el ref | arr *el*, str *property*, str *val*) Wrapper for setting style properties of HTMLElements.
void	**setX** (str | el ref | arr *el*, int *x*) Set the X position of the element(s) in page coordinates, regardless of how the element is positioned.
void	**setXY** (str | el ref | arr *el*, arr *pos*, b *noRetry*) Set the position of the element(s) in page coordinates, regardless of how the element is positioned.
void	**setY** (str | el ref | arr *el*, int *y*) Set the Y position of the element(s) in page coordinates, regardless of how the element is positioned.

Solutions

Get all elements to which the CSS class "header" has been applied:

```
headerEls =
   YAHOO.util.Dom.getElementsByClassName("header");
```

Get all elements by attribute:

```
checkTitle = function(el) {
  return (el.getAttribute("title")=="Click here.");}
myEls = YAHOO.util.getElementsBy(checkTitle, "a", "yui-
   main");
```

Set element's opacity using setStyle:

```
YAHOO.util.Dom.setStyle(myEl, "opacity", "0.5");
```

Dependencies

The Dom Collection requires the YAHOO Global Object.

Useful Dom Methods:

appendChild()
click()
cloneNode()
createElement()
createTextNode()
focus()
getAttribute()
getElementById()
getElementsBy TagName()
hasAttribute()
hasChildNodes()
insertBefore()
removeAttribute()
removeChild()
replaceChild()
scrollIntoView()
setAttribute()
setInterval()
setTimeout()

Dom Node Properties:

attributes
childNodes
className
disabled
firstChild
id
innerHTML
lastChild
nextSibling
nodeType
nodeName
nodeValue
offsetHeight
offsetWidth
parentNode
previousSibling
tagName

Note: These are not exhaustive lists.

YUI Library: Drag & Drop

2007-12-4c **v2.4**

Simple Use Case: Making an Element Draggable

```
myDDobj = new YAHOO.util.DD("myDiv");
```

Makes the HTML element whose id attribute is "myDiv" draggable.

Constructor (YAHOO.util.DD, DDProxy, DDTarget)

```
YAHOO.util.DD(str | el ref target[, str group name,
  obj configuration]);
```

Arguments:
(1) **Element:** ID or elem. ref. of the element to make draggable; deferral is supported if the element is not yet on the page.
(2) **Group Name:** An optional string indicating the DD group; DD objects only "interact with" other objects that share a group.
(3) **Configuration:** An object containing name-value pairs, used to set any of the DD object's properties.

Properties & Methods of YAHOO.util.DragDrop

Properties:
available (b)
dragOnly (b)
groups (ar)
id (s)
invalidHandle Classes (s[ ])
invalidHandleIds (obj)
isTarget (b)
maintainOffset (b)
padding (int[ ])
primaryButtonOnly (b)
xTicks (int[ ])
yTicks (int[ ])

Methods:
addInvalidHandle Class (s *cssClass*)
addInvalidHandleId (s *id*)
addInvalidHandle Type (s *tagName*)
addToGroup (s *groupName*)
clearTicks()
clearConstraints()
getDragEl()
getEl()
isLocked()
lock()
removeFromGroup(o *dd*, s *group*)
removeInvalid HandleClass(s *cssClass*)
removeInvalid HandleId(s *id*)
removeInvalidHandle Type (s *tagName*)
resetConstraints()
setDragElId(s *id*)
setHandleElId (s *id*)
setOuterHandleElId (s *id*)
setPadding(i *top*, i *right*, i *bottom*, i *left*)
setXConstraint(i *left*, i *right*, i *tick size*)
setYConstraint(i *up*, i *down*, i *tick size*)
unlock()
unreg()

Properties & Methods of YAHOO.util.DD & .DDProxy

Inherit from YAHOO.util.DragDrop and add the following:

YAHOO.util.DD Properties:
scroll (b)

YAHOO.util.DDProxy Properties:
centerFrame (b)
resizeFrame (b)

Interesting Moments in Drag & Drop

Moment	Point Mode	Intersect Mode	Event (e)
onMouseDown	e	e	mousedown
startDrag	x, y	x, y	n/a
onDrag	e	e	mousemove
onDragEnter	e, id	e, DDArray	mousemove
onDragOver	e, id	e, DDArray	mousemove
onDragOut	e, id	e, DDArray	mousemove
onDragDrop	e, id	e, DDArray	mouseup
onInvalidDrop	e	e	mouseup
endDrag	e	e	mouseup
onMouseUp	e	e	mouseup

These "moments" are exposed as events on your DD instances; they are methods of YAHOO.util.DragDrop. The table above identifies the arguments passed to these methods in Point and Intersect modes.

Solutions

Add a drag handle to an exsting DD object:

```
myDDobj.setHandleElId('myDragHandle');
```

Set the "padding" or "forgiveness zone" of a DD object:

```
myDDobj.setPadding(20, 30, 20, 30); //units are
  pixels, top/rt/bt/left
```

Get the "best match" from an onDragDrop event in Intersect Mode where the dragged element is over more than one target:

```
myDDobj.onDragDrop = function(e, DDArray) {
 oDDBestMatch =
   YAHOO.util.DragDropMgr.getBestMatch(DDArray);}
```

Override an interesting moment method for a DD object instance:

```
myDDobj = new YAHOO.util.DD("myDiv");
myDDobj.startDrag = function(x,y) {
  this.iStartX = x; this.iStartY = y;
}
```

Change the look and feel of the proxy element at the start of a drag event using YAHOO.util.DDProxy:

```
myDDobj.startDrag(x,y) {
  YAHOO.util.Dom.addClass(this.getDragEl(),
   "myCSSClass"); }
```

Lock Drag and Drop across the whole page:

```
YAHOO.util.DragDropMgr.lock();
```

Switch to Intersect Mode:

```
YAHOO.util.DragDropMgr.mode =
  YAHOO.util.DragDropMgr.INTERSECT;
```

Drag & Drop Manager: Properties

clickPixelThresh (i)
clickTimeThresh (i)
mode either YAHOO.util.DragDropMgr.POINT or .INTERSECT
preventDefault (b)
stopPropagation (b)
useCache (b)

Drag & Drop Manager: Methods

oDD=instance of DragDrop object

getBestMatch(a [oDDs])
getDDById(s *id*)
getLocation(oDD)
getRelated(oDD, b *targets only*)
isDragDrop(s *id*)
isHandle(s *DDId*, s *HandleId*)
isLegalTarget(oDD, oDD *target*)
isLocked()
lock()
refreshCache()
swapNode()
unlock()

***Note:**
YAHOO.util.DragDropMgr is a singleton; changes made to its properties (such as locking or unlocking) affect Drag and Drop globally throughout a page.

Dependencies

Drag & Drop requires the YAHOO object, DOM, and Event.

YUI Library: Event Utility & Custom Event

2007-12-4 **v2.4**

Simple Use Case: Adding Event Listeners

```
YAHOO.util.Event.addListener("myDiv", "click",
   fnCallback);
```

Adds the function `fnCallback` as a listener for the click event on an HTML element whose id attribute is `myDiv`.

Invocation (addListener)

```
YAHOO.util.Event.addListener(str | el ref | arr
   target[s], str event, fn callback[, obj
   associated object, b scope);
```

Arguments:
(1) **Element or elements:** You may pass a single element or group of elements in an array; references may be id strings or direct element references.
(2) **Event:** A string indicating the event ('click', 'keypress', etc.).
(3) **Callback:** The function to be called when the event fires.
(4) **Associated object:** Object to which your callback will have access; often the callback's parent object.
(5) **Scope:** Boolean — if true, the callback runs in the scope of the associated object.

Event Utility Solutions

Using **onAvailable**:

```
fnCallback = function() { //will fire when element
   becomes available}
YAHOO.util.Event.onAvailable('myDiv', fnCallback);
```

Using Event's **convenience methods**:

```
fnCallback = function(e, obj) {
  myTarget = YAHOO.util.Event.getTarget(e, 1);
  //2nd argument tells Event to resolve text nodes
}
YAHOO.util.Event.addListener('myDiv', 'mouseover',
   fnCallback, obj);
```

Prevent the event's default behavior from proceeding:

```
YAHOO.util.Event.preventDefault(e);
```

Remove listener:

```
YAHOO.util.Event.removeListener('myDiv',
   'mouseover', fnCallback);
```

Dependencies

Event Utility requires the YAHOO Global Object.

Simple Use Case: Custom Event

```
myEvt = new YAHOO.util.CustomEvent("my event");
mySubscriber = function(type, args) {
  alert(args[0]); } //alerts the first argument
myEvt.subscribe(mySubscriber);
myEvt.fire("hello world");
```

Creates a new Custom Event instance and a subscriber function; the subscriber alerts the event's first argument, "hello world", when the event is fired.

Constructor (Custom Event)

```
YAHOO.util.CustomEvent(str event name[, obj scope object,
   b silent, int signature ]);
```

Arguments:
(1) **Event name:** A string identifying the event.
(2) **Scope object:** The default scope in which subscribers will run; can be overridden in subscribe method.
(3) **Silent:** If true, hides event's activity from Logger when in debug mode.
(4) **Argument signature:** `YAHOO.util.CustomEvent.LIST` by default — all arguments passed to handler in a single array. `.FLAT` can be specified to pass only the first argument.

Subscribing to a Custom Event

```
myEvt.subscribe(fn callback[, obj associated object, b
   scope]);
```

Arguments for `subscribe`:
(1) **Callback:** The function to be called when the event fires.
(2) **Associated object:** Object to which your callback will have access as an argument; often the callback's parent object.
(3) **Scope:** Boolean — if true, the callback runs in the scope of the associated object.

Arguments received by your callback function:
When using the default argument signature (`YAHOO.util.CustomEvent.LIST`; see Constructor section above), your callback gets three arguments:

(1) **Type:** The type of Custom Event, a string.
(2) **Arguments:** All arguments passed in during `fire`, as an array.
(3) **Associated object:** The associated object passed in during `subscribe`, if present.

```
myEvt.fire(arg1, arg2);
var myHandler = function(sType, aArgs, oObj) {/*aArgs=[arg1, arg2]*/};
myEvt.subscribe(myHandler, oObj);
```

When using the optional argument signature (YAHOO.util.CustomEvent.FLAT; see Constructor section above), your callback gets two arguments:

(1) **Argument:** The first argument passed when the event is fired.
(2) **Associated object:** Passed in during `subscribe`, if present.

```
myEvt.fire(arg1);
var myHandler = function(arg, oObj) {/*arg=arg1*/};
myEvt.subscribe(myHandler, oObj);
```

Event Utility Methods:

addListener(…)
getCharCode(e)
getListeners(el [, type]**)**
getPageX(e)
getPageY(e)
getRelatedTarget(e)
getTarget(e)
getTime(e)
getXY(e): returns array [pageX, pageY]
onAvailable(s id || el ref, fn callback, o obj, b scope)
onContentReady(s id || el ref, fn callback, o obj, b scope)
onDOMReady(s id || el ref, fn callback, o obj, b scope)
preventDefault(e)
purgeElement(el [, recurse, type]**)**
removeListener(…)
stopEvent(e): same as preventDefault plus stopPropagation
stopPropagation(e)

DOM Event Object Properties & Methods:

altKey (b)
bubbles (b)
cancelable (b)
***charcode** (i)
clientX (i)
clientY (i)
ctrlKey (b)
currentTarget (el)
eventPhase (i)
isChar (b)
keyCode (i)
metaKey (i)
***pageX** (i)
***pageY** (i)
***preventDefault()**
***relatedTarget** (el)
screenX (i)
screenY (i)
shiftKey (b)
***stopPropagation**()
***target** (el)
***timestamp** (long)
type (s)
[**use Event Utility method*]

YUI Library: Get Utility

2007-12-4 **v2.4**

Simple Use Case: Get an External Script

With the **Get Utility** on the page, you can bring in an external script file in two steps:

1. Define the logic you want to execute when the script successfully loads;
2. Get the script.

```
var successHandler = function(oData) {
  //code to execute when all requested scripts have been
  //loaded; this code can make use of the contents of those
  //scripts, whether it's functional code or JSON data.
}
var aURLs = [
  "/url1.js", "/url2.js", "/url3.js" //and so on
];
YAHOO.util.Get.script(aURLs, {
  onSuccess: successHandler
});
```

Usage: YAHOO.util.Get.script()

`YAHOO.util.Get.script(str or arr` *`url[s]`*`[, obj` *`options`*`])`

Arguments:

(1) **URL[s]:** A string or array of strings containing the URL[s] to be inserted in the document via script nodes.

(2) **options:** An object containing any configuration options you'd like to specify for this transaction. See the **Configuration Options** table for the full list of options.

Returns:

(1) **Transaction Object:** Object containing single field, *stru* tId, a unique string identifying this transaction.

Note: Scripts downloaded will be executed immediately; only use this method to procure JavaScript whose source is trustworthy beyond doubt.

Usage: YAHOO.util.Get.css()

`YAHOO.util.Get.css(str or arr` *`url[s]`*`[, obj` *`options`*`])`

Arguments:

(1) **URL[s]:** A string or array of strings containing the URL[s] to be inserted in the document via link nodes.

(2) **options:** An object containing any configuration options you'd like to specify for this transaction. See the **Configuration Options** table for the full list of options.

Note: Returns the same Transaction Object as `script()`; see above.

Configuration Options

Field	Type	Description
onSuccess	fn	Callback method invoked by Get Utility when the requested file(s) have loaded successfully.
onFailure	fn	Callback method invoked by Get Utility when an error is detected or `abort` is called.
win	obj	The window into which the loaded resource(s) will be inserted. Default: the current window.
scope	obj	The execution scope in which the `onSuccess` or `onFailure` callback will run. Default: the current window.
data	any	Data to pass as an argument to `onSuccess` or `onFailure` callbacks. Default: null.
autopurge	bool	If `true`, script nodes will automatically be removed every 20 transactions (configurable via `YAHOO.util.Get.PURGE_THRESH` property. Default: `true` for script nodes, `false` for CSS nodes.
varName	arr (of strings)	Safari 2.x does not reliably report the load-complete status of script nodes; use this property to provide Get with a globally accessible property that will be available when the script has loaded. This array is parallel to the *urls* array passed in as the first argument to `script()`.

Use configuration options by passing an optional object containing config options to `YAHOO.util.Get.script` or `YAHOO.util.Get.css` as the second argument:
`YAHOO.util.Get.script("http://json.or/json.js", {onSuccess: function(o) {YAHOO.log("success!");}});`

Callback Arguments

Fields available in the object passed to your `onSuccess` or `onFailure` callback.

Field	Type	Description
tId	str	The unique identifier for this transaction; this string is available as the `tId` member of the object returned to you upon calling the script or css method.
data	any	The `data` field you passed to your configuration object when the `script` or `css` method was called. Default: `null`.
win	obj	The window into which the loaded resource(s) were inserted.
nodes	array	An array containing references to node(s) created in processing the transaction. These will be script nodes for JavaScript and link nodes for CSS.
purge	Fn	Calling the returned `purge()` method will immediately remove the created nodes.

Solutions

Set up a transaction making use of configuration options; then make use of the data passed to the callback handler:

```
var successHandler = function(o) {
  //o contains all of the fields described in the callback args table
  o.purge(); //removes the script node immediately after executing;
  YAHOO.log(o.data); //the data passed in configobject
}
var objTransaction = YAHOO.util.Get.script("http://json.org/json.js",
  { onSuccess: successHandler,
    scope: this, //successHandler will run in the scope of "this"
    data: {field1: value1, field2: value2}} //you can pass data in
                                            //any format here
);
```

YAHOO.util.Get Methods

css(string | arr *URLs*[, obj *config options*]) see usage at right

script(string | arr *URLs*[, obj *config options*]) see usage at right

abort(string | obj *tId*) takes either the transaction id or transaction object generated by `script` or `css`; aborts transaction if possible; fires `onFailure` handler

YAHOO.util.Get Global Configuration Properties

YAHOO.util.Get.POLL_FREQ int when polling is necessary to check on the status of a loading file (eg, where the load event is unreliable), this controls the polling interval in milliseconds

YAHOO.util.Get.PURGE_THRESH int controls the number of added script or link nodes that will accumulate prior to being automatically purged

Dependencies

The Get Utility requires only the YAHOO Global Object.

YUI Library: ImageLoader Utility [beta]

2007-12-4b **v2.4**

Simple Use Case: ImageLoader Group Object

Create a `YAHOO.util.ImageLoader.group` object with a trigger and time limit. Then register images with the group:

```
//group with 'someDivId' click trigger & 2 sec limit:
var myGroup = new YAHOO.util.ImageLoader.group('someDivId', 'click', 2);
myGroup.registerBgImage('imgDivId', 'http://some.image/url');
```

This will cause `imgDivId`'s background image to load either when `someDivId` is clicked or two seconds after page load, whichever comes first.

Constructor: ImageLoader Group Object

```
YAHOO.util.ImageLoader.group([obj triggerElement, str triggerAction, int timeLimit])
```

Arguments:

(1) **triggerElement:** The object of the trigger event. Can be a DOM id or object.

(2) **triggerAction:** The action of the trigger event. `triggerElement` and `triggerAction` are optional (can be `null`). But if one is supplied, the other must be as well.

(3) **timeLimit:** Maximum time to wait for the trigger event to be fired.

ImageLoader Image Registration

Source images (e.g., an <img> element):

```
myGroup.registerSrcImage('imgImgId',
   'http://some.image/url');
```

Background images (e.g., a element with a background image):

```
myGroup.registerBgImage('imgDivId',
   'http://some.image/url');
```

PNG background images (e.g., a element with a PNG bg image):

```
myGroup.registerPngBgImage('imgDivId',
   'http://some.png_image/url');
```

Solution: Simple Image Loading

Set up the HTML and JavaScript for delayed image loading:

```
<div id='square'>
<img id='squareImg' /><!-- note no "src" attribute -->
</div>

// in script, create group and register image
var sqGrp = new YAHOO.util.ImageLoader.group('square',
   'mouseover', 3);
sqGrp.registerSrcImage('squareImg',
   'http://some.image/url');
```

ImageLoader Objects: Members

See online docs for complete details on members.

YAHOO.util.ImageLoader.group

Member	Type	Description
timeoutLen	number	Length of time limit, in seconds. Also the third argument in the constructor.
foldConditional	boolean	Flag to check if images are above the fold.
className	string	CSS class name that will identify images belonging to the group.
name	string	Optional. Only used to identify the group in Logger Control logging statements.
addTrigger	method	Adds a trigger to the group.
registerBgImage	method	Registers a background image with the group.
registerSrcImage	method	Registers a src image with the group.
registerPngBgImage	method	Registers an alpha-channel-type png background image with the group.

YAHOO.util.ImageLoader.imgObj

Member	Type	Description
setVisible	boolean	Whether the style.visibility should be set to "visible" after the image is fetched.
width	number	Size to set as width of image after image is fetched. Only applies to source-type images. Third argument in group's registerSrcImage method.
height	number	Size to set as height of image after image is fetched. Only applies to source-type images. Fourth argument in group's registerSrcImage method.

Solution: Image Loading with Class Names

Set up the CSS, HTML, and JavaScript for delayed image loading:

```
/* set an overriding background:none */
.yui-imgload-circle { background:none !important; }

<!-- this div will get the trigger event -->
<div id='circle'>
   <!-- set the src to some transparent image, the
   background-image to the true image, and the class to match
   the CSS -->
   <img id='circleImg' src='http://some.transparent/image'
   style='background-image:url("http://some.image/url");'
   class='yui-imgload-circle' height='20' width='20' />
</div>

// create group and identify class name
var circleGroup = new YAHOO.util.ImageLoader.group('circle',
   'mouseover', 3);
circleGroup.className = 'yui-imgload-circle';
```

YAHOO.util.ImageLoader.group Methods:

addTrigger(obj *domObject* or str *domElementId*, str *eventAction*) adds a trigger event

registerBgImage(str *imgElId*, str *url*) adds a background-type image to the group returns YAHOO.util.ImageLoader.imgObj

registerSrcImage(str *imgElId*, str *url*, [int *width*, int *height*]) adds a source-type image to the group; optional width and height resize the image to those constraints returns YAHOO.util.ImageLoader.imgObj

registerPngBgImage(str *imgElId*, str *url*) adds a png-background-type image to the group returns YAHOO.util.ImageLoader.imgObj

Dependencies

The YUI ImageLoader Utility requires the Yahoo Global Object, Dom Collection, and Event Utility.

YUI Library: JSON Utility [beta]

2007-12-4b **v2.4**

Simple Use Case: Parse a JSON string

One of the core use cases for the JSON Utility is to take string data formatted in JavaScript Object Notation and to validate the string as genuine JSON before evaluating it and processing it in script. `YAHOO.lang.JSON.parse()` provides this functionality:

```
var jsonString = '{"productId":1234,
  "price":24.5, "inStock":true, "bananas":null}';

// Parsing JSON strings can throw a SyntaxError
// exception, so we wrap the call
// in a try catch block
try {
    var prod=YAHOO.lang.JSON.parse(jsonString);
}
catch (e) {
    alert("Invalid product data");
}

// We can now interact with the data
if (prod.price < 25) {
    prod.price += 10; // Price increase!
}
```

Usage: YAHOO.lang.JSON.parse()

`YAHOO.lang.JSON(str` *`JSON`*`[, fn` *`filter`*`])`

Arguments:
(1) **JSON:** A string containing JSON-formatted data that you wish to validate and parse.
(2) **filter**: A function that will be used to filter the JSON contents; see the Solutions box for more.

Returns:
JavaScript representation: The returned value of the evaluated JSON string (if no exception was thrown in its evaluation).

Usage: YAHOO.lang.JSON.stringify()

`YAHOO.lang.JSON.stringify(obj` *`object`*`[, arr` *`whitelist`*`, n` *`depth`*`])`

Arguments:
(1) **object:** The JavaScript object you want to stringify.
(2) **whitelist:** An optional array of acceptable keys to include.
(3) **depth:** An optional number specifying the depth limit to which stringify should recurse in the object structure (there is a practical minimum of 1).

Returns:
JSON string: A string representing the object in valid JSON notation.

Using the JSON Format

JSON data is characterized as a collection of objects, arrays, booleans, strings, numbers, and null. The notation follows these guidelines:

1. Objects begin and end with curly braces (`{}`).
2. Object members consist of a string key and an associated value separated by a colon (`"key" : VALUE`).
3. Objects may contain any number of members, separated by commas (`{ "key1" : VALUE1, "key2" : VALUE2 }`).
4. Arrays begin and end with square braces and contain any number of values, separated by commas (`[ VALUE1, VALUE2 ]`).
5. Values can be a string, a number, an object, an array, or the literals true, false, and null.
6. Strings are surrounded by double quotes and can contain Unicode characters and common backslash escapes ("`new\nline`").

JSON.org has helpful format diagrams and specific information on allowable string characters.

Solutions: Using the *filter* argument

You can filter out and/or reformat specific pieces of data while applying the `parse` method by passing in a second (optional) argument, `filter`. Filter is a function that is passed the key and value of the item it is filtering; based on the key and value, the filter can return a formatted value for the item or return `undefined` to omit the key altogether.

```
var currencySymbol = "$"
function myFilter(key,val) {
  // format price as a string
  if (key == "price") {
    var f_price = currencySymbol + (val % 1 ? val + "0" :
        val + ".00");
    return f_price.substr(0,f_price.indexOf('.') + 3);
  }
  // omit keys by returning undefined
  if (key == "bananas") {
    return undefined;
  }
}

var formattedProd = YAHOO.lang.JSON.parse(jsonString,
   myFilter);

// key "bananas" is not present in the formattedProd object
if (YAHOO.lang.isUndefined(formattedProd.bananas)) {
  alert("We have no bananas today");
}

// and the price is the formatted string "$24.50"
alert("Your price: " + formattedProd)
```

YAHOO.lang.JSON Methods

parse(str *JSON*[.fn *filter*]) see usage at left
stringify(obj *object*[, arr *whitelist*, n *depth*) see usage at left

Dependencies

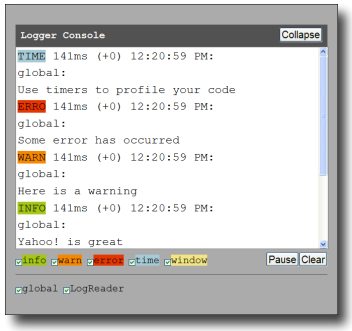
The JSON Utility requires only the YAHOO Global Object.

YUI Library: Logger

2007-12-4 **v2.4**

Simple Use Case (LogReader)

```
<div id="myLogger"></div>
<script>
var myLogReader = new
   YAHOO.widget.LogReader
   ("myLogger");
</script>
```



Instantiates a new LogReader object, myLogReader, which is bound to a div whose id attribute is 'myLogger'. The result will be a visual LogReader display.

To create a LogReader that floats outside the page context, omit the reference to a context div. Your LogReader will then be appended to the page and positioned absolutely. If the YUI Drag & Drop Library is included on the page, it will be draggable.

Constructor (LogReader)

```
YAHOO.widget.LogReader([str html id | obj element
   reference, obj configuration object]);
```

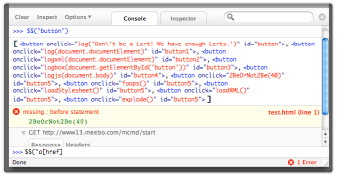
Arguments:

(1) **HTML element (string or object):** An optional reference to an HTML id string or element object binds the LogReader to an existing page element.

(2) **Configuration object (object):** An optional object literal defines LogReader settings. All properties of a LogReader instance can be set via the constructor by using this object.

Logging via `console.log()`

A growing number of browsers and extensions support the JavaScript method `console.log()`. The excellent FireBug extension to FireFox supports this method, as does the JavaScript console in Apple's Safari browser. Enable this feature using Logger's `enableBrowserConsole()` method.



Dependencies

Logger requires the YAHOO object, Dom, and Event; Drag & Drop is optional. Use in combination with –debug versions of YUI files for built-in logging from components.

Simple Use Case (Logger)

```
YAHOO.log("My log message", "error", "mysource");
```

Logs a message to the default console and to `console.log()`, if enabled; the source is "mysource" and the category is "error". Custom categories and sources can be added on the fly.

Constructor (LogWriter)

Creates a separate, named bucket for your log messages:

```
YAHOO.widget.LogWriter(str sSource);
```

Arguments:

(1) **Source (string):** The source of log messages. The first word of the string will be used to create a LogReader filter checkbox. The entire string will be prepended to log messages so they can be easily tracked by their source.

Solutions

Log a message using a pre-styled logging category:

```
YAHOO.log("My log message.", "warn");
```

Create a new logging category on the fly:

```
YAHOO.log("My log message.", "myCategory");
```

Style a custom logging category in CSS:

```
.yui-log .myCategory {background-color:#dedede;}
```

Log a message, **creating a new "source" on the fly**:

```
YAHOO.log("My log message.", "warn", "newSource");
```

In script, **hide and show** the logging console:

```
myLogReader.hide();
myLogReader.show();
```

In script, **pause and resume** output to the console:

```
myLogReader.pause();
myLogReader.resume();
```

Instantiate your own LogWriter to write log messages categorized by their source:

```
MyClass.prototype.myLogWriter = new
   YAHOO.widget.LogWriter("MyClass of MyApp");
var myInstance = new MyClass();
myInstance.myLogWriter.log("This log message can now
   be filtered by its source, MyClass."); //"MyClass
   of MyApp", the full name of the source, will be
   prepended to the actual log message
```

YAHOO.widget.Logger Static Properties:

loggerEnabled (b)
maxStackEntries (int)

YAHOO.widget.Logger Static Methods:

log(sMsg, sCategory, sSource)
disableBrowserConsole()
enableBrowserConsole()
getStack()
getStartTime()
reset()

YAHOO.widget.Logger Custom Events:

categoryCreateEvent
sourceCreateEvent
newLogEvent
logResetEvent

LogReader Properties:

verboseOutput (b)
newestOnTop (b)
thresholdMax (int)
thresholdMin (int)
outputBuffer (int)

LogReader Methods:

hide()/show()
pause()/resume()
collapse()/expand()
clearConsole()
hideCategory()/ showCategory()
hideSource()/ showSource()

LogWriter Methods:

log(sMsg, sCategory, sSource**)**

Categories

info
warn
error
time
window

(Pass in other categories to `log()` to add to this list.)

YUI Library: Menu

2007-12-4 **v2.4**

Simple Use Case: YAHOO.widget.Menu

Markup (optional, using standard module format):

```
<div id="mymenu">
  <div class="bd">
    <ul>
      <li><a href=" … ">item one</a></li>
      <li><a href=" … ">item two</a></li>
    </ul>
  </div>
</div>
```

Script:

```
var oMenu = new YAHOO.widget.Menu("mymenu");
oMenu.render();
oMenu.show();
```

Creates, renders and shows a menu using existing markup.

Constructor: YAHOO.widget.Menu

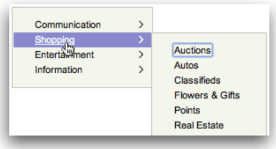
```
YAHOO.widget.Menu(str elId[, obj config]);
```

Arguments:

(1) **Element ID:** HTML ID of the element being used to create the Menu. If this element doesn't exist, it will be created and appended to the document body.

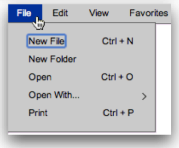
(2) **Configuration Object:** JS object defining configuration properties for the Menu instance. See Configuration section for full list.

Three Types of Menus



Classic Menu
YAHOO.widget.Menu
A classic menu is defined by a list of menu items, visible on pageload, each of which can contain sub-menus that fly out on mouseover or on click.

Context Menu
YAHOO.widget.ContextMenu
Context Menus are classic-style menus associated with a context element; they appear when an action, like right-clicking, is performed on the context element.



Menu Bar
YAHOO.widget.MenuBar
Menu Bars are horizontally arranged collections of menus, with each menu actuated by a click or mouseover action.

Key Interesting Moments in Menu

See online docs for a complete list of Menu's Custom Events.

beforeRenderEvent	renderEvent
renderEvent	beforeShowEvent
showEvent / hideEvent	beforeHideEvent

All Menu events are YUI Custom Events (see Event Utility docs); subscribe to these events using their subscribe method: `oMenu.hideEvent.subscribe(fnMyHandler);`.

Key Menu Configuration Options

See online docs for complete list of Menu options.

Option (type)	Default	Description
constrain toviewport (b)	true	Forces a menu to remain inside the confines of the viewport.
itemData (a)	null	Array of MenuItems to be added to Menu.
lazyLoad (b)	false	Boolean value specifies whether Menu should defer initialization and rendering of submenus until needed.
position (s)	"dynamic" ("static" for MenuBar)	Static: in the flow of the document, visible by default. Dynamic: hidden by default, outside of page flow.
submenuhidedelay (n)	250	Delay (in ms) for hiding a submenu as a user mouses out of parent MenuItem while mousing toward the submenu.
showdelay (n)/ hidedelay (n)	250 (show), 0 (hide)	Built-in delay when showing or hiding the Menu, in miliseconds.
trigger (s || o || a)	Null	The id(s) or node reference(s) for the element(s) whose contextmenu event triggers the context menu's display.
maxheight (n)	0	The maximum height (in pixels) for a menu before the contents of the body are scrolled.

Menu options can be set in the constructor's second argument (eg, `{visible: true}`) or at runtime via setProperty (eg, `oMenu.cfg.setProperty("visible", false);`).

Key MenuItem Configuration Options

See online docs for complete list of MenuItem options.

Option (type)	Default	Description
checked (b)	false	Renders the item with a checkmark.
disabled (b)	false	If set to true the MenuItem will be dimmed and will not respond to user input or fire events.
selected (b)	false	If set to true the MenuItem will be highlighted.
submenu (o)	null	Appends a menu to the MenuItem.
target (s)	null	Value for the "target" attribute of the item's anchor element.
text (s)	null	Text label for the item.
url (s)	"#"	URL for the anchor's "href" attr.

MenuItem options can be set in the constructor's second argument (eg, `{disabled: true}`) or at runtime via setProperty (eg, `oMenuItem.cfg.setProperty("disabled", true);`).

YAHOO.widget.Menu: Properties

parent
element
id

YAHOO.widget.Menu: Methods

addItem(o || s [,i])
addItems(o || s [,i])
getItem(i [,i])
getItems()
getItemGroups()
getSubmenus()
getRoot() returns root Menu instance
insertItem(o || s [,i] [,i])
removeItem(o || i [,i])
setItemGroupTitle(s [,i])
show()
hide()
clearContent()
render([el])
destroy()

YAHOO.widget.MenuItem: Properties

element
parent
id
groupIndex
index
value

YAHOO.widget.MenuItem: Methods

focus()
blur()

Dependencies

Menu requires the Container Core package, the YAHOO Object, Event, and Dom.

YUI Library: Panel

2007-12-4 **v2.4**

Simple Use Case: YAHOO.widget.Panel

Markup (optional, using standard module format):

```
<div id="myPanel">
  <div class="hd">Header content.</div>
  <div class="bd">Body content.</div>
  <div class="ft">Footer content.</div>
</div>
```

Script:

```
var oPanel = new YAHOO.widget.Panel("myPanel");
oPanel.render();
oPanel.show();
```

Creates, renders and shows a panel using existing markup and all default Panel settings.

Constructor: YAHOO.widget.Panel

```
YAHOO.widget.Panel(str elId[, obj config]);
```

Arguments:

(1) **Element ID:** HTML ID of the element being used to create the Panel. If this element doesn't exist, it will be created.

(2) **Configuration Object:** JS object defining configuration properties for the panel. See Configuration section for full list.

Solutions

There are three ways to **configure options on your Panel**:

```
// 1. In the constructor, via an object literal:
var myPanel = new YAHOO.widget.Panel("myPanel", {
   visible:false });
// 2. Via "queueProperty", prior to rendering:
myPanel.cfg.queueProperty("visible",false);
// 3. Via "setProperty" after rendering:
myPanel.cfg.setProperty("visible",false);
```

Align the top left corner of your Panel with the bottom right corner of an element whose HTML ID is "contextEl":

```
myPanel.cfg.setProperty("context", ["contextEl",
   "tl", "br"]);
```

Subscribe to a Panel Custom Event, listening for changes to the Panel's postion, alerting its new position after move:

```
alertMove = function(type, args) {
  alert(args[0] + ", " + args[1]);
}
myPanel.subscribe("move", alertMove);
```

Key Interesting Moments in Panel

See online docs for a complete list of Panel's Custom Events.

Event	Arguments
beforeRenderEvent	None.
renderEvent	None.
beforeShowEvent	None.
showEvent	None.
beforeHideEvent	None.
hideEvent	None.
beforeMoveEvent	X, Y to which the Panel will be moved.
moveEvent	X, Y to which the Panel was moved.
hideMaskEvent	None.
showMaskEvent	None.
changeContentEvent	None.
changeBodyEvent	String or element representing new body content (**Note:** there are corresponding Header and Footer change events, too).

All Panel events are YUI Custom Events (see Event Utility docs); subscribe to these events using their subscribe method: `myPanel.hideEvent.subscribe(fnMyHandler);`.

Key Panel Configuration Options

See online docs for complete list of Panel options; see Solutions (bottom left) for how to set your options.

Option (type)	Default	Description
close (b)	null	Display close icon.
draggable (b)	null	Make the Panel draggable.
modal (b)	null	Use a modal mask behind Panel when Panel is visible.
visible (b)	true	Sets the "display" style property to "block" (true) or "none" (false).
x, y, and xy (int, int, ar)	null	These properties can be used to set the Panel's "top" and/or "left" styles.
context (ar)	null	Anchors Panel to a context element; format: `[el contextEl, s panelCorner, s contextCorner]` with corners defined as "tr" for "top right" and so on.
fixedcenter (b)	false	Automatically center Panel in viewport?
width (s)	null	Sets "width" style property.
height (s)	null	Sets "height" style property.
zindex (int)	null	Sets "z-index" style property.
constrainto viewport (b)	false	When true, prevents the Panel from being dragged out of the viewport.
underlay (s)	"shadow"	Type of underlay: "shadow", "none", or "matte".
effect (obj)	null	Object defining effect (FADE or SLIDE) to use in showing and hiding Panel: `{effect: YAHOO.widget.ContainerEffect.FADE, duration:1}`

YAHOO.widget.Panel: Properties

body (el)
element (el) containing header, body & footer
footer (el)
header (el)
id (s) of the element

YAHOO.widget.Panel: Methods

appendToBody(el *element*)
appendToFooter(el *element*)
appendToHeader(el *element*)
hide()
render([el *element*])
Argument required for Panels not built from existing markup. Panel will not be in the DOM or visible until render is called
setBody(str or el *content*)
setFooter(str or el *content*)
setHeader(str or el *content*)
show()
bringToTop()

Dependencies

Panel requires the full Container package, the YAHOO object, Event, and Dom. Animation, and Drag and Drop are optional. **Note:** Panels use Drag and Drop by default — a simple Panel with default configuration options will not be draggable without it.

YUI Library: Profiler

2007-12-4 **v2.4**

Simple Use Case: Profiler Object

To use Profiler, register your target functions with the `YAHOO.tool.Profiler` object and then call the function as you would normally:

```
var object = {
    method: function(){
    }
};
//register the function
YAHOO.tool.Profiler.registerFunction(
    "object.method", object);
//call the function
object.method();
```

Usage: YAHOO.registerFunction()

`YAHOO.tool.Profiler.registerFunction(str name[, obj owner])`

Arguments:

(1) **name:** A string containing the fully-qualified name of the function (e.g. `myobject.mymethod`). Profiler knows to extract everything after the last dot as the short function name.

(2) **owner:** The object that owns the function (e.g. `myobject` for `myobject.mymethod`). This argument may be safely omitted if the *name* exists in the global scope.

Note: Only functions that exist on objects can be profiled. Global functions are considered properties of the window object, so they can be registered but functions declared inside of other functions cannot be registered unless attached to an object.

Usage: YAHOO.registerConstructor()

`YAHOO.tool.Profiler.registerConstructor(str name[, obj owner])`

Arguments:

(1) **name:** A string containing the fully-qualified name of the constructor (e.g. `YAHOO.widget.Menu`). Profiler knows to extract everything after the last dot as the short constructor name.

(2) **owner:** The object that owns the function (e.g. `YAHOO.widget` for `YAHOO.widget.Menu`). This argument may be safely omitted if the *name* exists in the global scope.

Note: Only constructors that exist on objects can be profiled. Global functions are considered properties of the window object, so they can be registered but functions declared inside of other functions cannot be registered unless attached to an object.

Function Report Object

When `YAHOO.tool.getFunctionReport()` is called, an object with the following properties is returned.

Member	Type	Description
avg	float	The average amount of time (in milliseconds) that the function took to execute.
calls	int	The number of times that the function was called.
min	float	The average amount of time (in milliseconds) that the function took to execute.
max	float	The average amount of time (in milliseconds) that the function took to execute.
points	float[]	An array containing the actual execution times (in milliseconds) of the function.

Usage: YAHOO.registerObject()

`YAHOO.tool.Profiler.registerObject(str name[, obj object[, bool recurse]])`

Use `registerObject` to register all of the methods on an object (use `registerFunction` to register a single method).

Arguments:

(1) **name:** A string containing the fully-qualified name of the object (e.g. `myobject` or `YAHOO.util.Dom`).

(2) **object:** The object represented by the *name*. This argument may be safely omitted if the *name* exists in the global scope.

(3) **recurse:** Indicates if object properties should also be should also have their methods registered.

Solutions

The basic use case of Profiler is to register one or more functions, run the application as you normally would, retrieve information about specific functions (or a complete report), and then unregister the functions (a necessary step to clean up memory if you wish to persist the browser session).

```
//register the function
YAHOO.tool.Profiler.registerFunction(
    "object.method", object);
//call the function
object.method();
//get specific function information
var calls =
   YAHOO.tool.Profiler.getCallCount("object.method");
var avg = YAHOO.tool.Profiler.getAverage("object.method");
var min = YAHOO.tool.Profiler.getMin("object.method");
var max = YAHOO.tool.Profiler.getMax("object.method");
//get all function information
var report =
   YAHOO.tool.Profiler.getFunctionReport("object.method");
```

YAHOO.tool.Profiler Registration Methods

registerConstructor(string *name*, func *owner*) registers a constructor for profiling

registerFunction(string *name*, func *owner*) registers a function for profiling

registerObject(string *name*, obj *object*, bool *recurse*) registers all methods on an object for profiling

unregisterConstructor(string *name*) unregisters a constructor that was previously registered

unregisterFunction(string *name*) unregisters a function that was previously registered

unregisterObject(string *name*) unregisters all methods on an object that were previously registered

YAHOO.tool.Profiler Reporting Methods

getAverage(str *name*) returns the average amount of time (in ms) the function with the given name took to execute

getCallCount(str *name*) returns the number of times that the given function was called

getMax(str *name*) returns the maximum amount of time (in ms) the function with the given name took to execute

getMin(str *name*) returns the minimum amount of time (in ms) the function with the given name took to execute

getFunctionReport(str *name*) returns an object containing all information about a given function including call count and average, min, and max calls times

getFullReport(func *filter*) returns an object containing profiling information for all registered functions

Dependencies

Profiler requires only the YAHOO Global Object.

YUI Library: Rich Text Editor [beta]

2007-12-4 **v2.4**

Simple Use Case: YAHOO.widget.Editor

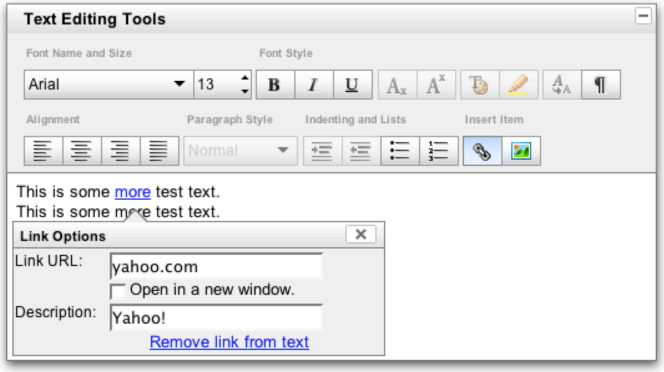
Markup:

```
<body class="yui-skin-sam">
<textarea id="msgpost">Preloaded HTML goes here.
</textarea>
</body>
```

Script:

```
var oEditor = new YAHOO.widget.Editor('msgpost',
   {
     height: '300px',
     width: '500px'
});
oEditor.render();
```

Creates an Editor instance with default configurations.



Constructor: YAHOO.widget.Editor

`YAHOO.widget.Editor(str | el ref container[, obj config])`

Arguments:

(1) **Container element:** <textarea> element or element id for the <textarea> that will be transformed into a Rich Text Editor.

(2) **Configuration object:** When instantiating an Editor, you can pass all configurations in as an object argument or configure the instance after instantiation. See Configuration Options section for common configuration object members.

Dependencies

Editor: Yahoo, Dom, Event, Element, ContainerCore; Animation, Menu and Button are optional. **SimpleEditor:** YAHOO, Dom, Event, and Element; Animation and ContainerCore are optional.

Interesting Moments in Rich Text Editor & Toolbar

See online docs for complete list of Rich Text Editor and Toolbar events.

Event	Fires...
editorContentLoaded	Fires after the editor iframe's document fully loads.
editorMouseUp, editorMouseDown, editorDoubleClick, editorKeyUp, editorKeyDown	Fires in response to the corresponding Dom event.
beforeExecCommand, afterExecCommand	Fires at the beginning/end of the execCommand process. Reference YAHOO.util.Element.html#addListener for more details.
beforeOpenWindow, afterOpenWindow	Fires before/after an editor window is opened.
closeWindow	Fires after an editor window is closed.
toolbarExpanded, toolbarCollapsed	Fires when toolbar is expanded/collapsed via the collapse button.
buttonClick	Fires when a toolbar button receives a click event.

All Editor events are Custom Events (see Element docs); subscribe to these events using their subscribe method: `oEditor.on('afterNodeChange',fnMyHandler);`.

Key Rich Text Editor Configuration Options

See online docs for complete list of Rich Text Editor configuration options.

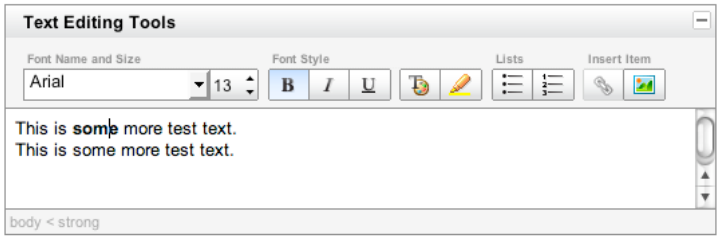
Option (type)	Default	Description
height, width	best guessed size of textarea	The height/width of the editor iframe container not including the toolbar.
animate	false	Indicates whether or not the editor should animate movements.
disabled	false	Toggle for the editor's disabled state. When disabled, design mode is off and a mask is placed over the iframe so no interaction can take place.
dompath	false	Toggles the display of the current Dom path below the editor.
toolbar	See editor.js.html	The default toolbar config.
handleSubmit	false	When true, the editor will attempt to attach a submit listener to the parent form that would trigger the editor's save handler and place the new content back into the textarea before the form is submitted.

Editor options can be set in the constructor's second argument (eg, `{height: '300px'}`) or at runtime via `set` (eg, `oEditor.set("height", "300px");`).

Constructor: YAHOO.widget.SimpleEditor

`YAHOO.widget.SimpleEditor(str | el ref container[, obj config])`

Creates a SimpleEditor instance with default configurations. SimpleEditor is a lighter version of the Editor Control.



YAHOO.widget.Editor: Methods

render() Causes the toolbar and the editor to render and replace the textarea.

setEditorHTML(string *html*) Loads HTML into the editor's body.

getEditorHTML() Returns the unprocessed HTML from the editor.

saveHTML() Cleans the HTML with the cleanHTML method and places the string into the textarea.

cleanHTML(string *html*) Processes the HTML with a few regexes to clean it up and stabilize the output.

clearEditorDoc() Clears the editor doc.

destroy() Destroys the editor along with all of its elements and objects.

toString() Returns a string representing the Editor.

nodeChange() Handles toolbar setup, getting the Dom path, and fixing nodes.

execCommand(str *command*[, str *arg*]) Levels the differences in the support by various browsers of execCommand actions.

YAHOO.widget.Toolbar: Methods

addButton(obj *config*) Add a new button to the toolbar.

addButtonGroup(obj *config*) Adds a new button group to the Toolbar.

addButtonToGroup(obj *config*) Adds a new button group to a toolbar group.

addSeparator() Adds a new button separator to the toolbar.

getButtonByValue(str | obj *command*) Gets a button instance or a menuitem instance from the toolbar by its value.

disableButton(str | number | obj *button*) Disables a button in the toolbar.

enableButton(str | number | obj *button*) Enables a button in the toolbar.

selectButton(str | number | obj *button*) Selects a button in the toolbar.

deselectButton(str | number | obj *button*) Deselects a button in the toolbar.

YUI Library: Selector Utility

2007-12-4c **v2.4**

Usage: query()

Use `query` to select one or more DOM elements based on a simple selector string. The `query` method is used to return *all* nodes that match your criteria unless the `firstOnly` arg is true.

```
var matchingNodes =
   YAHOO.util.Selector.queryAll("ul li a",
   "itemList");
```

Note: Will return all anchor elements within list-items of unordered lists who are descendants of the element whose id attribute is "itemList".

Usage: YAHOO.util.Selector.query()

```
YAHOO.util.Selector.queryAll(string selector[,
   node | string startingNode, bool firstOnly])
```

Arguments:

(1) **selector:** A string representing the CSS selector you want to target.

(2) **startingNode:** The node at which to begin the search (defaults to *document*). Be as specific as possible in choosing your startingNode to maximize performance.

(3) **firstOnly:** Whether or not to return only the first match.

Returns:

(1) **Matching Node(s):** An array of nodes that match your selector criteria. If `firstOnly` is true, this returns a single node or null if no match.

Usage: YAHOO.util.Selector.filter()

```
YAHOO.util.Selector.filter(arr | nodeset nodes,
   string selector)
```

Arguments:

(1) **nodes:** A nodeList or an array of nodes from which you want to select specific nodes that match your criteria.

(2) **selector:** A CSS selector against which you want to test and filter the *nodes*.

Usage: YAHOO.util.Selector.test()

```
YAHOO.util.Selector.test(str | elRef node,
   string selector)
```

Arguments:

(1) **node:** A node to test

(2) **selector:** A CSS selector against which you want to test ther the *node*.

Note: returns `true` if the *node* matches the *selector*, otherwise `false`.

Pseudo-classes

The Selector Utility supports the use of the pseudo-classes listed here; for more info on these, see the W3C Selectors working draft (http://www.w3.org/TR/css3-selectors/#pseudo-classes).

Pseudo-class	Description
:root	The root of the document; in HTML 4.x, this is the HTML element.
:nth-child(*an+b*)	Starting from the *b*th child, match every *a*th element.
:nth-last-child(*an+b*)	An element that has *a*n + *b* siblings after it.
:nth-of-type(*an+b*)	An element that has *a*n + *b* siblings before it that share the same element name.
:nth-last-of-type(*an+b*)	An element that has *a*n + *b* siblings after it that share the same element name.
:first-child	Same as :nth-child(1) — the first child of a given element.
:last-child	Same as :nth-last-child(1) — the last child of a given element.
:first-of-type	Same as :nth-of-type(1) — the first child of a given element with a given element name.
:last-of-type	Same as :nth-last-of-type(1) — the last child of a given type of the specified element.
:only-child	An element who is the only child of its parent node.
:only-of-type	An element whose element name is not shared by any sibling nodes.
:empty	An element that has no children.
:not()	The negation pseudo-class; takes a simple selector as an argument, representing an element not represented by the argument.
:contains()	An element whose textual contents contain the substring provided in the argument.
:checked	A radio button or checkbox that is in a checked state.

Notes regarding (an+b) notation:
Starting from the *bth* child, match every *ath* element. For example, "nth-child(2n+1)" starts from the first element and returns every other element. The "odd" and "even" keywords are supported, so "2n+1" is equivalent to "odd". "1n+2" and "n+2" are equivalent. "nth-child(0n+3)" is equivalent to "nth-child(3)". Zero value means no repeat matching, thus only the first *bth* element is matched. "3n+0" is equivalent to "3n".

Attribute Operators

att=val	equality	**att^=val**	value starts with *val*
att!=val	inequality	**att$=val**	value ends with *val*
att~=val	value matches one of space-delimited words in *val*	**att*=val**	value contains at least one occurrence of *val*
att|=val	value starts with *val* or *val-*	**att**	test for the existence of the attribute

Solutions

```
Selector.query("#nav ul:first-of-type > li:not(.selected)");  //
   Starting from the first "ul" inside of "nav" , return all "li"
   elements that do not have the "selected" class.

Selector.query("ul:first-of-type > li.selected", "nav", true);  //
   Starting from the first "ul" inside of "nav" , return the first
   "li" element that has the "selected" class.

Dom.addClass(Selector.query("#data tr:nth-child(odd)"), "odd" ) //
   add the class "odd" to all odd rows within the "data" element.
```

YAHOO.util.Selector Methods

query(string *selector*[, node | string *startingNode*, bool *firstOnly*]) the startingNode can be passed in as a string element ID or as an element reference and defaults to the document element; returns an array of matching nodes

filter(arr | nodeList *nodes*, string *selector*) returns any *nodes* that match the *selector*

test(str | elRef *node*, string *selector*) returns boolean indicating whether the *node* matches the *selector* criteria

Combinators

The Selector Utility supports the following four combinators:

" "	*Descendant Combinator:* "A B" represents an element B that has A as an ancestor.
>	*Child Combinator:* "A > B" represents an element B whose parent node is A.
+	*Direct Adjacent Combinator:* "A + B" represents an element B immediately following a sibling element A.
~	*Indirect Adjacent Combinator:* "A ~ B" represents an element B following (not necessarily immediately following) a sibling element A.

Dependencies

The Selector Utility requires only the YAHOO Global Object.

YUI Library: Slider

2007-12-4 **v2.4**

Simple Use Case

Markup:

```
<div id="sliderbg">
   <div id="sliderthumb"><img src="thumbimg"></div>
</div>
```

Script:

```
var slider =
   YAHOO.widget.Slider.getHorizSlider("sliderbg",
   "sliderthumb", 0, 200);
```

Creates a horizontal Slider within the `sliderthumb` div that can move 0 pixels left and 200 pixels to the right.

Constructor: YAHOO.widget.Slider

```
YAHOO.widget.Slider.getHorizSlider(str bgid, str
   thumbid, int lft/up, int rt/dwn[, int tick]);
```

Arguments for Horizontal and Vertical Sliders:

(1) **Background element ID:** HTML ID for the slider's background.
(2) **Thumb element ID:** HTML ID for the thumb element.
(3) **Left/Up:** The number of pixels the thumb can move left or up.
(4) **Right/Down:** The number of pixels the thumb can move right or down.
(5) **Tick interval:** Number of pixels between each tick mark.

Region Sliders take four args for range: left, right, up, down.

Solutions

Create a vertical Slider with a range of 300 pixels, ticks at 10 px intervals, and an initial value of 160:

```
var slider =
   YAHOO.widget.Slider.getVertSlider("sliderbg",
   "sliderthumb", 0, 300, 10);
slider.setValue(160, true); //set to 160, skip anim
```

Create a 300x400 pixel region Slider and set the initial thumb position to 263 on the x-axis and 314 on the y-axis:

```
var slider =
   YAHOO.widget.Slider.getSliderRegion("sliderbg",
   "sliderthumb", 0, 300, 0, 400);
slider.setRegionValue(263, 314, true);
```

Assuming an instance of a horizontal Slider in variable `mySlider`, **write a handler** for its `onSlideEnd` event:

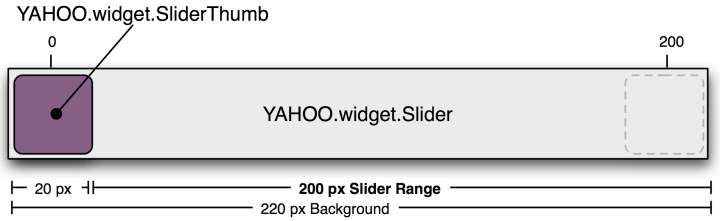
```
mySlider.subscribe("slideEnd", function() {
  alert(this.getValue()); //alerts offset from start
});
```

Interesting Moments in Slider see online docs for complete list

Event	Fires...	Arguments
slideStart	...at the **beginning** of a user-initiated change in the thumb position.	none
slideEnd	... at the **end** of a user-initiated change in the thumb position.	none
change	...each time the thumb position changes during a user-initiated move.	int *or* {x: int, y:int} offset from the starting position, one offset per slider dimension

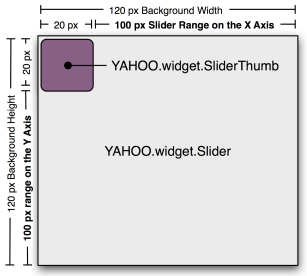
Slider events are Custom Events; subscribe to them by name using the following syntax: `mySlider.subscribe("change", fn);`.

Slider Design Considerations



A Slider is an implementation of a "finite range control." The *range* defined by the Slider is incremented in pixels. **The maximum range of a slider is the pixel-width of the Slider's background minus the width of the Slider Thumb.**

Region Sliders:



A two-dimensional Slider is referred to as a **Region Slider**. Region Sliders report two values `onChange` (x offset, y offset) and have their own method for setting value in JavaScript: `setRegionValue` takes x offset and y offset as arguments, followed by the boolean flag for skipping animation. Design considerations regarding range and thumb width apply in both vertical and horizontal dimensions.

Dependencies

Slider requires the YAHOO object, Event, Drag & Drop, Dom, and (optionally) Animation.

YAHOO.widget.Slider: Factory Methods

getHorizSlider()
getVertSlider()
getSliderRegion()

Each method returns a Slider object. See Constructor section for args list.

YAHOO.widget.Slider: Properties

animate (b)
animationDuration (n) default 0.2, roughly in seconds
keyIncrement (n) number of pixels to move slider on arrow keypress

YAHOO.widget.Slider: Methods

getValue()
getXValue()
getYValue()
lock()
setRegionValue(int *newXOffset*, int *newYOffset*, b *skipAnimation*)
setValue(int *newOffset*, b *skipAnimation*)
unlock()

YAHOO.widget.SliderThumb:

SliderThumb inherits from YAHOO.util.DD, part of the Drag & Drop library.

CSS Notes:

- Slider background should be `position:relative;`
- Slider thumb should be `position:absolute;`
- Slider thumb image should **not** be a background image

YUI Library: TabView

2007-12-4 **v2.4**

Simple Use Case: YAHOO.widget.TabView

Markup (optional, using standard module format):

```
<div id="mytabs" class="yui-navset">
  <ul class="yui-nav">
    <li><a href="…">tab one</a></li>
    <li><a href="…">tab two</a></li>
  </ul>
  <div class="yui-content">
  <div><p>Tab one content.</p></div>
  <div><p>Tab two content.</p></div>
  </div>
</div>
```

Script:

```
var myTabs = new YAHOO.widget.TabView("mytabs");
```

Creates a TabView instance using existing markup.

Constructor: YAHOO.widget.TabView

```
YAHOO.widget.TabView(str|HTMLElement|obj el[,
   obj config]);
```

Arguments:

(1) **el:** HTML ID or HTMLElement of existing markup to use when building tabView. If neither, this is treated as the Configuration object.

(2) **Configuration Object:** JS object defining configuration properties for the TabView instance. See Configuration section for full list.

Solutions

Listen for a TabView Event and make use of the Event's fields.

```
var tabView = new YAHOO.widget.TabView('demo');
var handleActiveTabChange = function(e) {
    alert(e.newValue);
};
tabView.addListener('activeTabChange',
   handleActiveTabChange);
```

Add a new Tab with with dynamic source to an existing TabView instance:

```
tabView.addTab(new YAHOO.widget.Tab({label: 'My
   Label', dataSrc: 'mySource.html',
   cacheData:true}));
```

Remove an existing tab from a TabView:

```
tabView.removeTab(tabView.getTab(1));
```

Key Interesting Moments in TabView

See online docs for a complete list of TabView's Events; see Solutions for how to access Event Fields.

Event:	Event Fields:
available	type (s), target (el)
beforeActiveTabChange	type (s), prevValue (Tab), newValue (Tab)
contentReady	type (s), target (el)
activeTabChange	type (s), prevValue (Tab), newValue (Tab)

All TabView events are Custom Events (see Event Utility docs); subscribe to these events using "addListener": (e.g. `myTabs.addListener('activeTabChange", fn);`).

Key Interesting Moments in Tab

See online docs for a complete list of Tab's Events; see Solutions for how to access Event Fields.

Event:	Event Fields:
beforeContentChange	type (s), prevValue (s), newValue (s)
beforeActiveChange	type (s), prevValue (Tab), newValue (Tab)
contentChange	type (s), prevValue (s), newValue (s)
activeChange	type (s), prevValue (Tab), newValue (Tab)

All TabView events are Custom Events (see Event Utility docs); subscribe to these events using `addListener` (e.g. `myTabs.addListener('activeChange", fn);`).

Key TabView Configuration Options

See online docs for complete list of TabView options.

Option (type)	Default	Description
activeTab (Tab)	null	The currently active Tab.
orientation	"top"	The orientation of the Tabs relative to the TabView. ("top", "right", "bottom", "left")
element	null	HTMLElement bound to TabView

TabView options can be set in the constructor's second argument (eg, `{activeTab: tabInstance}`) or at runtime via `set` (eg, `myTabs.set("activeTab", tabInstance);`).

Key Tab Configuration Options

See online docs for complete list of Tab configuration options.

Option (type)	Default	Description
active (b)	false	Whether or not the Tab is active.
disabled (b)	false	Whether or not the Tab is disabled.
label (s)	null	The text (or innerHTML) to use as the Tab's label.
content (s)	null	The HTML displayed when the Tab is active.
labelEl (el)	null	The HTMLElement containing the `label`.
contentEl (el)	null	The HTMLElement containing the `contentEl`.
dataSrc (s)	null	Url to use for retrieving content.
cacheData (b)	false	Whether or not data retrieved from `dataSrc` should be cached or reloaded each time the Tab is activated.
Element (el)	null	HTMLElement bound to Tab

Tab options can be set in the constructor's second argument (eg, `{disabled: true}`) or at runtime via `set` (eg, `myTab.set("disabled", true);`).

YAHOO.widget.TabView: Properties

CLASSNAME
TAB_PARENT_CLASSNAME
CONTENT_PARENT_CLASSNAME

YAHOO.widget.TabView: Methods

addTab(Tab)
removeTab(Tab)
getTab(i)
getTabIndex(Tab)
contentTransition()
set(option, value)
get(option)

YAHOO.widget. Tab: Properties

LABEL_TAGNAME
ACTIVE_CLASSNAME
DISABLED_CLASSNAME
LOADING_CLASSNAME

YAHOO.widget. Tab: Methods

set(option, value)
get(option)

Dependencies

TabView requires the YAHOO object, Event, Dom, and Element.

YUI Library: TreeView

2007-12-4 **v2.4**

Simple Use Case

```
var tree = new YAHOO.widget.TreeView("treeDiv1");
var root = tree.getRoot();
var tmpNode = new YAHOO.widget.TextNode("mylabel",
   root, false);
tree.draw();
```

Places a Tree control in the HTML element whose ID attribute is "treediv1"; adds one node to the top level of the Tree and renders.

Constructor: YAHOO.widget.TreeView

```
YAHOO.widget.TreeView(str | element target);
```

Arguments:

(1) **Element id or reference:** HTML ID or element reference for the element being into which the Tree's DOM structure will be inserted.

Nodes: TextNode, MenuNode, HTMLNode

TextNode (for simple labeled nodes):

```
YAHOO.widget.TextNode(obj | str oData, Node obj
   oParent[, b expanded]);
```

Arguments:

(1) **Associated data:** A string containing the node label or an object containing str `label`, str `href`, and any other custom members desired. If no `oData.href` is provided, clicking on the TextNode's intrinsic `<a>` tag will invoke the node's `expand` method.

(2) **Parent node:** The node object of which the new node will be a child; for top-level nodes, the parent is the Tree's root node.

(3) **Expanded state:** A boolean indicating whether the node is expanded when the Tree is rendered.

MenuNode (for auto-collapsing node navigation):

MenuNodes are identical to TextNodes in construction and behavior, except that only one MenuNode can be open at any time for a given level of depth.

HTMLNode (for nodes with customized HTML for labels):

```
YAHOO.widget.HTMLNode(obj | str HTML, Node obj
   oParent[, b expanded,b hasIcon]);
```

Arguments:

(1) **HTML:** A string containing markup for the node's label; no event handlers are provided by default for this markup.

(2) **Parent node:** See TextNode.

(3) **Expanded state:** See TextNode.

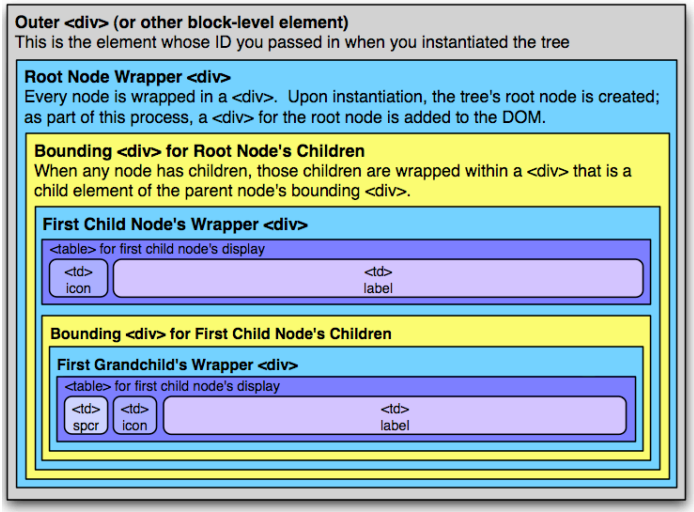
(4) **Has Icon:** Stipulates whether the expanded/contracted icon (and its horizontal space) should be rendered for this node.

Interesting Moments in TreeView see docs for complete list

Event	Fires...	Arguments
expand	...before a node expands; return false to cancel.	Node obj *expanding node*
collapse	...before a node collapses; return false to cancel	Node obj *collapsing node*
labelClick	...when text label clicked	Node obj *clicked nd*

TreeView events are Custom Events; subscribe to them by name using the following syntax: `tree.subscribe("expand", fn);`.

TreeView DOM Structure



Solutions:

Dynamically load child nodes:

```
fnLoadData = function(oNode, fnCallback) {
 //create child nodes for oNode
 var tmp = new YAHOO.widget.TextNode("lbl", oNode);
 fnCallback(); //then fire callback}
var tree = new Yahoo.widget.TreeView(targetEl);
tree.setDynamicLoad(fnLoadData);
var root = tree.getRoot();
var node1 = new YAHOO.widget.TextNode("1st", root);
tree.draw();
```

Dependencies

TreeView requires the YAHOO global object and the Event Utility.

YAHOO.widget.TreeView: Properties

id (str)
nodeCount (int)

YAHOO.widget.TreeView: Methods

collapseAll()
draw()
expandAll()
getNodesByProperty()
getRoot()
popNode(node) returns detached node, which can then be reinserted
removeChildren(node)
removeNode(node, b autorefresh)
setDynamicLoad(fn)

YAHOO.widget.Node: Properties

Inherited by Text, Menu, & HTML nodes

data (obj)
expanded (b)
hasIcon (b)
href (str)
iconMode (i)
labelStyle (s) Text/MenuNodes only. Use to style label area, e.g. for custom icons. Use `contentStyle` property for HTMLNodes
nextSibling (node obj)
parent (node obj)
previousSibling (node obj)
target (str)
tree (TreeView obj)

YAHOO.widget.Node: Methods

Inherited by Text, Menu, & HTML nodes

appendTo()
collapse()
collapseAll()
expand()
expandAll()
getEl() returns node's wrapper element
getHTML() includes children
getNodeHTML() sans children
hasChildren()
insertBefore()
insertAfter()
isDynamic()
isRoot()
setDynamicLoad()
toggle()

YUI Library: The YAHOO Global Object

2007-12-4 **v2.4**

Simple Use Case: YAHOO Object

In its simplest usage, the YAHOO global object requires no implementer action; it serves as a container and provider of utility methods to all other components of the YUI Library.

Usage: YAHOO.namespace()

```
YAHOO.namespace(str namespace)
```

Arguments:

(1) **namespace:** A string containing a single namespace (e.g. "myproduct") or a deeper namespace (e.g. "myproduct.weatherModule"). Namespace objects are created within the YAHOO object.

Note: Be careful when naming packages. JavaScript reserved words may work as property names in some browsers and not others.

Usage: YAHOO.lang.augmentObject()

```
YAHOO.lang.augmentObject(fn receiver, fn
   supplier[, str property1, str property2, … ,
   str propertyn])
```

Arguments:

(1) **receiver:** The object to be augmented.

(2) **supplier:** The object serving as the source of the augmentation.

(3-n) **properties:** By default, YAHOO.lang.augmentObject will apply all members of the supplier object to the receiver if the receiver doesn't already have them; arguments 3 through *n* can be used to supply string member names that designate the specific members to be augmented from the supplier to the receiver.

Note: The default operation, in which all of the supplier's members are applied to the receiver, `YAHOO.lang.augment` will avoid overwriting existing members on the receiver. If you specify supplier members to use for augmentation (via arguments 3 through *n*), the augmentation will overwrite those members if they already exist on the receiver.

Usage: YAHOO.lang.augmentProto()

```
YAHOO.namespace(fn receiver, fn supplier[, str
   property1, str property2, … , str propertyn])
```

This function is symmetrical with `YAHOO.lang.augmentObject` (see above); however, it augments only from the supplier's prototype to the receiver's prototype. Instance members are not copied from the supplier to the receiver.

Dependencies

The YAHOO Global Object is a dependency for all YUI components; it has no dependencies of its own.

YAHOO Object: Default Members

See online docs for complete documentation on each default member of the YAHOO object.

Member	Type	Description
env	object	Environment object. Contains information about what YUI modules are loaded and provides a method for obtaining version information.
example	object	An empty object used as a namespace for example implementations.
lang	object	Contains utility methods. Full list at right.
util	object	Namespace for YUI utilities. Do not add your own members to this object
tool	object	Namespace for developer tools like YUITest. Do not add your own memebers to this object.
widget	object	Namespace for YUI controls (widgets). Do not add your own members to this object.
log	method	Calls `YAHOO.widget.Logger.log`; prevents log messages from throwing errors when the Logger Control is not present.
register	method	Registers a module with the YAHOO object.

Usage: YAHOO.lang.extend()

```
YAHOO.extend(obj subclass, obj superclass[, obj
   overrides])
```

Arguments:

(1) **subclass:** The object you're using to extend the base object.

(2) **superclass:** The base object being extended by the "subclass".

(3) **overrides:** An object whose members will be added to the subclass prototype, overriding members of the same name if they exist on the superclass prototype.

Solutions

`YAHOO_config` is not included as part of the YUI library. Instead it is an object that can be defined by the implementer immediately before including the YUI library. Use `YAHOO_config` to set up a listener that fires when YUI components are loaded:

```
var YAHOO_config = {
 listener: function(moduleInfo) {
  //executes when any YUI module loads, including YAHOO
   object
 }
}
```

Note: See Module Info table at right for the format of object passed to your listener function.

Get version information for a YUI component that has been loaded on the page:

```
var YAHOO.env.getVersion("animation"); //returns module info
   object
```

YAHOO.lang Methods

dump(obj or arr) returns string representation
isArray(any) returns boolean
isBoolean(any) returns boolean
isFunction(any) returns boolean
isNull(any) returns boolean
isNumber(any) returns boolean
isObject(any) returns boolean
isString(any) returns boolean
isUndefined(any) returns boolean
hasOwnProperty(obj, property) returns boolean
augmentObject() general mixin function
augmentProto() see usage section
extend() see usage section
isValue(any) returns false for null/undefined/NaN, else true — note that `false` has value and returns true
merge(obj1, obj2, …) returns object with all properties of all args
substitute() see docs
trim(string) removes leading/trailing space

YAHOO.env Method:

getVersion(str *yuimodulename*) returns module info; see below

Module Info

`YAHOO.env.modules` is an object indexed by *module name*; each member contains information about a single YUI module. Module info objects contain the following information:

name	**str** *module name*
version	**str** *last loaded version*
build	**n** *last loaded build*
versions	**arr** *all loaded versions*
builds	**arr** *all loaded builds*
mainClass	**fn** *reference to main class for this module*

YUI module names: animation, autocomplete, base, button, calendar, colorpicker, connection, container, containercore, datasource, datatable, dom, dragdrop, editor, element, event, fonts, grids, history, imageloader, logger, menu, reset, slider, tabview, treeview, yahoo, yuiloader, yuitest.

YUI Library: YUI Loader Utility [beta]

2007-12-4c **v2.4**

Simple Use Case: YAHOO.util.YUILoader

Markup:

```
<script src="yuiloader-beta.js"></script>
```

Script:

```
//instantiate Loader:
loader = new YAHOO.util.YUI Loader();

//identify the components you want to load:
loader.require("colorpicker", "treeview");

//configure the Loader instance
loader.loadOptional = true;

//Load files using the insert() method. Insert() takes an optional
//configuration object, and in this case we are setting up an onSuccess
//callback. Your callback will be executed once all required files are
//loaded.
loader.insert({ onSuccess: function() {
    //this is your callback function; you can use
    //this space to call all of your instantiation
    //logic for the components you just loaded.
}});
```

Sets up a YUI Loader instance, configures it to load Color Picker and TreeView, and then executes the load.

Constructor: YAHOO.util.YUILoader

`YAHOO.util.YUILoader(obj config)`

Arguments:

(1) **Configuration object:** When instantiating YUI Loader, you can pass all configurations in as an object argument or configure the instance after instantiation. See Configuration Options section for common configuration object members.

Using YUI Loader to Load Non-YUI Files

See online docs for full syntax and example using `addModule()`.

YUI Loader's `addModule` method can be used to extend YUI Loader to add non-YUI modules. `addModule` takes an object argument with the following members:

name (s)	String modulename.
type (s)	String moduletype (eg, "js" or "css").
path (s)	Path to source file, including file name; will be prefixed with instance's `base` path setting.
fullpath (s)	Full URI to module file. Supersedes `path`.
varName (s)	If module is JavaScript, a variable name (as string) that will be defined by the loaded script; alternatively, use `YAHOO.register`.
requires (arr)	Array of required dependencies, with each member a string module name.
optional (arr)	Array of optional dependencies, with each member a string module name.
skinnable (b)	Does this component have a skin CSS file (in the standard skin-CSS directory?

Key YUI Loader Configuration Options

See online docs for complete list of YUI Loader configuration options.

Option (type)	Default	Description
allowRollup (b)	true	Allow aggregate files (e.g., utilities.js) where appropriate? (improves performance)
base (s)	current build dir on yui.yahooapis.com	Base directory for YUI build.
filter (s | o)	null	Filter that can be applied to YUILoader filenames prior to loading. Use `'DEBUG'` for debug versions of YUI files
loadOptional (b)	false	Load all optional dependencies for the required components?
onFailure (f)	null	Callback function fired if an insert operation fails.
onProgress	null	Callback function fired each time a resource loads successfully.
onSuccess (f)	null	Callback function to run when loading of all required components and dependencies is complete.
require (arr)	true	Array of required YUI components, each of which is a string representing the modulename for the component.
skin (o)	by default, the YUI Sam Skin is applied to skinned components	Object which allows you to specify a global `defaultSkin` and per-component `overrides`. See User's Guide for full syntax.
varName (s)	null for `insert()`; "YAHOO" for `sandbox()`	Only needed for non-YUI scripts. This is the variable defined by the exterrnal script whose presence indicates load-completion; for `sandbox()`, this is the root variable for the loaded library.

YUI Loader options can be set in the constructor's second argument (eg, `{base: '../../'}`) or at runtime on a YUILoader instance (eg, `oLoader.base = '../../'`).

Solutions

Using YUI Loader to create a sandboxed/private YUI:

```
var loader = new YAHOO.util.YUI Loader();
loader.sandbox({
    require: ["treeview"], // what to load
    base: '../../build/',  // relative path to library files
    loadOptional: true,    // pull in optional components

    // Executed once the sandbox is successfully created:
    onSuccess: function(o) {
        var myYAHOO = o.reference; //ref to private YAHOO
        // TreeView in myYAHOO can now be used; note that
        // YAHOO.widget.TreeView may not exist!
        myYAHOO.util.Event.onAvailable("treeEl",function() {
           var tree = new myYAHOO.widget.TreeView("treeEl");
        });
    },
});
```

YAHOO.util.YUILoader: Properties

See also configuration options; all configuraton options can be treated as instance members.

inserted obj list of modules inserted by the YUILoader instance

sorted arr listed of sorted dependencies; available after insert or calculate is called

YAHOO.util.YUILoader: Methods

addModule(o) adds non-YUI module; obj argument specifies all needed metadata for new module

calculate() calculates the list of needed modules based on required components but does not insert them in the page

insert(o) calculates needed modules, inserts them, fires o.onSuccess if that is supplied

Components & Module Names

YUI Loader refers to YUI components by their unique module names — strings by which components are referenced within YUI. Here is the full list of YUI module names:

animation, autocomplete, base, button, calendar, charts, colorpicker, connection, container, container_core, datasource, datatable, dom, dragdrop, editor, element, event, fonts, get, grids, history, imageloader, json, logger, menu, profiler, reset, selector, slider, tabview, treeview, yahoo, yuiloader, yuitest.

Dependencies

YUI Loader does not have any required dependencies; it can be used on pages with no YUI content present or to bring additional YUI components onto pages where YUI is already being used. For some use cases, you may need to include the Yahoo Global Object first; see docs for more on this.

YUI Library: YUI Test Utility [beta]

2007-12-4b **v2.4**

Simple Use Case: TestCase Object

Create a TestCase object with desired dests, add your TestCase to the TestRunner object, and run the test:

```
//set up a test case:
var oTestCase = new YAHOO.tool.TestCase({
  name: "Simple Math",
  testEquality: function () {
    YAHOO.util.Assert.areEqual(4, (2+2), "2+2=4");
  };
});

//add the test case to the TestRunner:
YAHOO.tool.TestRunner.add(oTestCase);

YAHOO.tool.TestRunner.run(); //run the test
```

Key Members of the TestCase Object

name	The name of the TestCase. This will be visible in the logging of TestCase events.	
test*name*	A function that tests functional code via one or more assertions; name must begin with the string "test". A TestCase can have one or more test members.	
setUp	Method that prepares your test case to run by, for example, creating needed data constructs or objects.	
tearDown	Method that nulls out variables in use by the TestCase, detaches event handlers, etc.	
_should	Special object that provides granular configuration of the test case. It can have the following members:	
	ignore	A name:value pair consisting of a testname and Boolean indicating whether to ignore the test (e.g.: `ignore: {testOne : true /*ignore testOne*/}`.

	error	A name:value pair consisting of a testname and Boolean indicating whether the test is should fail:
		`error: {testTwo : true /*testTwo should fail*/}`
		Or, a specific error string can substitute for the Boolean:
		`error: {testTwo : "Expected string." /*testTwo should fail with this specific error message*/}.`

All TestCase configurations should be passed into the TestCase constructor as an object literal; see Simple Use Case.

Key Events in YUITest

See online docs for full list of custom events, including TestSuite- and TestRunner-level events.

All YUITest events are subscribed to at the TestRunner level; e.g.: `YAHOO.tool.TestRunner.subscribe(YAHOO.tool.TestRunner.TEST_FAIL_EVENT, myFn);`.

Test-level Events

Event:	Fires:
TEST_PASS_EVENT	When an individual test passes.
TEST_FAIL_EVENT	When an individual test fails.
Argument Data Object for Test-level Events:	
type	Type of event.
testCase	The testCase object to which this test belongs.
testName	The string name of this test.

TestCase-level Events

Event:	Fires:		
TEST_CASE_BEGIN_EVENT	Before the TestCase is run.		
TEST_CASE_COMPLETE_EVENT	After the TestCase is run.		
Argument Data Object for TestCase-level Events:			
type	Type of event.		
testCase	Current TestCase instance.		
results (TEST_CASE_END event only)	**passed**	Number of tests that passed.	
	failed	Number of tests that failed.	
	testname	**result**	`pass` or `fail`.
		message	String returned by the test.

Note: TestSuite- and TestRunner-level events are also available, containing summary data in addition to specific TestCase results objects. See online docs for full details.

Assertions

Assertions are accessed via `YAHOO.util.Assert`

Equality assertions:
areEqual(expected, actual)
areNotEqual(expected, actual)
equivalent to == test

Sameness assertions:
areSame(expected, actual)
areNotSame(expected, actual)
equivalent to === test

Data-type assertions:
isArray(arg)
isBoolean(arg)
isFunction(arg)
isNumber(arg)
isObject(arg) object and function return true
isString(arg)

isTypeOf assertion:
isTypeOf(sType, sTest, sFailureMessage)

```
YAHOO.util.Assert.isTypeOf("string", 5, "Expected string.");
//fails
```

isInstanceOf assertion:
isInstanceOf(oConstructor, oTestObject, sFailureMessage)

```
YAHOO.util.Assert.isInstanceOf(String, "Madrone", "Expected string."); //passes
```

can be used to test non-native objects, too

Special Value Assertions:
isFalse, isTrue, isNaN, isNotNaN, isNull, isNotNull, isUndefined, isNotUndefined

```
YAHOO.util.Assert.isNull(7, "Expected null."); //fails
```

YAHOO.tool.TestSuite Methods:

add(obj *testCase* or *testSuite*) adds a testCase or testSuite object to a testSuite

YAHOO.tool.TestRunner Methods:

add(obj *testCase* or *testSuite*) adds a testCase or testSuite object to the list of items to run
clear() removes all test objects from the runner
run() runs all testCase and testSuites currently queued in the TestRunner

YAHOO.tool.TestCase Methods:

wait([fn *segment*, int *delay*]) causes the TestCase to delay the specified number of milliseconds before *segment* is executed
resume([fn *segment*]) resmes a paused test; if *segment* is omitted, the test automatically passes

YAHOO.util.UserAction Methods:

click(obj *target*, *obj* options) simulates a click on the target
mousedown(obj *target*, *obj* options) simulates a mousedown on the target
mouseup(obj *target*, *obj* options) simulates a mouseup on the target
mouseover(obj *target*, *obj* options) simulates a mouseover on the target
mouseout(obj *target*, *obj* options) simulates a mouseout on the target
mousemove(obj *target*, *obj* options) simulates a mousemove on the target
keydown(obj *target*, *obj* options) simulates a keydown on the target
keyup(obj *target*, *obj* options) simulates a keyup on the target
keypress(obj *target*, *obj* options) simulates a keypress on the target

Dependencies

The YUI Test Utility requires the Yahoo Global Object, Dom Collection, Event Utility, and Logger Control. The Logger's CSS file and the YUI Test CSS file are also required.