# YUI Library: Animation

2011-3-21 **v2.9**

## 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

2011-3-21 **v2.9**

## Simple Use Case

**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

```
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.

## Solutions

**Custom cell formatting**:

```
myAC.resultsTypeList = false; // pass data as an object
myAC.formatResult = function(oData, sQuery, sMatch) {
   return (sMatch + "(" + oData.param + ")" );
}
```

**Custom local filtering**:

```
myAC.applyLocalFilter = true; // pass results thru filter
myAC.filterResults = function(sQuery, oFullResponse,
   oParsedResponse, oCallback) {
   var matches = [], matchee;
   for(var i=0; i<oParsedResponse.results.length; i++) {
      if(oParsedResponse.results[i].someValue > 0) {
             matches[matches.length] =
                 oParsedResponse.results[i]
      }
   }
   oParsedResponse.results = matches;
   return oParsedResponse;
}
```

## Interesting Moments

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

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

## Abstract Methods

| Method | Description |
|---|---|
| doBeforeLoadData | This overridable abstract method gives implementers access to the DataSource response before it is consumed by the AutoComplete instance and rendered into the results container. |
| doBeforeExpandContainer | This overridable abstract method gives implementers access to result data and DOM elements after the container has been rendered with results but before it is displayed to the user, for example to move the container to a different position on the screen. |

## Dependencies

AutoComplete requires the YAHOO Global Object, Dom, and Event, and DataSource. Animation (for animated opening of the suggestion container) is optional.

## YAHOO.widget. AutoComplete Key Properties:

**alwaysShow Container** (b)
**animHoriz** (b)
**animSpeed** (int)
**animVert** (b)
**applyLocalFilter** (b)
**autoHighlight** (b)
**delimChar** (char || array)
**forceSelection** (b)
**highlightClassName** (string)
**maxResultsDisplayed** (int)
**minQueryLength** (int)
**prehighlightClass Name** (string)
**queryDelay** (int)
**queryMatchCase** (b)
**queryMatchContains** (b)
**queryMatchSubset** (b)
**queryQuestionMark** (b)
**resultsTypeList** (b)
**suppressInputUpdate** (b)
**typeAhead** (b)
**typeAheadDelay** (int)
**useIFrame** (b)
**useShadow** (b)

# YUI Library: Button

2011-3-21 **v2.9**

## 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 | option |

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 Button 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. (For 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. (For to buttons of type "link.") |
| menu (o) | null | Element id, array of YAHOO.widget.MenuItem configuration attributes, or YAHOO.widget.Menu instance. (For buttons of type "menu" and "split.") |
| menumaxheight (n) | 0 | The max height height of a Menu before it scrolls. |
| menualignment (a) | ["tl","bl"] | How the Menu is aligned to the Button. |

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

2011-3-21 **v2.9**

## 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]);
```

*Arguments:*

(1) **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.
(2) **container:** HTML ID of (or a reference to) an existing but empty HTML element into which the new Calendar will be inserted.
(3) **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<br>HIDE_BLANK_WEEKS | SHOW_WEEK_ HEADER<br>PAGES (CalendarGroup only) | MULTI_SELECT<br>NAVIGATOR |

## Localizing Calendar and CalendarGroup

Calendar instances can be localized via configuration options, including:

| | | |
|---|---|---|
| MONTHS_SHORT<br>MONTHS_LONG<br>LOCALE_MONTHS | WEEKDAYS_1CHAR<br>WEEKDAYS_SHORT<br>LOCALE_WEEKDAYS | WEEKDAYS_MEDIUM<br>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 Global 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: Carousel

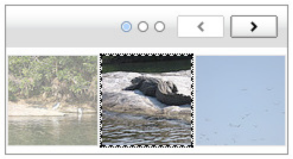
2011-3-21 **v2.9**

## Simple Use Case: YAHOO.widget.Carousel

**Styles:**
```
.yui-carousel-element li { width: Ypx; height: Zpx; }
```

**Markup:**
```
<div id="container"><ul><li>item</li></ul></div>
```

**Script:**
```
var carousel = new YAHOO.widget.Carousel("container");
carousel.render();
```



Creates a simple Carousel of items inside the list container (UL).

Carousel ships with a skin (left) that provides visual elements for the navigation controls. Apply the CSS class yui-skin-sam to the body element or other parent element of your Carousel's container in order to make use of the default skin.

## Constructor: YAHOO.widget.Carousel

```
YAHOO.widget.Carousel([str container,] [,obj config]);
```

*Arguments:*

(1) **container:** HTML ID or HTMLElement of existing markup to use when building Carousel. This should be a UL or an OL element.

(2) **Configuration Object:** JS object defining configuration properties for the Carousel instance. See Configuration section and online docs for a list of these options.

## Carousel Options

Configure options using the constructor configuration object or *set()*, as described in "Solutions"

Carousel objects include several configurable options, including:

| | | |
|---|---|---|
| animation<br>autoPlayInterval | isCircular<br>isVertical | numVisible<br>revealAmount |

## Dependencies

Carousel requires YAHOO, Dom, Event and Element.

## Interesting Moments in Carousel

See online docs for complete list of Carousel events.

| Event | Fires... | Passes back: |
|---|---|---|
| afterScroll | After the viewport in the Carousel has scrolled. | The indices of the first and last items in the viewport. |
| itemAdded | After an item has been added to the Carousel. | The content of the inserted item and the position where it is inserted. |
| pageChangeEvent | When the Carousel navigates to a new page. | Page number of the current page. |
| navigationStateChange | When the state of the navigation buttons change (enabled/disabled). | The state of the navigation buttons (true if enabled, false otherwise) |
| loadItems | When the Carousel needs more items to be loaded for displaying them. | The first and last visible items in the Carousel and the number of items to be loaded. |

All Carousel events are Custom Events (see Event Utility docs); subscribe to these events using their `subscribe` method: `carousel.loadItems.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 to which you can subscribe.

## Solutions

**Basic markup** for a Carousel:

```
<div id="container">
 <ul>
   <li><img alt="" src="..."></li>
 </ul>
</div>
```

Render a Carousel that scrolls **automatically** every two seconds and animating while scrolling:

```
carousel = new YAHOO.widget.Carousel("container", {
        autoPlayInterval: 2000, animation: { speed: 0.5 }
});
carousel.render();
carousel.startAutoPlay();
```

There are two ways to **configure options on your Carousel**:

```
// 1. In the constructor, via an object literal:
carousel = new YAHOO.widget.Carousel("container", {
        isCircular: true
});

// 2. Via "set":
carousel.set("isCircular", true);
```

## YAHOO.widget.Carousel

**containerId** (str)
**cfg** (Config)
the carousel's configuration object

## YAHOO.widget.Carousel Methods

### Manipulation:

**addItem**(String, index)
**addItems**(array)
**replaceItem**(String, index)
**replaceItems**(array)
**clearItems**()
**removeItem**(int)
**registerPagination**(String)
**updatePagination**()

### Navigation:

**scrollBackward**()
**scrollForward**()
**scrollTo**(int)

### Rendering:

**render**([String])
renders current state to page
**show**()
**hide**()

### Selection:

**set**("selectedItem", int)

### Other:

resets carousel to original state
**carousel.set**
(propName, propValue)
**carousel. get**
(propName)

YUI Library: Carousel 2011-3-21 v2.9

## Multi-row Carousel

**Styles:**

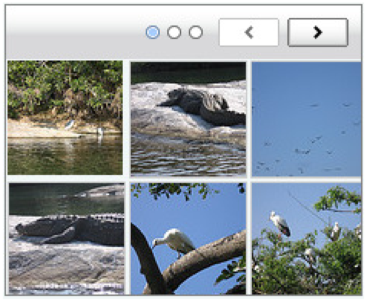
```
.yui-carousel-element li { width: Ypx; height: Zpx; }
```

**Markup:**

```
<div id="container"><ul><li>item</li></ul></div>
```

**Script:**

```
var carousel = new YAHOO.widget.Carousel("container", {
            numVisible: [ 3, 4 ]
});
carousel.render();
```



Creates a simple multi-row Carousel of items inside the list container (UL) to display items in an album view layout.

Accomplished by setting Carousel's **numVisible** property to an **array** containing the number of **columns and rows** of items to display.

Items can be applied any width or height via CSS and can contain more markup than just image tags.

Other single-row Carousel events, methods, and options are still available in multi-row including (but not limited to): **loadItems**, **animation**, **autoPlayInterval**, **isCircular**, and **isVertical**.

## Pagination Template

Render a Carousel that displays pagination data:

```
var carousel = new YAHOO.widget.Carousel("container");
carousel.registerPagination("<strong>Image</strong>
{firstVisible} - {lastVisible} <strong>of</strong>
{numItems}");
carousel.render();
```

After Carousel instantiation, we use the public **registerPagination** method to register a text or HTML template embedded with Carousel {tokens}.

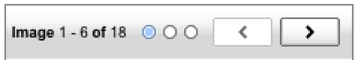


**Image** 1 - 6 **of** 18

The public **updatePagination** method automatically substitutes each token with its respective data on render and every time Carousel's state changes.

Any combination of the following tokens can be embedded in your template: **numVisible**, **numPages**, **numItems**, **selectedItem**, **currentPage**, **firstVisible**, and/or **lastVisible**.

# YUI Library: Charts

2011-3-11 **v2.9**

## Simple Use Case: YAHOO.widget.LineChart

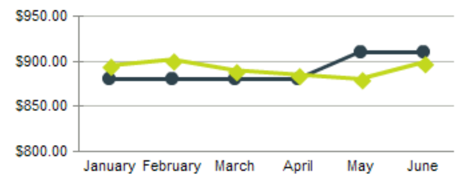
**Markup:**

```
<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)<br>args.item (Object)<br>args.index (Number)<br>args.seriesIndex (Number)<br>args.x (Number)<br>args.y (Number) |
| itemDragStartEvent, itemDragEvent, itemDragUpdateEvent | args.type (String)<br>args.item (Object)<br>args.index (Number)<br>args.seriesIndex (Number)<br>args.x (Number)<br>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)<br>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)<br>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. |
| dataSource | null | Replace the current DataSource |
| wmode | "window" | The window mode of Flash Player. May be "window", "opaque" or "transparent". |

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, DataSource Utility and SWF Utility. **Note:** *On the client, Charts requires Flash Player 9.0.45 or later.*

# YUI Library: Color Picker Control

2011-3-21 **v2.9**

## Simple Use Case: YAHOO.widget.ColorPicker

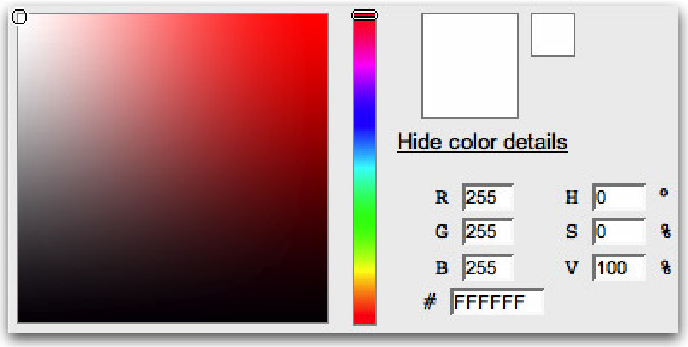
**Markup:**

```
<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 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

2011-3-21 **v2.9**

## Simple Use Case

```
var callback = {
  success: function(o) {
    YAHOO.util.Dom.get('myEl').innerHTML = o.responseText;
  }
}
var conn = 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. |
| getResponseHeader[] | Array collection of response headers and their corresponding values, indexed by header label. |
| getAllResponseHeaders | 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: Cookie Utility

2011-3-21 **v2.9**

## Simple Use Case: Reading/writing a cookie

```
var value = YAHOO.util.Cookie.get("name");
```

Retrieves the cookie named “name” and stores its value in a variable.

```
YAHOO.util.Cookie.set("name", "value");
```

Sets a cookie named “name” and stores “value” in it.

## Usage: YAHOO.util.Cookie.get()

```
var value = YAHOO.util.Cookie.get(str name[,
   func converter])
```

*Arguments:*
(1) **name:** The name of the cookie to get.
(2) **converter:** (Optional) A conversion function to run the value through before returning it. This function is not called if the cookie with the given name doesn’t exist.

## Usage: YAHOO.util.Cookie.remove()

```
YAHOO.util.Cookie.remove(str name, [, obj
   options])
```

*Arguments:*
(1) **name:** The name of the cookie to set.
(2) **options:** (Optional) Options indicating setting the accessibility of the cookie. These settings should be the same as the ones used to set the cookie.

## Usage: YAHOO.util.Cookie.set()

```
YAHOO.util.Cookie.set(str name, str value [, obj
   options])
```

*Arguments:*
(1) **name:** The name of the cookie to set.
(2) **value:** The value of the cookie to set.
(3) **options:** (Optional) Options for setting the accessibility of the cookie based on expiration date, domain, path, and security.

## Cookie Options Object

Options that can be set on the options object in remove(), set(), removeSub(), setSub(), and setSubs().

| Member | Type | Description |
|---|---|---|
| expires | Date | A Date object representing the date and time at which the cookie should expire. |
| domain | string | The domain for which the cookie should be accessible. |
| path | string | The path for which the cookie should be accessible. |
| secure | boolean | Indicates if the cookie is accessible only via SSL. |

## Usage: YAHOO.util.Cookie.getSub()

```
var value = YAHOO.util.Cookie.getSub(str name, str
   subname [, func converter])
```

*Arguments:*
(1) **name:** The name of the cookie to get.
(2) **subname:** The subcookie name to get.
(3) **converter:** (Optional) A conversion function to run the value through before returning it. This function is not called if the subcookie with the given name doesn’t exist.

## Usage: YAHOO.util.Cookie.removeSub()

```
YAHOO.util.Cookie.removeSub(str name, str subname [,
   obj options])
```

*Arguments:*
(1) **name:** The name of the cookie in which the subcookie exists.
(2) **subname:** The subcookie name to remove.
(3) **options:** (Optional) Options indicating setting the accessibility of the cookie. These settings should be the same as the ones used to set the cookie.

## Usage: YAHOO.util.Cookie.setSub()

```
YAHOO.util.Cookie.setSub(str name, str subname, str
   value [, obj options])
```

*Arguments:*
(1) **name:** The name of the cookie to set.
(2) **subname:** The subcookie name to set.
(3) **value:** The value for the subcookie.
(4) **options:** (Optional) Options for setting the accessibility of the cookie based on expiration date, domain, path, and security.

## YAHOO.util.Cookie Cookie Methods

**get(str *name*, func *converter*)** returns a cookie value
**set**(str *name*, str *value*, obj *options*) sets a cookie
**remove**(str *name*, obj *options*) removes a cookie

## YAHOO.util.Cookie Subcookie Methods

**getSub**(str *name*, str *subname*, func *converter*) returns a subcookie value
**getSubs**(str *name*) returns all subcookies in an object
**setSub**(str *name*, str *subname*, str *value*, obj *options*) sets a subcookie
**setSubs**(str *name*, obj *subcookies*, obj *options*) sets all subcookies in a cookie
**removeSub**(str *name*, str *subname*, obj *options*) removes a subcookie

## Dependencies

The Cookie utility requires only the YAHOO Global object.

# YUI Library: DataSource

2011-3-21 **v2.9**

## Simple Use Case

**Script:**

```
var myDataSource = new YAHOO.util.DataSource
    ([{name:"a",id:"1"}, {name:"b",id:"2"}]);
myDataSource.responseType =
    YAHOO.util.DataSource.TYPE_JSARRAY;
myDataSource.responseSchema = [fields:["name","id"];
```

Instantiates a new DataSource object, `myDataSource`, which manages data retrieval for use by other widgets.

## Constructors

`YAHOO.util.LocalDataSource(mixed data[, obj configurations]);`

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

`YAHOO.util.FunctionDataSource(fn function[, obj configurations]);`

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

`YAHOO.util.ScriptNodeDataSource(str uri[, obj configurations]);`

(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.util.XHRDataSource(str uri[, obj configurations]);`

(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.

`YAHOO.util.DataSource(mixed data[, obj configurations]);`

(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.

## Key Configuration Properties

| Property | Description |
|---|---|
| responseType | Determines which parsing algorithm to use on response data. |
| responseSchema | Determines what data gets parsed out of response for consumption. |

## Interesting Moments

| Event | oArgs passed to handler |
|---|---|
| cacheFlushEvent | none |
| cacheRequestEvent | oArgs.request {obj} The request object<br>oArgs.callback {obj} The callback object |
| cacheResponseEvent | oArgs.request {obj} The request object<br>oArgs.response {obj} The response object<br>oArgs.callback {obj} The callback object |
| dataErrorEvent | oArgs.request {obj} The request object<br>oArgs.callback {obj} The callback object<br>oArgs.message {str} Error message |
| requestEvent | oArgs.request {obj} The request object<br>oArgs.callback {obj} The callback object<br>oArgs.tId {int} Unique transaction ID |
| responseCacheEvent | oArgs.request {obj} The request object<br>oArgs.response {obj} The response object<br>oArgs.callback {obj} The callback object |
| responseEvent | oArgs.request {obj} The request object<br>oArgs.response {obj} The response object<br>oArgs.callback {obj} The callback object<br>oArgs.tId {int} Unique transaction ID |
| responseParseEvent | oArgs.request {obj} The request object<br>oArgs.response {obj} The response object<br>oArgs.callback {obj} The callback object |

Subscribe to DataSource Custom Events on your DataSource instance: `myDS.subscribe("requestEvent", myFn);`

## Abstract Methods

| Method | Description |
|---|---|
| doBeforeCallback | This overridable abstract method gives implementers an opportunity to access the data before it has been cached or returned to the callback. Implementers should be sure to return data in a ready-to-return state to avoid errors. |
| doBeforeParseData | This overridable abstract method gives implementers an opportunity to munge the data before it is schema-parsed. Implementers should be sure to return data in a ready-to-parse state to avoid errors. |

## Dependencies

DataSource requires the YAHOO Global Object and the Event Utility. Connection Manager (for XHRDataSource), the Get Utility (for ScriptNodeDataSource), and the JSON Utility (for JSON data) are optional.

### YAHOO.util. DataSourceBase Properties:

**dataType** (int)
**liveData** (mixed)
**responseSchema** (obj)
**responseType** (int)

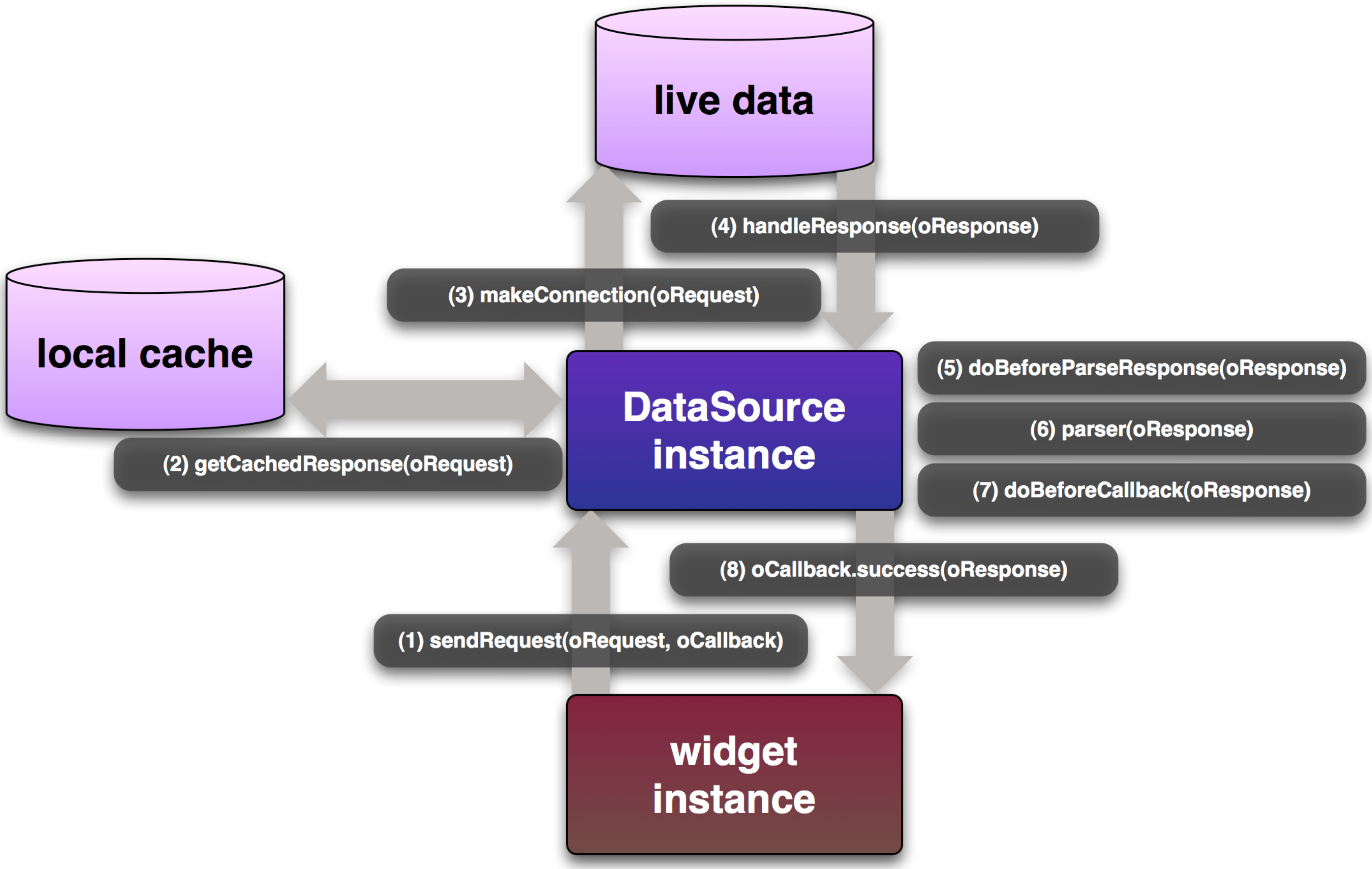
### YAHOO.util.XHRDataSource Properties:

**connMethodPost** (b)
**connMgr** (YAHOO.util.Connection)
**connTimeout** (int)
**connXhrMode** ("queueRequests" | "cancelStaleRequests" | ignoreStaleResponses" | "allowAll")

### YAHOO.util.ScriptNodeDataSource Properties:

**asyncMode** ("ignoreStaleResponses" | "allowAll")
**getUtility** (YAHOO.util.Get)
**scriptCallbackParam** (str)

**Data Flow in DataSource**


live data
(4) handleResponse(oResponse)
(3) makeConnection(oRequest)
local cache
DataSource instance
(5) doBeforeParseResponse(oResponse)
(6) parser(oResponse)
(7) doBeforeCallback(oResponse)
(2) getCachedResponse(oRequest)
(8) oCallback.success(oResponse)
(1) sendRequest(oRequest, oCallback)
widget instance

# YUI Library: DataTable

2011-3-21 **v2.9**

## Simple Use Case: YAHOO.widget.DataTable

**Markup (container can be empty or pre-populated for progressive enhancement):**

```
<div id="myContainer"><div>
```

**Script:**

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

## 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 empty 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.

## Constructor: YAHOO.widget.ScrollingDataTable

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

*Arguments:*

(1) **el:** HTML ID or HTMLElement for a DataTable container. May be empty 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, including width and height of scrollable area.

## Cell Editing

```
var myCE = new YAHOO.widget.TextboxCellEditor;
var myColumnDefs = [{key:"col1}, {key:"col2",
     editor: myCE];
...
myDT.subscribe("cellClickEvent",
     myDT.onEventShowCellEditor)
```

## Key Interesting Moments in DataTable

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

| Event | oArgs Properties |
|---|---|
| *element*ClickEvent, *element*DblclickEvent, *element*MousedownEvent, *element*MouseoutEvent, *element*MouseoverEvent | oArgs.event (HTMLEvent)<br>oArgs.target (el)<br><br>An *element* is a DOM element, such as button, cell, row, theadCell, theadLabel, etc. |
| *unit*HighlightEvent, *unit*SelectEvent, *unit*UnhighlightEvent, *unit*UnselectEvent, cellFormatEvent | oArgs.el (el)<br>oArgs.record (YAHOO.widget.Record)<br><br>When *unit* is a cell:<br>oArgs.key (string)<br><br>A *unit* is a cell, row, or column. |
| columnSortEvent | oArgs.column (YAHOO.widget.Column)<br>oArgs.dir (string) YAHOO.widget.DataTable.CLASS_ASC \|\| YAHOO.widget.DataTable.CLASS_DESC |
| editorRevertEvent, editorSaveEvent | oArgs.editor (object), oArgs.newData (object), oArgs.oldData (object) |
| initEvent, renderEvent | n/a |
| rowAddEvent | oArgs.record (YAHOO.widget.Record) |
| rowDeleteEvent | oArgs.oldData (object)<br>oArgs.recordIndex (number)<br>oArgs.trElIndex (number) |
| rowUpdateEvent | oArgs.record (YAHOO.widget.Record)<br>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 Attributes

| Option (type) | Default | Description |
|---|---|---|
| caption (s)<br>summary (s) | null<br>null | String values for caption element and summary attribute. |
| draggableColumns (b) | false | Enables Drag & Drop Column reordering. |
| initialLoad (b\|o) | true | Enables or customizes data load at instantiation. |
| initialRequest (mixed) | null | Request value to send to DataSource at instantiation for data to populate the table, if initialLoad is set to true. |
| paginator (o) | null | Instance of YAHOO.widget.Paginator. |
| renderLoopSize | 0 | Number of rows to render into the DOM each timeout loop. |
| scrollable (b)<br>width (s)/height (s) | false<br>null | Enables scrolling. |
| selectionMode (s) | "standard" | Configures row or cell selection. |
| sortedBy (o) | null | Displays sorted Column UI. |

## Abstract Methods

| Method | Description |
|---|---|
| doBeforeLoadData | Overridable method gives implementers a hook to access data before it gets added to RecordSet and rendered to the TBODY. |
| doBeforeShowCellEditor | Overridable abstract method to customize CellEditor before showing. |
| doBeforeSortColumn | Overridable method gives implementers a hook to show loading message before sorting Column. |

### YAHOO.widget.Column: Properties

**abbr**
**children**
**className**
**editor**
**formatter**
**hidden**
**key**
**label**
**maxAutoWidth**
**minWidth**
**resizeable**
**selected**
**sortable**
**sortOptions.defaultDir**
**sortOptions.sortFunction**
**width**

### YAHOO.widget.ScrollingDataTable Key Attributes

**COLOR_COLUMNFILLER**
**height**
**width**

### YAHOO.widget.BaseCellEditor Subclasses

**CheckboxCellEditor**
**DateCellEditor**
**DropdownCellEditor**
**RadioCellEditor**
**TextareaCellEditor**
**TextboxCellEditor**

**Note:** Refer to online documentation for a complete API reference.

## Dependencies

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

# YUI Library: Dialog & SimpleDialog

2011-3-21 **v2.9**

## 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

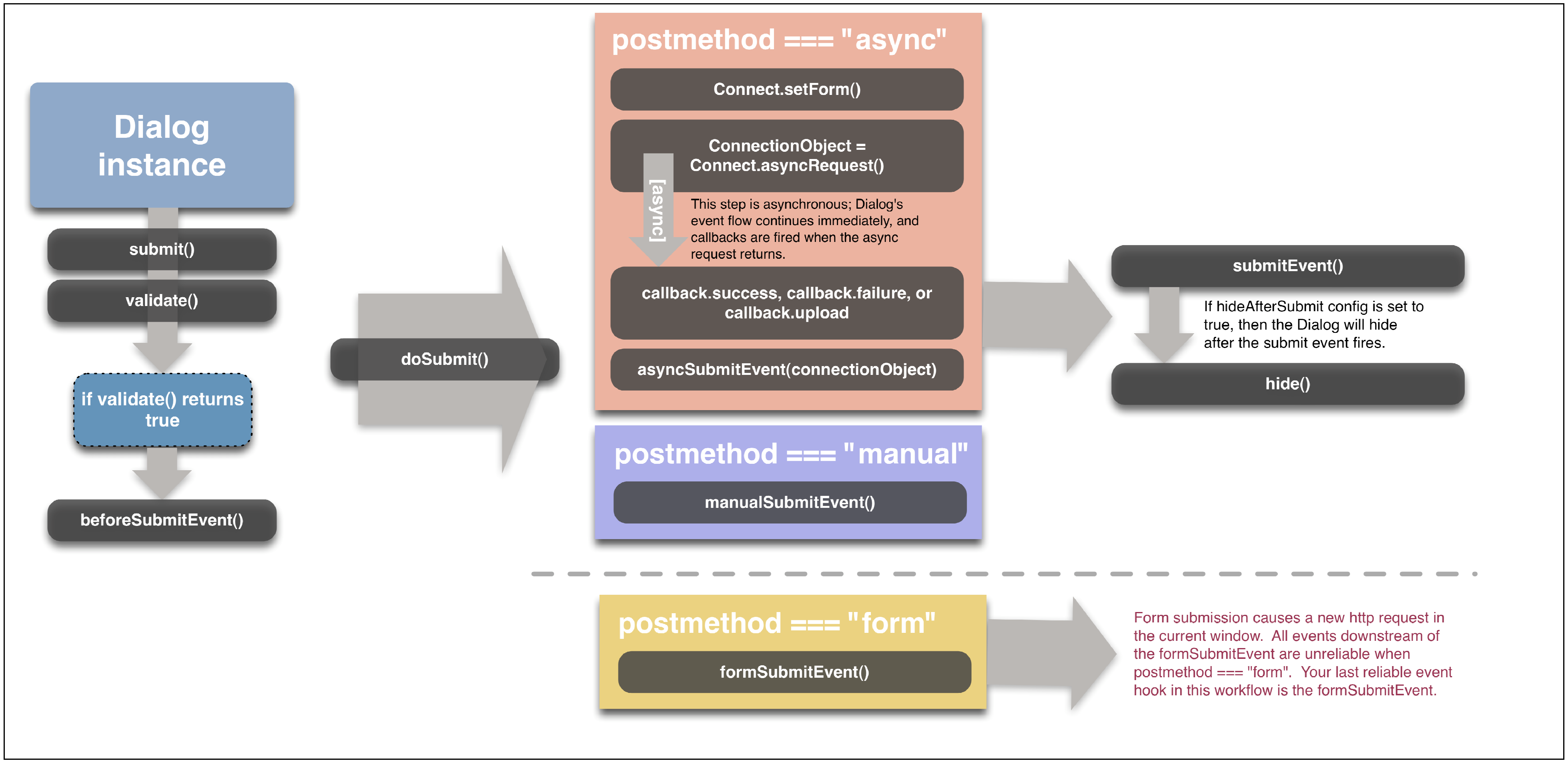
**body** (el)
**form** (el)
**callback** (o) Connection Manager callback object for async transactions.
**element** (el) containing header, body & footer
**footer** (el)
**header** (el)
**id** (s) of the element

## 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: Dialog Event Flow

**v2.9**


Dialog instance
submit()
validate()
if validate() returns true
beforeSubmitEvent()
doSubmit()
postmethod === "async"
Connect.setForm()
ConnectionObject = Connect.asyncRequest()
[async]
This step is asynchronous; Dialog's event flow continues immediately, and callbacks are fired when the async request returns.
callback.success, callback.failure, or callback.upload
asyncSubmitEvent(connectionObject)
submitEvent()
If hideAfterSubmit config is set to true, then the Dialog will hide after the submit event fires.
hide()
postmethod === "manual"
manualSubmitEvent()
postmethod === "form"
formSubmitEvent()
Form submission causes a new http request in the current window. All events downstream of the formSubmitEvent are unreliable when postmethod === "form". Your last reliable event hook in this workflow is the formSubmitEvent.

# YUI Library: Dom Collection

2011-3-21 **v2.9**

## 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 an array of HTMLElements with the given class. For **optimized performance**, include a **tag** and/or **root node**. |
| 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 an 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.Dom.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()**
**getElementsByTagName()**
**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

2011-3-21 **v2.9**

## 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)
useShim (b)
groups (ar)
id (s)
invalidHandleClasses (s[ ])
invalidHandleIds (obj)
isTarget (b)
maintainOffset (b)
padding (int[ ])
primaryButtonOnly (b)
xTicks (int[ ])
yTicks (int[ ])

**Methods:**
addInvalidHandleClass (s *cssClass*)
addInvalidHandleId (s *id*)
addInvalidHandleType (s *tagName*)
addToGroup (s *groupName*)
clearTicks()
clearConstraints()
getDragEl()
getEl()
isLocked()
lock()
removeFromGroup( o *dd*, s *group*)
removeInvalidHandleClass(s *cssClass*)
removeInvalidHandleId(s *id*)
removeInvalidHandleType (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 existing 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)
**useShim** (b)
**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

2011-3-21 **v2.9**

## 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:** 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(…)** || **on(…)**
**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 cllbck, o obj, b scp)
**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 Props & 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

2011-3-21 **v2.9**

## 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 when the requested file(s) have loaded successfully. |
| onFailure | fn | Callback method invoked when an error is detected or `abort` is called. |
| onTimeout | Fn | Callback method invoked when a resource doesn't load within the timeframe specified by the `timeout` config. |
| timeout | Int | The number of millisecond to wait for a resource to load. |
| win | obj | The window into which the loaded resource(s) will be inserted. |
| scope | obj | The execution scope in which the callbacks will run. |
| data | any | Data to pass as an argument to all callbacks. |
| 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()`. |
| insertBefore | str\|el | Element reference or id of a node that should be used as the insertion point for new nodes. This is useful for making sure CSS rules are parsed in the correct order (place your style overrides in a single style block and insertBefore this node). |
| charset | str | The charset attribute for new node(s). Default: utf-8 |

## 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

Define a callback handler and configure a transaction:

```
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.

* 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!");}});
```

# YUI Library: Browser History Manager

2011-3-21 **v2.9**

## 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.

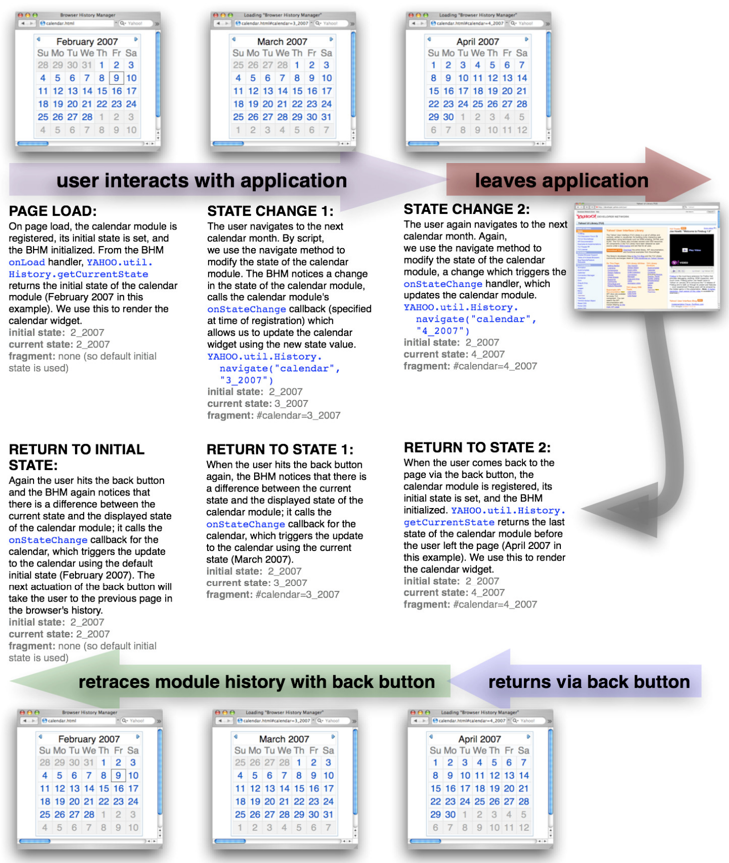
```
YAHOO.util.History.navigate(str module, str new state);
```

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



**PAGE LOAD:**
On page load, the calendar module is registered, its initial state is set, and the BHM initialized. From the BHM `onLoad` handler, `YAHOO.util.History.getCurrentState` returns the initial state of the calendar module (February 2007 in this example). We use this to render the calendar widget.
**initial state:** 2_2007
**current state:** 2_2007
**fragment:** none (so default initial state is used)

**STATE CHANGE 1:**
The user navigates to the next calendar month. By script, we use the navigate method to modify the state of the calendar module. The BHM notices a change in the state of the calendar module, calls the calendar module's `onStateChange` callback (specified at time of registration) which allows us to update the calendar widget using the new state value.
```
YAHOO.util.History.
  navigate("calendar",
  "3_2007")
```
**initial state:** 2_2007
**current state:** 3_2007
**fragment:** #calendar=3_2007

**STATE CHANGE 2:**
The user again navigates to the next calendar month. Again, we use the navigate method to modify the state of the calendar module, a change which triggers the `onStateChange` handler, which updates the calendar module.
```
YAHOO.util.History.
  navigate("calendar",
  "4_2007")
```
**initial state:** 2_2007
**current state:** 4_2007
**fragment:** #calendar=4_2007

**RETURN TO INITIAL STATE:**
Again the user hits the back button and the BHM again notices that there is a difference between the current state and the displayed state of the calendar module; it calls the `onStateChange` callback for the calendar, which triggers the update to the calendar using the default initial state (February 2007). The next actuation of the back button will take the user to the previous page in the browser's history.
**initial state:** 2_2007
**current state:** 2_2007
**fragment:** none (so default initial state is used)

**RETURN TO STATE 1:**
When the user hits the back button again, the BHM notices that there is a difference between the current state and the displayed state of the calendar module; it calls the `onStateChange` callback for the calendar, which triggers the update to the calendar using the current state (March 2007).
**initial state:** 2_2007
**current state:** 3_2007
**fragment:** #calendar=3_2007

**RETURN TO STATE 2:**
When the user comes back to the page via the back button, the calendar module is registered, its initial state is set, and the BHM initialized. `YAHOO.util.History.getCurrentState` returns the last state of the calendar module before the user left the page (April 2007 in this example). We use this to render the calendar widget.
**initial state:** 2_2007
**current state:** 4_2007
**fragment:** #calendar=4_2007

## 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: ImageCropper Control

2011-3-21 **v2.9**

## Simple Use Case: YAHOO.widget.ImageCropper

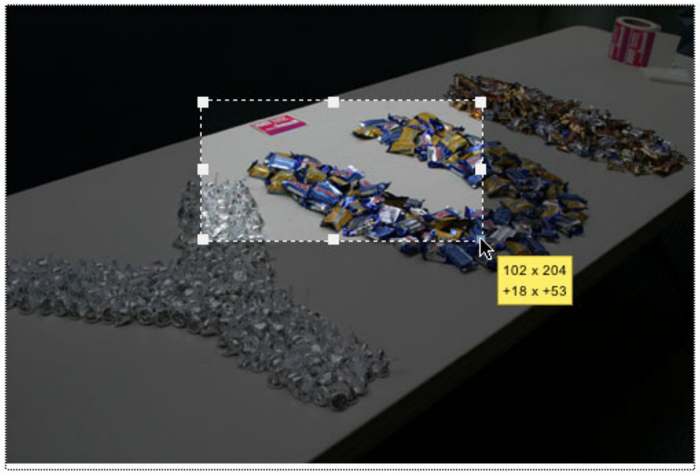
**Markup:**

```
<img src="yui.jpg" id="crop1">
```

**Script:**

```
crop = new YAHOO.widget.ImageCropper('crop1');
```

Using the above code will get you a rich control that looks something like this:



## Constructor: YAHOO.widget.ImageCropper

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

*Arguments:*

(1) **Element:** The element to make croppable.

(2) **Configuration object:** When instantiating the ImageCropper Control, 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

ImageCropper requires the Yahoo Global Object, Dom, Event, Element, DragDrop, and Resize.

## ImageCropper Events

See online docs for complete list of Resize events.

| Event | Fires... |
|---|---|
| dragEvent | Fires when the DragDrop dragEvent is fired. |
| startResizeEvent | Fires when a resize action is started. |
| resizeEvent | Fires on event element resize |
| moveEvent | Fires on every element move. Inside these methods: _handleKeyPress, _handleDragEvent, _handleResizeEvent |

All ImageCropper events are Custom Events (see Element docs); subscribe to these events using their subscribe method: `crop.on(‘resize',fnMyHandler);`.

## Key ImageCropper Configuration Options

See online docs for complete list of ImageCropper configuration options.

| Option | Default | Description |
|---|---|---|
| initHeight | 1/4 of the image height | Set the initlal height of the crop area. |
| initWidth | 1/4 of image width | Set the initla width of the crop area. |
| initialXY | [10,10] | Array of the XY position that we need to set the crop element to when we build it. |
| keyTick | 1 | The pixel tick for the arrow keys. |
| ratio | false | Constrain the resize to a ratio of the current size. |
| status | True | Show the Resize Utility status. |

Most of ImageCropper's options can be set in the constructor's second argument (eg, `{ratio: true}`) or at runtime via `set` (eg, `crop.set("ratio", true);`).

## Getting the crop area

```
//set up a new cropper
var cropImg = new YAHOO.widget.ImageCropper('img1');
//get the crop area
var cropArea = cropImg.getCropCoords();

// cropArea now looks like this:

{
    top: 70,
    left: 122,
    height: 86,
    width: 172,
    image: 'yui.jpg'
}
```

## YAHOO.widget.ImageCropper: Methods

**getActiveHandleEl()** Get the HTML reference for the currently active resize handle.

**getCropCoords()** Get the coordinates needed to crop the image

**getCropperById()** Get an ImageCropper object by the HTML id of the image associated with the ImageCropper object.

**getEl()** Get the HTML reference for the image element.

**getMaskEl()** Get the HTML reference for the mask element.

**getResizeEl()** Get the HTML reference for the resize element.

**getResizeMaskEl()** Get the HTML reference for the resizable object's mask element.

**getResizeObject()** Get the Resize Utility object.

**getWrapEl()** Get the HTML reference for the wrap element.

**destroy()** Destroys the ImageCropper object and all of its elements & listeners.

# YUI Library: ImageLoader Utility

2011-3-21 **v2.9**

## 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('someDi
   vId', 'click', 2);
myGroup.registerBgImage('imgDivId', 'http://some.image
   /url');
```

This will cause `imgDivId`'s bg image to load either when `someDivId` is clicked or two secs 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:** Max 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 <div> element with a background image):

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

PNG background images (e.g., a <div> 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. |
| addCustomTrigger | method | Adds a custom event 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

**addCustomTrigger**(obj *YAHOO.util.CustomEvent object*) adds a custom event trigger

**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*, [obj *ailProps*]) adds a png-background-type image to the group; optional properties to set AlphaImageLoader properties returns YAHOO.util.ImageLoader.imgObj

## Dependencies

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

# YUI Library: JSON Utility

2011-3-21 **v2.9**

## 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 wrap 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 reviver])`

*Arguments:*

(1) **JSON:** A string containing JSON-formatted data that you wish to validate and parse.
(2) **reviver**: A function that will be executed on each member of the JSON object; 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 | fn replacer[, n depth]])`

*Arguments:*

(1) **object:** The JavaScript object you want to stringify.
(2) **whitelist:** An optional array of acceptable keys to include.
(2*) **replacer:** A function to offer a replacement value for serializing. Executed on each member of the input object.
(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 JSON string representing the object.

## 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 *reviver* argument

You can filter out and/or reformat specific pieces of data while applying the `parse` method by passing in a second (optional) argument, `reviver`. Reviver 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.price)
```

## YAHOO.lang.JSON Methods

**parse**(str *JSON*[,fn *reviver*]) see usage at left
**stringify**(obj *object*[, arr *whitelist* | fn *replacer*[, n *depth*]]) see usage at left
**isValid**(str *JSON*) see online docs

## Dependencies

The JSON Utility requires only the YAHOO Global Object.

# YUI Library: Layout Manager

2011-3-21 **v2.9**

## Simple Use Case: Creating a Layout

Layouts can be created targeting the full browser viewport:

```
var layoutFull = new YAHOO.widget.Layout({/* Config
   here */});
layoutFull.render();
```

Layouts can also target a specific page element:

```
//Element Based Layout
var layoutEl = new YAHOO.widget.Layout('demo', { /*
   Config here */});
layoutEl.render();
```

Layouts consist of up to five Layout Units (top, right, bottom, left and center; center is required, fluid, and cannot be resized).

```
var layoutFull = new YAHOO.widget.Layout({
   units: [{position: 'top'}, {position: 'center'}]
});
layoutFull.render();
```

See Layout Units section for more on configuring a Layout Unit.

## Constructor: YAHOO.widget.Layout

```
YAHOO.widget.Layout([str | obj container,] obj
   configuration)
```

*Arguments:*

(1) **Container (optional):** A reference to a DOM element (by ID or direct reference) that will contain the Layout; if this argument is omitted, the Layout will take up the full browser viewport.

(2) **Configuration:** An optional object containing your desired configuration options, including information about your Layout Units. See Layout Units and Configuration Options sections for details.

## Layout Units: Key Configuration Attributes

| | | | |
|---|---|---|---|
| animate | Use animation on expand/ collapse? | resize | Is this unit resizeable? |
| collapse | Adds collapse icon | scroll | Is units body content scrollable? |
| duration | Duration in ms of animation transition | width | Width of unit in px |
| easing | Animation easing effect to use (see anim docs) | header/ body/ footer | Contents of the header, body and footer sections of the unit |
| gutter | Gutter surrounding unit (in px; supports “t r b l” or “tb rl” css-style syntax) | | |
| height | Height of this unit in px | | |
| maxHeight/Width, minHeight/Width | Max/min dimensions of unit in px. | | |
| position | Position of this unit in the Layout (top, right, bottom, left or center) | | |

Layout Units can be instantiated or created and configured as part of the Layout constructor:

```
var layoutFull = new
    YAHOO.widget.Layout({
     units: [
     {position: 'center',
      gutter: "5 7 0 5”,
      scroll: true,
      minHeight: 225
    }]
});
layoutFull.render();
```

## Layout Configuration Options

| Field | Type | Description |
|---|---|---|
| height | integer | Height of the Layout in pixels. |
| minHeight | integer | Minimum height of the Layout in pixels. |
| minWidth | integer | Minimum width of the Layout in pixels. |
| parent | Layout object | If this Layout is a child of another Layout, sets the relationship and binds the Layouts’ resize evts together. |
| width | integer | Width of the Layout in pixels. |

Configuration options should be set in the second argument of the constructor:

```
var pv = new YAHOO.widget.Layout(“myEl”, {height: 400});.
```

## Key Interesting Moments in Layout

| Event | Description/Fields: |
|---|---|
| render | Event fires when the rendering of the Layout is complete. |
| beforeResize | Fires at the beginning of the resize process; return false to prevent resize. |
| resize | Fires after the resize process completes. |

**Subscribe:** `layout.on(“render”, function(o){});.`

## Key Interesting Moments in LayoutUnit

| Event | Description/Fields: |
|---|---|
| close | Fires when the unit is closed. |
| collapse | Fires when the unit is collapsed. |
| contentChange | Fires when header/body/footer content is changed via API. |
| expand | Fires when the unit is expanded. |
| beforeResize | Fires at the beginning of the resize process; return false to prevent resize. |
| resize | Fires after the resize process completes. |

**Subscribe:** `layoutUnit.on(“close”, function(o){});.`

## Solutions: Embedding a Layout inside another Layout

```
var layout = new YAHOO.widget.Layout({
  units: [
   { position: 'top', height: 300, body: 'Top #1'},
   { position: 'center', body: '' } //empty body for next layout
  ]
});

layout.on('render', function() {
   var c = layout.getUnitByPosition('center');
    //Apply the new layout to the body element of the first layout
   var layout2 = new YAHOO.widget.Layout(c.body, {
     parent: layout,
     units: [
       { position: 'left', width: 200, body: 'Left #2'},
       { position: 'center', body: 'Center #2' }
     ]
    });
    layout2.render();
});
layout.render();
```

## YAHOO.widget.Layout Methods

**addUnit**(o *cfg*) there must not be a LayoutUnit at the new unit’s position
**getLayoutById**(s *id*) static method, returns Layout whose parent is element *id*
**getSizes**() returns object containing sizes of all child Layout Units
**getUnitById**(s *id*) returns LayoutUnit whose parent is element *id*
**getUnitByPosition**(s *pos*) returns Layout Unit at specified position
**removeUnit**(o *unit*) removes the unit; Layout resizes automatically

## YAHOO.widget.LayoutUnit Properties

**body, header,** *and* **footer** HTML elements for specified sections

## YAHOO.widget.LayoutUnit Methods

**close**() collapses and removes the unit
**collapse**() collapses the unit, if not already collapsed
**destroy**() removes the unit and cleans up references and listeners
**expand**() expands the unit, if not already expanded
**getLayoutUnitById**(s *id*) static method returns the unit that is associate with a given HTML id
**getUnitByPosition**(s *pos*) returns Layout Unit at specified position
**getSizes**() returns object containing size information for this unit

## Dependencies

Layout Manager/Unit requires: Yahoo, Dom, Event and Element. Animation, DragDrop, Resize and Selector are optional.

# YUI Library: Logger

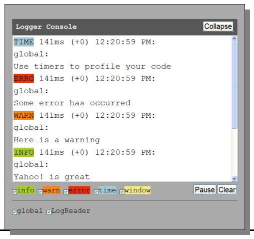
2011-3-21 **v2.9**

## 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.



```
Logger Console
TIME 141ms (+0) 12:20:59 PM:
global:
Use timers to profile your code
ERRO 141ms (+0) 12:20:59 PM:
global:
Some error has occurred
WARN 141ms (+0) 12:20:59 PM:
global:
Here is a warning
INFO 141ms (+0) 12:20:59 PM:
global:
Yahoo! is great
```

## 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

2011-3-21 **v2.9**

## 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

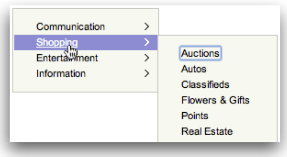
```
YAHOO.widget.Menu(str elId[, obj config]);
```

*Arguments:*

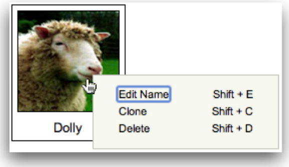
(1) **Element ID:** HTML ID of the element being used to create the Menu. If this element doesn't exist, it will be created and appended to the document body.

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

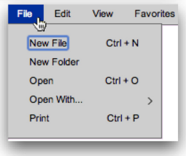
## Three Types of Menus



**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.

| | |
|---|---|
| itemAdded | show |
| itemRemoved | hide |
| render | beforeShow |
| beforeRender | beforeHide |

All Menu events are YUI Custom Events (see Event Utility docs); subscribe to these events using their subscribe method: `oMenu.subscribe("show", 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. |
| shadow | b | Renders a Menu with a shadow. |
| keepopen | b | Keeps the Menu open when clicked. |

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

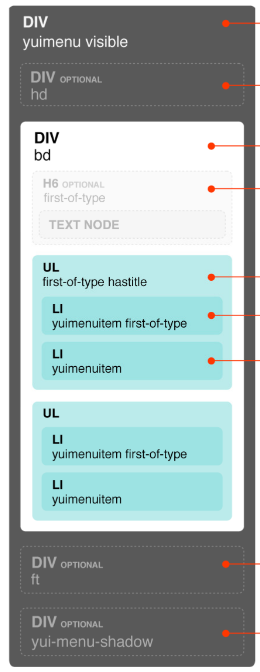
Menu requires the Container Core package, the YAHOO Object, Event, and Dom.

# YUI Library: Menu HTML & CSS Reference Guide

2011-3-21

v2.9

## CLASSIC MENU: `YAHOO.widget.Menu`



**<DIV class="yuimenu">**
Container for a `YAHOO.widget.Menu` instance
**Other Optional CSS Classes:** `visible` class is applied when a `Menu` instance is visible

**<DIV class="hd">**
OPTIONAL: Used for the top scroll control
**Other Optional CSS Classes:** `topscrollbar` | `topscrollbar_disabled`

**<DIV class="bd">**
Used to hold a list of menu options
**Other Optional CSS Classes:** `yui-menu-body-scrolled`

**<H6>**
OPTIONAL: Text node of **H6** used to title each **UL** node.
**CSS Class Usage:**
`first-of-type` class is applied to the first **H6** node that is a child of the node `<DIV class="bd">`

**<UL>**
List of menu options
**CSS Class Usage:**
`hastitle` class is only applied when this **UL** node's preceding node is an **H6** node
`first-of-type` class is applied to the first **UL** node that is a child of the node `< DIV class="bd">`

Repeating Unit
This **H6** + **UL** node structure can be repeated any number of times to group items in a `Menu` instance

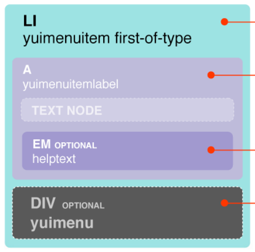
**<LI class="yuimenuitem">**
Defines a menu option.
**CSS Class Usage:**
`first-of-type` class is applied to the first **LI** node of each **<UL>**

Repeating Unit
This **LI** node structure can be repeated any number of times to specify a set of options

**<DIV class="ft">**
OPTIONAL: Used for the bottom scroll control
**Other Optional CSS Classes:** `bottomscrollbar` | `bottomscrollbar_disabled`

**<DIV class="yui-menu-shadow">**
OPTIONAL: Used to display a shadow under the menu
**Other Optional CSS Classes:** `yui-menu-shadow-visible` class is applied when a `Menu` instance is visible

## CLASSIC MENU - MENU ITEM: `YAHOO.widget.MenuItem`



**<LI class="yuimenuitem">**
Container for a `YAHOO.widget.MenuItem` instance
**Other Optional CSS Classes:**
`first-of-type` class is applied to the first **LI** node of each <**UL**>

| | |
|---|---|
| `.yuimenuitem-hassubmenu` | `.yuimenuitem-checked` |
| `.yuimenuitem-selected` | `.yuimenuitem-disabled` |
| `.yuimenuitem-checked-selected` | `.yuimenuitem-checked-disabled` |
| `.yuimenuitem-hassubmenu-selected` | `.yuimenuitem-hassubmenu-disabled` |

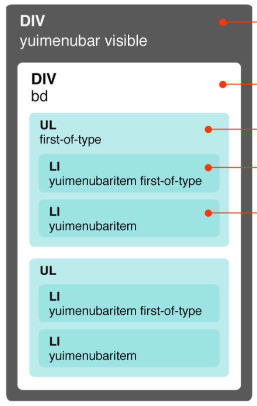
**<A class="yuimenuitemlabel">**
Link for each menu item
**Other Optional CSS Classes:**

| | |
|---|---|
| `.yuimenuitemlabel-hassubmenu` | `.yuimenuitemlabel-checked` |
| `.yuimenuitemlabel-selected` | `.yuimenuitemlabel-disabled` |
| `.yuimenuitemlabel-checked-selected` | `.yuimenuitemlabel-checked-disabled` |
| `.yuimenuitemlabel-hassubmenu-selected` | `.yuimenuitemlabel-hassubmenu-disabled` |

**<EM class="helptext">**
OPTIONAL: Exists if the `MenuItem` has help text

**<DIV class="yuimenu">**
OPTIONAL: Exists if the `MenuItem` has a submenu

## MENU BAR: `YAHOO.widget.MenuBar`



**<DIV class="yuimenubar">**
Container for a `YAHOO.widget.MenuBar` instance
**Other Optional CSS Classes:**
`visible` class is applied when a `MenuBar` instance is visible

**<DIV class="bd">**
Used to hold a list of menu options

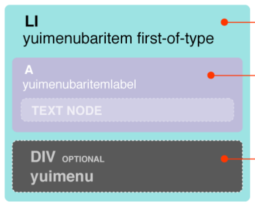
**<UL>**
List of menu options
**CSS Class Usage:**
`first-of-type` class is applied to the first **UL** node that is a child of the node `<DIV class="bd">`

Repeating Unit
This **UL** node can be repeated any number of times to group items in a `MenuBar` instance

**<LI class="yuimenuitem">**
Defines a menu option.
**CSS Class Usage:**
`first-of-type` class is applied to the first **LI** node of each **<UL>**

Repeating Unit
This **LI** node structure can be repeated any number of times to specify a set of options

## MENU BAR - MENU ITEM: `YAHOO.widget.MenuBarItem`



**<LI class="yuimenubaritem">**
Container for a `YAHOO.widget.MenuBarItem` instance
**Other Optional CSS Classes:**
`first-of-type` class is applied to the first **LI** node of each <**UL**>

| | |
|---|---|
| `.yuimenubaritem-hassubmenu` | `.yuimenubaritem-selected` |
| `.yuimenubaritem-disabled` | `.yuimenubaritem-hassubmenu-selected` |
| `.yuimenubaritem-hassubmenu-disabled` | |

**<A class="yuimenubaritemlabel">**
Link for each menu item
**Other Optional CSS Classes:**

| | |
|---|---|
| `.yuimenubaritemlabel-hassubmenu` | `.yuimenubaritemlabel-selected` |
| `.yuimenubaritemlabel-disabled` | `.yuimenubaritemlabel-hassubmenu-selected` |
| `.yuimenubaritemlabel-hassubmenu-disabled` | |

**<DIV class="yuimenu">**
OPTIONAL: Exists if the `MenuBarItem` has a submenu

# YUI Library: Panel 2011-3-21 v2.9

## 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 | Enforce Panel modality with a modal mask |
| visible (b) | true | Sets the "display" style property to "block" (true) or "none" (false). |
| x, y, and xy (int, int, arr) | null | These properties can be used to set the Panel's "top" and/or "left" styles. |
| context (arr) | 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}` |
| autofillheight (s) | "body" | The element which will fill out the container height when the "height" config property is set: "body", "footer", "header" or null |

## 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**()
**focusFirst**()
**focusLast()**

## Dependencies

Panel requires the full Container package, the YAHOO object, Event, and Dom. Animation, and Drag and Drop are optional. **Note:** By default, Panels enable Drag and Drop if included on the page

# YUI Library: Profiler

2011-3-21 **v2.9**

## 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.

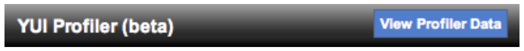
# YUI Library: ProfilerViewer Control

2011-3-21 **v2.9**

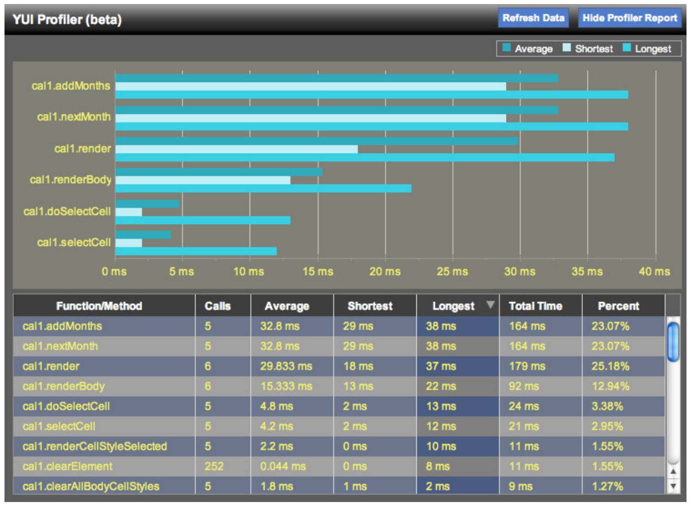
## Simple Use Case: Profiling an Object

```
//assuming you have an object with functions called
   "myObject":
YAHOO.tool.Profiler.registerObject("myObject",
   myObject); //see Profiler docs for more on how
              //to set up your code profiles.
var pv = new YAHOO.widget.ProfilerViewer("myEl");
```

This code tells the Profiler to profile `myObject` and tells ProfilerViewer to create a viewer display in the DOM element whose id is `myEl`. ProfilerViewer will show all profiled functions; use the `filter` attribute to limit a viewer instance to a subset of profiled functions. Upon instantiation, the ProfilerViewer launcher is rendered:



Once the View Profiler Data button is clicked, ProfilerViewer will load the DataTable and Charts controls and display the viewing console. A ProfilerViewer console with default options will look like this:



| Function/Method | Calls | Average | Shortest | Longest | Total Time | Percent |
|---|---|---|---|---|---|---|
| cal1.addMonths | 5 | 32.8 ms | 29 ms | 38 ms | 164 ms | 23.07% |
| cal1.nextMonth | 5 | 32.8 ms | 29 ms | 38 ms | 164 ms | 23.07% |
| cal1.render | 6 | 29.833 ms | 18 ms | 37 ms | 179 ms | 25.18% |
| cal1.renderBody | 6 | 15.333 ms | 13 ms | 22 ms | 92 ms | 12.94% |
| cal1.doSelectCell | 5 | 4.8 ms | 2 ms | 13 ms | 24 ms | 3.38% |
| cal1.selectCell | 5 | 4.2 ms | 2 ms | 12 ms | 21 ms | 2.95% |
| cal1.renderCellStyleSelected | 5 | 2.2 ms | 0 ms | 10 ms | 11 ms | 1.55% |
| cal1.clearElement | 252 | 0.044 ms | 0 ms | 8 ms | 11 ms | 1.55% |
| cal1.clearAllBodyCellStyles | 5 | 1.8 ms | 1 ms | 2 ms | 9 ms | 1.27% |

## Constructor: YAHOO.widget.ProfilerViewer

```
YAHOO.widget.ProfilerViewer([str | obj container, obj
   configuration])
```

*Arguments:*

(1) **Container:** (Optional) A reference to a DOM element (by ID or direct reference) that will contain the ProfilerViewer display. For best results, us an element with at least 750px of viewable width. If no element is passed, a new element will be created as the first child of the <body> element.

(2) **Configuration:** (Optional) An optional object containing your desired configuration options. See Configuration Options section for details.

## Key ProfilerViewer Configuration Options

| Field | Type | Description |
|---|---|---|
| base | string | Path to your YUI base directory, to be used by YUI Loader in pulling in dependencies on-demand. Default: YUI files will be served from yui.yahooapis.com. |
| filter | function | The filter used by ProfilerViewer in determining which profilerd functions to show in the display. See Solutions below for more. |
| maxChart Functions | string | The maximum number of functions to profile in the Chart display. Default: *6.* |
| showChart | boolean | Determines whether or not the Chart Control should be used to visualize profiling data. Default: *true.* |
| sortedBy | object | `{key: string, dir: string}` The default sort column and direction for data in the DataTable. Valid **keys** are: *fn, calls, avg, min, max, total, pct.* Valid **dir** values are: *YAHOO.widget.DataTable.CLASS_ASC* and *YAHOO.widget.DataTable.CLASS_DESC.* |
| swfUrl | string | Relative path or url to the YUI Charts Control .swf file. Defaults to current version hosted on yui.yahooapis.com. |
| tableHeight | string | Height of the DataTable portion of the console. Default: *"15em".* |
| visible | boolean | If *true*, the ProfilerViewer Console will render immediately upon instantiation. Otherwise, just the launcher will render initially. Default: *false.* |

Configuration options should be set in the second argument of the constructor:
`var pv = new YAHOO.widget.ProfilerViewer("myEl", {visible:true});`.

## Key Interesting Moments in ProfilerViewer

| Event | Description/Fields: |
|---|---|
| renderEvent | Event fires when the viewer canvas first renders. No arguments passed. |
| refreshDataEvent | Event fires when a data refresh is requested through the UI or programmatically. |
| sortedByChange | Event fires when the DataTable is resorted. Argument: {newValue: *new value*, oldValue: *old value*}. |
| visibleChange | Event fires when the viewer console is shown/hidden. Argument: {newValue: *new value*, oldValue: *old value*} |

**Subscribe:** `pv.subscribe("visibleChange", function(o){});`.

## Solutions:

Configure ProfilerViewer to ***not* use the Charts Control**:

```
var pv = new YAHOO.widget.ProfilerViewer("myEl",
      showChart: false);
```

**Use a filter function** to only display profiling data for functions that have been called at least once:

```
var pv = new YAHOO.widget.ProfilerViewer("myEl",
      filter: function(o) {return o.calls > 0;}
   );
```

## YAHOO.widget. ProfilerViewer Properties

**STRINGS** static member containing strings used in the UI; see API docs for the format of this object and examples for how to use this object to internationalize your ProfilerViewer console.

## YAHOO.widget. ProfilerViewer Methods

**getBodyEl**() returns the element containing the console body (chart and table)

**getChart**() returns the Chart Control instance (if present)

**getChartEl**() returns the element containing the Chart Control and legend

**getDataTable**() returns the DataTable Control instance

**getHeadEl**() returns the element containing console header and buttons

**getTableEl**() returns the element containing DataTable's DOM structure.

**refreshData**() programmatically refreshes the data in the console

## Dependencies

The ProfilerViewer Control requires the YUI Loader (which includes the Yahoo Global Object and the Get Utiltiy), Dom Collection, Event Utility, Profiler, and Element Utility. (The DataTable Control and Charts Control are brought in dynamically by YUI Loader when the ProfilerViewer UI is first made visible.)

# YUI Library: ProgressBar

2011-3-21 **v2.9**

## Simple Use Case

```
var pb = new
   YAHOO.widget.ProgressBar().render("pbDiv1");
```

Creates a simple ProgressBar in the container with an ID of "pbDiv1" with all default values.

## Constructor: YAHOO.widget.ProgressBar

```
YAHOO.widget.ProgressBar( {attName:attValue, ...});
```

*Argument is an optional object literal of attribute name:value pairs. Constructor inherits from YAHOO.util.Element. Element's attributes and methods are also available.*

## CSS Class Names

| Selector | Attribute | Description |
|---|---|---|
| .yui-pb | width<br>height | Overall size of the ProgressBar. It can also be set via the *width* and *height* configuration settings. |
| .yui-pb | background-image<br>background-color | Background to be used on the area that the bar is not covering. |
| .yui-pb | border | Border around the component. |
| .yui-pb-bar | background-image<br>background-color | Image or color to use for the bar itself. |
| .yui-pb-bar | margin | Offset from the edge of the ProgressBar to where the transparency of the mask (if applicable) starts. |
| .yui-pb .yui-pb-anim | background-image<br>background-color | Image or color to use for the bar while it is moving. |
| .yui-pb-mask div | background-image | Mask with transparencies that allow the bar to show through. |
| .yui-pb-caption | font and others | Not used by the ProgressBar. Defined in the "Sam" skin to style the display of the bar's value. |
| .yui-pb-range | font and others | Not used by the ProgressBar. Defined in the "Sam" skin to style the display of the *minValue* and *maxValue* values. |

## Interesting Moments in ProgressBar

| Event | Fires... | Arguments |
|---|---|---|
| start | ... once when the bar is about to move. | Returns the value represented by the bar at the instant event is fired. |
| progress | ... at least once while the bar is moving. | |
| end | ... when the bar has reached the value set. | |

ProgressBar events are Custom Events; subscribe to them by name using the following syntax: `pb.subscribe("start", fn);`.

## Solution: Cancel Default "Sam" Skin

```
.yui-skin-sam .yui-pb  {
   background-color:transparent;
   background-image:none;
   border:none;
}

.yui-skin-sam .yui-pb-bar {
   background-color:transparent;
   background-image:none;
}

.yui-skin-sam .yui-pb-mask {
   border:none;
   margin:0;
}
```

## Solution: Vertical Thermometer

Draw a vertical ProgressBar representing a thermometer with a range between 32 and 212 degrees Ferenheit. Width and height values are set in pixels. The bar will be red over a silver background and with a thin black border using custom style values.

```
var pb = new
   YAHOO.widget.ProgressBar({width:10,height:100,
direction:"btt",minValue:32,maxValue:212,value:70,
anim:true,ariaTextTemplate:"{value} degrees
   Fahrenheit"});

pb.render("thermometerDiv");

var anim = pb.get("anim");
anim.duration = 2; // seconds
anim.method = YAHOO.util.Easing.easeBoth;

YAHOO.util.Dom.setStyle(pb.get("barEl"),"backgroundColor"
   , "red");
pb.setStyle("backgroundColor","silver");
pb.setStyle("border","thin solid black");
```

## YAHOO.widget.ProgressBar : Attributes

**value** (n)
**minValue** (n)
**maxValue** (n)
**width** (n|str)
**height** (n|str)
**direction** ("ltr"|"rtl"|"ttb"|"btt")
**anim** (b) Getter returns either an instance of YAHOO.util.Anim or null.
**ariaTextTemplate** (str)
**element** (el) Read-only.
**maskEl** (el) Read-only.
**barEl** (el) Read-only.

## YAHOO.widget.ProgressBar: Methods

**render(container *el*, before *el*)**
**redraw**()
**destroy**()

## YAHOO.widget.ProgressBar: Dependencies

**ProgressBar requires the common YUI Core components (Yahoo, Dom, and Event) and Element. Animation is optional.**

# YUI Library: Resize Utility

2011-3-21 **v2.9**

## Simple Use Case: YAHOO.util.Resize

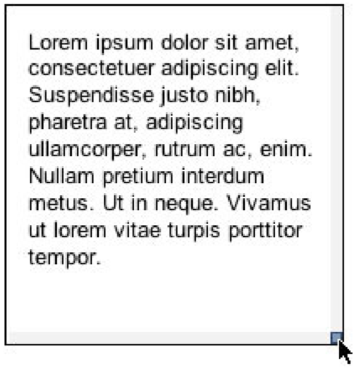
**Markup:**

```
<div id="resizeMe"><p>Lorem ipsum</p></div>
```

**Script:**

```
var resize = new YAHOO.util.Resize('resizeMe');
```

Creates a Resize instance with default configurations.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Suspendisse justo nibh, pharetra at, adipiscing ullamcorper, rutrum ac, enim. Nullam pretium interdum metus. Ut in neque. Vivamus ut lorem vitae turpis porttitor tempor.

## Constructor: YAHOO.util.Resize

```
YAHOO.util.Resize(str | el ref container[, obj
   config])
```

*Arguments:*

(1) **Element:** The element to make resizable.

(2) **Configuration object:** When instantiating Resize, 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.

## Solutions: Customizing the Proxy Element

The following code demonstrates customization of the proxy element.

```
//instantiate Resize:
var myResize = new YAHOO.util.Resize('resizeMe', {
  proxy: true
});

//customize proxy during resize via innerHTML:
myResize.on('startResize', function() {
  myResize.getProxyEl().innerHTML = 'I am the proxy';
  YAHOO.util.Dom.setStyle(myResize.getProxyEl(),
'opacity', ',5');
});
```

## Resize Events

| Event | Fires... |
|---|---|
| dragEvent | Fires when the DragDrop dragEvent is fired for the config option draggable. |
| startResize | Fires when a resize action is started. |
| beforeResize | Fires before every element resize, after the size calculations have been done. Returning false will cancel the resize. |
| resize | Fires on event element resize (only fires once when used with proxy config setting) |
| proxyResize | Fires on every element resize (only fires when used with proxy config setting). |

All Resize events are Custom Events (see Element docs); subscribe to these events using their subscribe method: `resize.on(‘resize,fnMyHandler);`.

## Key Resize Configuration Options

| Option | Default | Description |
|---|---|---|
| proxy | false | Resize a proxy element instaed of the actual element . |
| animate | false | Indicates whether or not the resize should animate sizes (only works with proxy). |
| status | false | Should we show the status tooltip. |
| handles | [‘r’, ‘br’, ‘b’] | The handles to use (any combination of): 't', 'b', 'r', 'l', 'bl', 'br', 'tl', 'tr'. Can use a shortcut of All. Note: 8 way resizing should be done on an element that is absolutely positioned. |
| ratio | false | Constrain the resize to a ratio of the current size. |
| draggable | false | A convenience method to make the element draggable. |

Most of Resize options can be set in the constructor's second argument (eg, `{animate: true}`) or at runtime via `set` (eg, `resize.set("animate", true);`).

## Resize Handles

The Resize Utility supports the following handle positions: Top, Bottom, Left, Right, Top Right, Top Left, Bottom Right, Bottom Left. The default handle positions are: Right, Bottom, and Bottom Right.

The default look of the handles is to take up all available space around the element to be resized. There are a few configuration options built in that will alter this look:

- **hiddenHandles** - Handles are always transparent, the user gets feedback from the cursor change.
- **hover** - Handles are hidden by default until the user hovers over them, then they appear.
- **knobHandles** - Used for the classic 8-way resize.

Take a look at the Resize Utility's examples for demos of all of these options.

**Note**: To get the best effect out of using all 8 resize handles, it is recommended that the element be absolutely positioned (and if possible be a direct child of the body).

## YAHOO.util.Resize: Methods

**getActiveHandleEl()** Get the HTML reference for the currently active resize handle.

**getProxyEl()** Get the HTML reference for the proxy, returns null if no proxy.

**getResizeById()** Get a resize object by the HTML id of the element associated with the Resize object.

**getStatusEl()** Get the HTML reference for the status element.

**getWrapEl()** Get the HTML reference for the wrap element, returns the current element if not wrapped.

**isActive()** Returns true or false if a resize operation is currently active on the element.

**reset()** Resets the element to its start state.

**resize()** Resizes the element, wrapper, or proxy based on the data from the handlers.

**destroy()** Destroys the resize object and all of its elements & listeners.

## Dependencies

The Resize Utility requires Yahoo, Dom, Element, Event, Drag and Drop and Animation (optional).

# YUI Library: Rich Text Editor

2011-3-21 **v2.9**

## Simple Use Case: YAHOO.widget.Editor

## Interesting Moments in Rich Text Editor & Toolbar

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

## YAHOO.widget.Editor: Methods

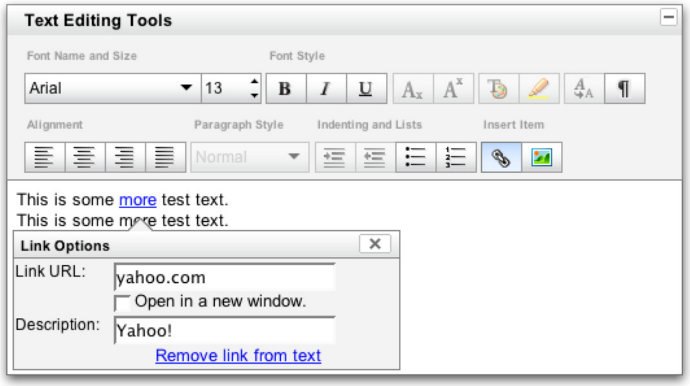
**Markup:**

```
<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.



This is some more test text.
This is some more test text.

Link URL: yahoo.com
Description: Yahoo!

## Constructor: YAHOO.widget.Editor

`YAHOO.widget.Editor(str|el` *`container`*`[, obj` *`cfg`*`])`

*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.

| 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.

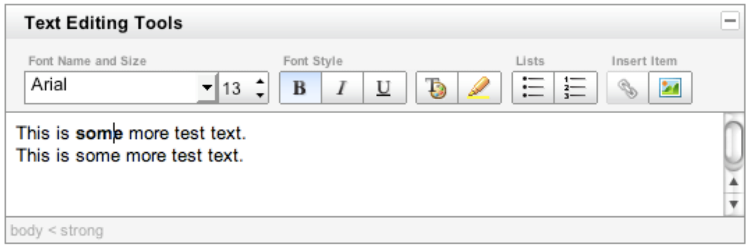
| Option (type) | Default | Description |
|---|---|---|
| height, width | best guessed size of textarea | The height/width of the editor iframe container not including the toolbar. |
| animate | false | Indicates whether or not the editor should animate movements. |
| disabled | false | Toggle for the 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.



This is some more test text.
This is some more test text.

**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

2011-3-21 **v2.9**

## 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.query
    ("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.query(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 *an + b* siblings after it. |
| :nth-of-type(*an+b*) | An element that has *an + b* siblings before it that share the same element name. |
| :nth-last-of-type(*an+b*) | An element that has *an + 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

2011-3-21 **v2.9**

## 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:

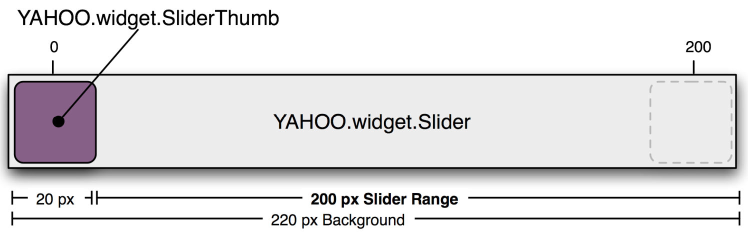
```
mySlider.subscribe("slideEnd", function() {
  alert(this.getValue()); //alerts offset from start
});
```

## 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 |

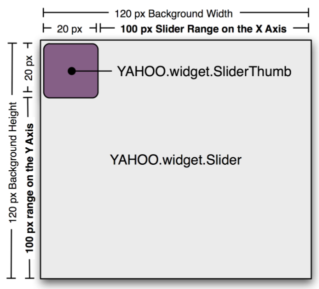
Slider events are Custom Events; subscribe to them by name using the following syntax: `mySlider.subscribe("change", fn);`.

## Slider Design Considerations



A Slider is an implementation of a "finite range control." The *range* defined by the Slider is incremented in pixels. **The max 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**()

### CSS Notes:

- Slider background should be `position:relative;`
- Slider thumb should be `position:absolute;`
- Slider thumb image should **not** be a background image
- Alternately use Slider's skin CSS file for default appearance. (see online docs)

# YUI Library: Slider with Dual Thumbs

2011-3-21 **v2.9**

## Simple Use Case

**Markup:**

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

**Script:**

```
var slider =
   YAHOO.widget.Slider.getHorizDualSlider(
          "sliderbg","minthumb","maxthumb", 200);
```

Creates two thumbs (minthumb and maxthumb) that can move within a horizontal 200 pixel range on a slide background (slidebg).

## Constructor: YAHOO.widget.DualSlider

```
YAHOO.widget.Slider.getHorizDualSlider(str bgid,
     str minthumbid, str maxthumbid, int range[, int
   tick[,array initVals]]]);
```

*Arguments for Horizontal and Vertical DualSliders:*

(1) **Background element ID:** HTML ID for the slider's background.
(2) **Min Thumb element ID:** HTML ID for the thumb element representing the lower value.
(3) **Max Thumb element ID:** HTML ID for the thumb element representing the upper value.
(4) **Range:** The maximum pixel offset for the Max Thumb.
(5) **Tick interval:** Number of pixels between each tick mark.
(7) **Initial Values:** Array containing the desired Min Thumb and Max Thumb pixel offsets to assign during instantiation.

## Solutions

**Create a vertical DualSlider** with a 300 pixel range, ticks at 10 px intervals, and initial values of 160 and 220:

```
var slider =
   YAHOO.widget.Slider.getVertDualSlider("sliderbg"
   , "minthumb", "maxthumb", 300, 10, [160,220] );
```

Assuming an instance of a DualSlider in variable `mySlider`, **write a handler** for its `change` event:

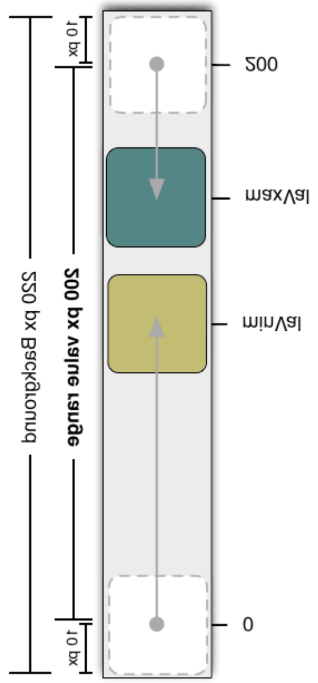
```
mySlider.subscribe("change", function() {
  alert("MIN: "+this.minVal+" MAX: "+this.maxVal);
});
```

## Interesting Moments in DualSlider see docs for full list

| Event | Fires... | Arguments |
|---|---|---|
| slideStart | ...at the **beginning** of a user-initiated change in either thumb position. | Slider instance<br>Slider instance housing the active thumb |
| slideEnd | ... at the **end** of a user-initiated change in either thumb position. | Slider instance<br>Slider instance housing the active thumb |
| change | ...each time either of the thumbs' positions change during a user-initiated move. | DualSlider instance |

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

## DualSlider Design Considerations



DualSlider is an implementation of a "finite range control." The *range* defined by the DualSlider thumbs is expressed in pixels.

**The maximum range of a slider is the pixel-width of the DualSlider's background minus half the width of the Min Thumb minus half the width of the Max Thumb.**

Values for each thumb are calculated according to their center point. To quantify the space between the thumbs, use this formula:

innerDiff = maxVal - minVal - (maxThumbWidth/2) - (minThumbWidth/2)

## YAHOO.widget.Slider: Factory Methods

**getHorizDualSlider**()
**getVertDualSlider**()

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

## YAHOO.widget.DualSlider: Properties

**minVal** (n) *(read only)*
**maxVal** (n) *(read only)*
**isHoriz** (b) *(read only)*
**minSlider** (Slider) *(read only)*
**maxSlider** (Slider) *(read only)*

**minRange** (n) **(read / write)**
minimum number of pixels the inner edges of the thumbs can be apart

## YAHOO.widget.DualSlider: Methods

**setMinValue**(int)
**setMaxValue**(int)
**setValues**(int *newMinOffset*, int *newMaxOffset*, b *skipAnimation*)

## YAHOO.widget.DualSlider: Dependencies

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

## CSS Notes:

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

# YUI Library: SWFStore

2011-3-21 **v2.9**

## Instantiating the SWFStore

```
<div id="SWFStoreDiv" style="width:0px;height:0px;">
</div>

<script>
var swfstore = new YAHOO.util.SWFStore("SWFStoreDiv");
</script>
```

Instantiates a new SWFStore object, `swfstore`, which is bound to a div whose id attribute is `'SWFStoreDiv'`. This will create an invisible component that won't take up any space. To make the component large enough to display user settings, simply set the size of the div to `height:215; width:138;`

## Constructor

```
YAHOO.util.SWFStore(str element, bool shareData,
   bool useCompression );
```

*Arguments:*

(1) **element:** HTML ID for the SWFStore container. May be empty or contain alternative content. Size and background color will propagate to SWF
(2) **shareData:** Whether to share data across browsers. *(optional)*
(3) **useCompression:** Whether to compress data when stored. *(optional)*

## Dependencies

SWFStore requires the Yahoo Global Object, Dom, Cookie, Event and Element, as well as SWF and SWFDetect utilities. The SWFStore also uses the `swfstore.swf` file that must reside in the same path as the page. On the client side, SWFStore requires that the user have **Flash 9.0.115** or later installed in their browser, and have not turned off local storage for Flash Player.

## Security Considerations

By default, SWFStore uses a whitelist to determine which pages are allowed to load the swfstore.swf file and manipulate storage. To use this, create an XML file called storage-whitelist.xml in the same directory as the swfstore.swf file. Include any number of allow-access-from nodes, which point to a full URL. Any URL specified here will be allowed access. Be as specific as necessary. For instance, specifying http://www.yahoo.com will allow http://www.yahoo.com/mail and http://www.yahoo.com/preferences. However, only specifying http://www.yahoo.com/preferences would not allow http://www.yahoo.com.

```
<?xml version="1.0" encoding="utf-8"?>
<url-policy>
       <allow-access-from url="http://www.yahoo.com" />
       <allow-access-from url="http://www.yahoo.com/mail" />
</url-policy>
```

## Simple Use Case

```
swfstore.addEventListener("save", onSuccess);
function onSuccess (event) {
   alert("Your username has been stored");
}
swfstore.setItem('my_username', 'Jed90210');
```

Creates a store for a username under the location "my_username". When the item is successfully stored, an alert will pop up.

## Considerations

(1) By default, SWFStore looks for `swfstore.swf` in the same directory as the calling page, and this is not currently configurable. Therefore, you must host the swf on your own server.
(2) SWFStore genrally allows 100kb of storage for a particular domain, by default. If you expect your storage to go above the 100kb limit, an advance call to `setSize()` and notification of your users is recommended.
(3) The local storage files are **unencrypted** binary files, easily accessible in a specific folder on the user's system. Storing user-identifiable, or private data is therefore not recommended. Note that clearing browser cookies does not remove this data.

## Solutions

**Requesting more storage**
To request storage over 100kb, a Flash dialog will pop up in the SWFStore SWF. If your container element is sized to not show the SWFStore, it will need to be resized if more storage is requested. If the current size of the SWF is not at least 215px wide and 138px tall, the dialog will not be able to display and the attempted store will fail.

```
SWFStoreDiv.style.width = "215px";
SWFStoreDiv.style.height = "138px";
```

The following requests 1mb of storage for the domain:

```
swfstore.setSize(1000);
```

If an item is attempted to be stored that is larger than the current available size, this settings dialog will display automatically.

**Removing all data from storage**
If you would like to clear all data from a particular store:

```
swfstore.clear();
```

### YAHOO.util.SWFStore Events:

**success**
**error**
**pending**
**openDialog**
**openExternalDialog**

### YAHOO.util.SWFStore Methods:

**calculateCurrentSize()**
**clear()**
**displaySettings()**
**getItems()**
**getNameAt(index)**
**getLength()**
**getModificationDate()**
**getShareData()**
**setShareData()**
**getTypeOf(str)**
**getTypeAt(int)**
**getUseCompression()**
**getValueOf(str)**
**getValueAt(int)**
**removeItem(str)**
**removeItemAt(int)**
**setItem(str, obj)**
**setSize(int)**
**setUseCompression()**

# YUI Library: TabView

2011-3-21 **v2.9**

## 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

2011-3-21 **v2.9**

## Simple Use Case

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

Places a TreeView 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, oCfg);
```

*Arguments:*

(1) **Element id or reference:** HTML ID or element reference for the element into which the Tree's DOM structure will be inserted. If the given element contains a series of nested ordered or unordered lists, they will be used to build the tree.

(2) **Object literal:** an object containing the full tree definition

## Nodes: Text, Menu, HTML, Date

### Node (abstract base class for all others)

```
YAHOO.widget.TextNode(obj | str oData, Node obj
   oParent);
```

*Arguments:*

(1) **Associated data:** A string containing the node label or an object containing values for any public properties of the node

(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.

### TextNode (for simple labeled nodes):

If `oData` is a string it will be used as the label. If an object, it should contain a `label` property. If no `oData.href` is provided, clicking on the TextNode's will invoke the node's `expand` method.

### 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):

A string containing markup for the node's label or an object containing at least an `html` property

### DateNode (for nodes containing dates):

Same as TextNode, will use Calendar widget for cell editing

## 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* |
| clickEvent | ...when node is clicked | Node clicked and event |
| dblClickEvent | ... when node is double clicked | Node clicked and event |
| enterKeyPressed | ... when Enter key is pressed when a node has the focus | Node obj with the focus |

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

## TreeView object definition

```
var tree = new YAHOO.widget.TreeView("treeDiv1", [
 "label0",
 {type:"text", label: "label1", … , children: [… ]}, …
]);
```

Tree definition is an array containing node definitions. If node definition is a string, a TextNode is build. If an object, it should have a `type` property of `"text"`, `"menu"` or `"html"` or the full name of a node type (i.e., `"HTMLNode"`) plus any other properties as would be provided to a Node constructor. Each node can have an optional `children` property with further node definitions.

## TreeView from existing markup

```
<ul><li>List 0
  <ul><li>List 0-0</li>...</ul>
 </li>
 <li><a href="www.elsewhere.com">elsewhere</a></li>
</ul>
```

## 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);
var node2 = new YAHOO.widget.TextNode("2nd", root);
node2.isLeaf = true; //leaf node, not dynamic
tree.render();
```

## Dependencies

TreeView requires Yahoo, Dom and Event. Animation is optional; the the Calendar Control may be used for date editing.

## YAHOO.widget. TreeView: Properties

**id** (str)

## YAHOO.widget. TreeView: Methods

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

## YAHOO.widget.Node: Properties

Inherited by Text, Menu, & HTML nodes

**data** (obj)
**expanded** (b)
**hasIcon** (b)
**href** (str)
**isLeaf** (b)
**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)
**editable (b)**

## YAHOO.widget.Node: Methods

Inherited by Text, Menu, & HTML nodes

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

## Instantiating the Uploader

```
<div id="upldrContainer"
     style="width:200px;height:50px"></div>
<script>
var myUploader = new YAHOO.widget.Uploader(
"upldrContainer", "btnSprite.jpg");
</script>
```

Instantiates a new Uploader object, `myUploader`, which is bound to a div whose id attribute is `'upldrContainer'`. The result will be a Flash button, skinned by *btnSprite.jpg* (an image of four-equal sized button skins stacked vertically: up, hover, down, disabled). If no skin URL is passed, the uploader will render as a transparent button. In both cases, clicking the button invokes the "Browse" dialog.

## Constructor

```
YAHOO.widget.Uploader(str html id,
                      str btnSkinURL);
```

*Arguments:*

(1) **HTML element (string or object):** A reference to an HTML id string or element object binds the Uploader to an existing page element. This parameter is required.

(2) **Button Skin URL (string):** A url of an image consisting of four equal-sized button skins stacked vertically. The top-to-bottom order is: **up, hover, down, disabled.** This parameter is optional.

## Limitations

**1.** The Uploader can only send data to servers in the same security sandbox as the `uploader.swf` file. If `uploader.swf` hosted by `yui.yahooapis.com` is used, then the server must contain a cross-domain permissions file allowing `yui.yahooapis.com` to upload files.
**2.** The intended behavior of the uploader is not to send any cookies with its requests. As a workaround, we suggest either using a cookieless upload method or appending `document.cookie` to the upload request.
**3.** When the uploader is rendered as a transparent layer, it does not respond to keyboard. When the uploader is rendered as an image, it receives "Space" and "Enter" key presses as triggers, but only if the focus is on the uploader component itself.
**4.** The uploader does not support basic authentication.
**5.** The behavior when working through a proxy server is inconsistent and unreliable. Also, when uploading to HTTPS servers, be aware that Flash does not support self-signed certificates.

## Simple Use Case

```
myUploader.setAllowMultipleFiles(true);

myUploader.setFileFilters([{description:"Images", extensions:"*.jpg,
*.gif"}]);

myUploader.addListener("fileSelect", onFileSelect);

function onFileSelect (event:Object) {
   myUploader.uploadAll("YOUR UPLOAD URL");
}
```

Sets the permission to browse for multiple files, and allows the users to select files with either **jpg** or **gif** extension (on Windows, the filtering is suggestive, rather than strict.) The "Browse" dialog with these settings comes up when the uploader control is clicked. When files are selected, they are queued and uploaded to the specified URL using automatic queue management.

## Solutions

**Track upload progress** and log it in the YUI Logger (must be included on the page):

```
myUploader.addListener("uploadProgress", onUploadProgress);
function onUploadProgress (event:Object) {
   YAHOO.log(event.id + ": " + event.bytesLoaded + "/" +
             event.bytesTotal);
}
```

**Send custom variables** in the same POST request as the file submission:

```
myUploader.upload("file0", "YOUR UPLOAD URL", "POST", {var1:
"foo", var2: "bar", var3: "baz"});
```

**Modify the file form field name** from the default "Filedata":

```
myUploader.upload("file0", "YOUR UPLOAD URL", POST, null,
"DifferentFileFieldName");
```

**Accept the file upload using PHP** on the server side:

```
<?php
foreach ($_FILES as $fieldName => $file) {
move_uploaded_file($file['tmp_name'], "./" . $file['name']);
echo (" ");
}
exit();?>
```

## Dependencies

**YUI:** Yahoo Global **Object**, **Dom**, **Event**, **Element** and uploader.swf.
**Client:** Flash **9.0.45** or later installed on their browser.

### YAHOO.widget.Uploader Properties:

SWFURL (str)

### YAHOO.widget.Uploader Events:

contentReady
fileSelect
uploadStart
uploadProgress
uploadCancel
uploadComplete
uploadCompleteData
uploadError

*When transparent:*
mouseUp
mouseDown
rollOver
rollOut
click

### YAHOO.widget.Uploader Methods:

setAllowLogging()
setAllowMultiple()
setFileFilters()
enable()
disable()
upload()
uploadAll()
cancel()
clearFileList()

# YUI Library: YUI Loader Utility

2011-3-21 **v2.9**

## Simple Use Case: YAHOO.util.YUILoader

**Markup:**

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

**Script:**

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

//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 files are loaded.
loader.insert({ onSuccess: function() {
    //Define a callback function that executes code
     //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 as string module names. |
| optional (arr) | Array of optional dependencies as string module names. |
| skinnable (b) | Does this component have a skin CSS file (in the standard skin-CSS directory? |
| after (arr) | Array of modules that are not dependencies, but need to be included above this component if present. |

## Key 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. |
| combine (b) | false | Use the combo service on the Yahoo! CDN to reduce the number of HTTP requests. |
| charset (s) | utf8 | The charset attribute for new node(s). Default: utf-8 |
| 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 (f) | 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. |
| onTimeout (f) | null | |
| timeout (i) | 0 | The number of msecs to wait for a resource to load. |
| 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 externral script whose presence indicates load-completion; for `sandbox()`, this is the root variable for the loaded library. |
| insertBefore (s \| el) | null | Element reference or id of a node that should be used as the insertion point for new nodes. This is useful for making sure CSS rules are parsed in the correct order (place your style overrides in a single style block and insertBefore this node). |

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

## Solutions

```
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, carousel, charts, colorpicker, connection, container, container_core, cookie, datasource, datatable, dom, dragdrop, editor, element, event, fonts, get, grids, history, imagecropper, imageloader, json, layout, logger, menu, paginator, profiler, profilerviewer, reset, resize, selector, simpleeditor, slider, stylesheet, tabview, treeview, uploader, yahoo, yuiloader, yuitest.

## Dependencies

YUI Loader has no dependencies. Its package includes the Yahoo Global Object and the Get Utility. Do not load the YUI Loader file and the Yahoo Global Object or Get Utility files on the same page.

# YUI Library: YUI Test Utility

2011-3-21 **v2.9**

## Simple Use Case: TestCase Object

Create a TestCase object with desired tests, 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("stri
   ng", 5, "Expected string.");
   //fails
```

**isInstanceOf assertion:**
isInstanceOf(oConstructor, oTestObject, sFailureMessage)

```
YAHOO.util.Assert.isInstanceOf(S
   tring, "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.