



The Yahoo! User Interface Library

version 2.2

<http://developer.yahoo.com/yui>

<http://yuiblog.com>

<http://tech.groups.yahoo.com/group/ydn-javascript/>

<http://sourceforge.net/projects/yui/>

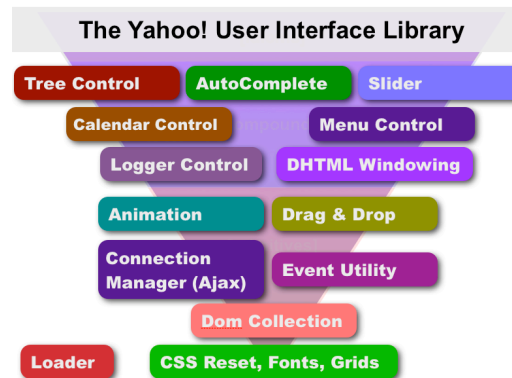
The Yahoo User Interface Library



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

The Yahoo! User Interface Library (YUI) provides a suite of JavaScript foundation classes and UI controls that help developers create rich, dynamic interfaces within the browser. It also provides three core CSS utilities that help to normalize your baseline styles and to build robust, semantic, CSS-driven page layouts utilizing a compact and flexible toolkit.

YUI is an à la carte library, one that allows you to pick and choose which features you want on any given page. The code is compact and efficient. If you enable server-side compression, no single YUI component requires a download of more than 25KB, including



dependencies. Today, YUI includes a CSS foundation, five JavaScript utilities (**Event**, **Dom**, **Connection/Ajax**, **Animation**, **Drag & Drop**) and a range of user interface controls (**AutoComplete**, **Calendar**, **Dialog**, **Menu**, **Panel**, **Slider**, **Tooltip**, **TreeView** and more). It's one of the most richly documented and supported open-source JavaScript toolkits in the world. And Yahoo! is working hard every day to make it even better.

For more on the YUI Library, please visit us on the Yahoo! Developer Network website at <http://developer.yahoo.com/yui>.

Y! YUI Library: Animation

2007-2-20

v2.2

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 object, DOM, and Event.

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 milliseconds}

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

Y! YUI Library: AutoComplete

2007-2-20

v2.2

Simple Use Case (AutoComplete)

Markup:

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

Script:

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

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

Constructor (AutoComplete)

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

Arguments:

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

Constructors (DataSource Classes)

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

Arguments:

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

```
YAHOO.widget.DS_JSFunction(function js function[,
    obj configuration object]);
```

Arguments:

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

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

Arguments:

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

Interesting Moments (AutoComplete)

Event	Arguments (passed via <i>args</i> array)
textBoxFocusEvent/ textBoxBlurEvent	[0] AC instance
textBoxKeyEvent	[0] AC instance; [1] keycode int
dataRequestEvent/ dataErrorEvent	[0] AC instance; [1] query string
dataReturnEvent	[0] AC instance; [1] query string; [2] results array
containerExpandEvent/ containerCollapseEvent	[0] AC instance
itemArrowToEvent/ itemArrowFromEvent	[0] AC instance; [1] element
itemMouseOverEvent/ itemMouseOutEvent	[0] AC instance; [1] element
itemSelectEvent	[0] AC instance; [1] element; [2] item data object or array
selectionEnforceEvent	[0] AC instance
unmatchedItemSelectEvent	[0] AC instance; [1] user selected string
typeAheadEvent	[0] AC instance; [1] query string; [2] prefill string
Subscribe to AutoComplete Custom Events on your AutoComplete instance: <code>myAC.containerExpandEvent.subscribe(myFn[, myObj, bScope]);</code>	

Simple Use Case (DataSource)

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

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

Custom Formatting (AutoComplete)

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

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

Dependencies

AutoComplete requires the YAHOO object, Dom, and Event; Connection Manager and Animation are optional.

YAHOO.widget.
AutoComplete
Properties:

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

YAHOO.widget.
DataSource Properties:

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

YAHOO.widget.DS_XHR
Properties:

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

Y! YUI Library: Browser History Manager [experimental]

2007-4-4

v2.2

Getting Started with Browser History Manger

1. Module Registration and the `register` Method

At least one module must be registered with the Browser History Manager prior to initialization. Use the following method to register a module:

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

Arguments:

- (1) **Module:** An arbitrary, non-empty string identifying the module you are registering.
- (2) **Initial state:** A string representing the initial state of the module (corresponding to its *earliest* history entry). Use `YAHOO.util.History.getBookmarkedState` to determine this initial state when the page is accessed via a bookmark; use in conjunction with a default (non-bookmarked) state of your own.
- (3) **Callback:** A 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.

2. Using the `onLoad` Event

Once you've registered one or more modules, you should subscribe to the Browser History Manager's `onLoad` event. In your handler, initialize your module(s) based on their current state. Use `YAHOO.util.History.getCurrentState` to retrieve the current state of your module(s).

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

Note: When the Browser History Manager `onLoad` event fires, the current state of a module may be different from its initial state, because the `onLoad` event is fired every time the page loads, and this includes the case when the user returns to the page using the back button. In this case, the current state of a module is the last state it was in before the user left the page.

3. Initializing the Browser History Manager

With steps 1 and 2 done, you must initialize the Browser History Manager within a script block inside the `<body>` of the document.

```
YAHOO.util.History.initialize()
```

Note: Call `initialize` inside a script block directly following the opening `<body>` tag.

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 via the back/forward buttons and one that can be bookmarked in the browser. You can create new history entries in your script using the `navigate` method.

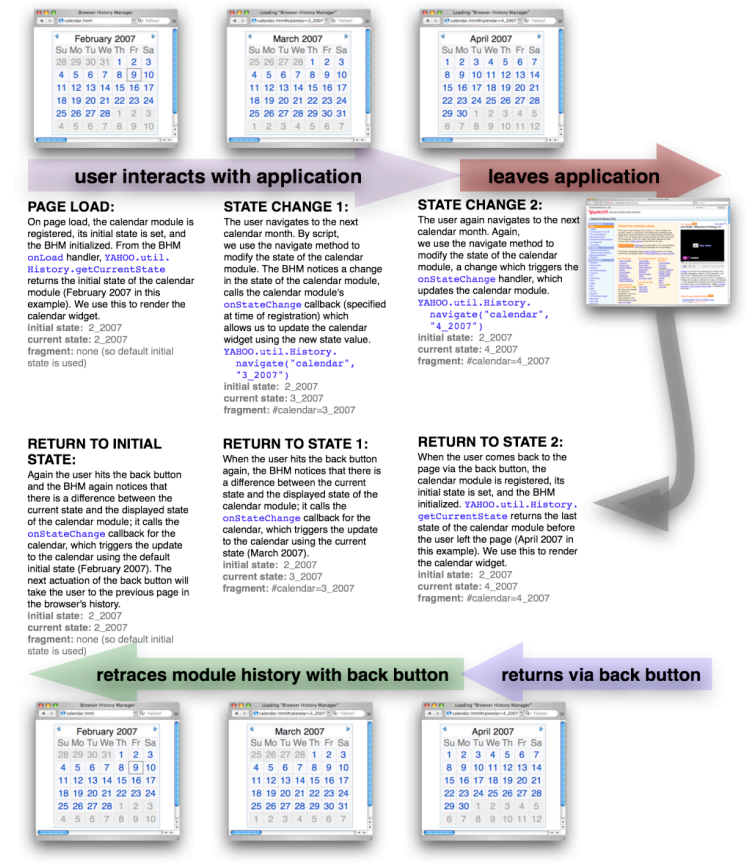
```
YAHOO.util.History.navigate(str module, str new state)
```

Arguments:

- (1) **Module:** The module identifier you used when you originally registered the module.
- (2) **New state:** A string representing the new state of the module.

Note: The `navigate` method returns a Boolean indicating whether the new state was successfully stored.

A Sample Interaction



YAHOO.util.History
Methods:

```
getBookmarkedState(str module)
    returns str bookmarked state
getCurrentState(str module)
    returns str current state
getQueryStringParameter(str
    param name[, str query string])
    returns str param value
initialize()
navigate(str module, str state)
    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.

Note

Browser History Manager is an experimental component. We are excited to hear your feedback about the project, but please note that this component will likely evolve significantly before it is ready for production use.

Y! YUI Library: Button

2007-2-20a

v2.2

Simple Use Cases: YAHOO.widget.Button

Create a Button instance using existing markup:

Markup:

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

Script:

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

Create a Button instance using script alone:

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

Constructor: YAHOO.widget.Button

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

Arguments:

- (1) **element**: HTML ID or HTMLElement of existing markup to use when building Button. If neither, this is treated as the Configuration object.
- (2) **configuration object**: JS object defining configuration properties for the Button instance. See Configuration section for full list.

Simple Use Cases: YAHOO.widget.ButtonGroup

Create a ButtonGroup instance using existing markup:

Markup:

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

Script:

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

Constructor: YAHOO.widget.ButtonGroup

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

Arguments:

- (1) **element**: HTML ID or HTMLElement of existing markup to use when building ButtonGroup. If neither, this is treated as the Configuration object.
- (2) **configuration object**: JS object defining configuration properties for the ButtonGroup instance. See Configuration section for full list.

Key Interesting Moments in Button

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

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

Key Interesting Moments in ButtonGroup

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

Event:	Event Fields:
checkedButtonChange	type (s), prevValue (s), newValue (s)
All ButtonGroup events are Custom Events (see Event Utility docs); subscribe to these events using addListener (e.g. oButtonGroup.addListener("checkedButtonChange", fn);).	

Key Button Configuration Options

See online docs for complete list of Button configuration options.

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

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
TAG_NAME

YAHOO.widget.Button: Methods

blur()
destroy()
focus()
getForm()
getMenu()
hasFocus()
isActive()
set(option, value)
get(option)

YAHOO.widget.ButtonGroup: Properties

CSS_CLASS_NAME
TAG_NAME

YAHOO.widget.ButtonGroup: Methods

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

Dependencies

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

Y! YUI Library: Calendar and CalendarGroup

2007-2-20

2.2

Simple Use Case: YAHOO.widget.Calendar

Markup:

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

Script:

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

Creates a 1-up Calendar instance set to the current month.

Constructor: YAHOO.widget.Calendar, CalendarGroup

```
YAHOO.widget.Calendar(str newElID, str containerID,  
    [obj config]);
```

Arguments:

- (1) **New element ID:** HTML ID for a new element created by the control to house the Calendar's DOM structure.
- (2) **Container element ID:** HTML ID for an existing but empty HTML element into which the new Calendar will be inserted.
- (3) **Configuration object:** The configuration object that contains the Calendar's settings

Solutions

Render a **1-up calendar** set displaying January 2008, with **Spanish weekdays and monthnames**:

```
myCal = new YAHOO.widget.Calendar("myCalEl",  
    "container",  
    { pagedate: "12/2008",  
      MONTHS_LONG: ["Jenero", "Febrero", ...,  
                    "Diciembre"],  
      WEEKDAYS_SHORT: ["Lu", "Ma", "Mi", "Ju", "Vi",  
                       "Sa", "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:  
var myCal = new YAHOO.widget.Calendar("myPanel", {  
    pagedate: "5/2007" });  
// 2. Via "setProperty" after rendering:  
myCal.cfg.setProperty("pagedate", "5/2007");
```

Interesting Moments in Calendar, CalendarGroup

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]])
beforeSelectEvent	...when a date is selected, before processing change.	none
changePageEvent	...when the Calendar navigates to a new month.	none
clearEvent	...when the Calendar is cleared.	none
deselectEvent	...after a date is deselected.	Array of date fields deselected by the current action (e.g., [[2008,8,1],[2008,8,2]])
All Calendar events are Custom Events (see Event Utility docs); subscribe to these events using their subscribe method: <code>myCal.selectEvent.subscribe(fnMyHandler);</code>		

Calendar Options

Configure options using the constructor or `setProperty`, as described in the "Solutions" section.

Calendar objects include several configurable options, including:

SHOW_WEEKDAYS HIDE_BLANK_WEEKS	SHOW_WEEK_HEADER pages (CalendarGroup only)	MULTI_SELECT
-----------------------------------	--	--------------

Localizing Calendar and CalendarGroup

Calendar instances can be localized via configuration options, including:

MONTHS_SHORT MONTHS_LONG LOCALE_MONTHS	WEEKDAYS_1CHAR WEEKDAYS_SHORT LOCALE_WEEKDAYS	WEEKDAYS_MEDIUM WEEKDAYS_LONG
--	---	----------------------------------

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

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

Dependencies

Calendar requires the YAHOO Object, Event, and Dom.

YAHOO.widget.Calendar,
CalendarGroup: Properties

id (str)

cfg (Config) the Calendar's
configuration object

oDomContainer (el) the
Calendar's outer container eleme

YAHOO.widget.Calendar &
CalendarGroup: Methods

addMonthRenderer(int
month, fn renderer)

addMonths(int) navigates to
current month + int months

addRenderer(s dates, fn
renderer)

addWeekdayRenderer
(int wk, fn renderer)

addYears(int) navigates int
years forward

clear() removes all selected dates

getSelectedDates() return
an array of JS date objects

nextMonth()

nextYear()

previousMonth()

previousYear()

render() renders cal. to page

reset() resets to original state

select(str date)

setMonth(int)

setYear(int)

subtractMonths(int)

subtractYears(int)

Y! YUI Library: Connection Manager

2007-2-20

v2.2

Simple Use Case

```
var callback = {
  success: function(o) {
    document.getElementById('someEl').innerHTML =
      o.responseText;
  }
}

var connectionObject =
  YAHOO.util.Connect.asyncRequest('GET',
    'file.php', callback);
```

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

Invocation (asyncRequest)

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

Arguments:

- (1) **HTTP method (string)**: GET, POST, HEAD, PUT, DELETE, etc. PUT and DELETE are not supported across all A-Grade browsers.
- (2) **URL (string)**: A url referencing a file that shares the same server DNS name as the current page URL.
- (3) **Callback (object)**: An object containing success and failure handlers and arguments and a scope control; see Callback Object detail for more.
- (4) **POST body (string)**: If you are POSTing data to the server, this string holds the POST message body.

Returns: **Transaction object**. { `tId`: int *transaction id* }
The transaction object allows you to interact (via Connection Manager) with your XHR instance; pass `tId` to CM methods such as `abort()`.

Callback Object: Members (All Optional)

1. **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.
2. **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.
3. **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.
4. **scope (obj)**: The object in whose scope your handlers should run.
5. **timeout (int)**: Number of milliseconds CM should wait on a request before aborting and calling failure handler.
6. **upload (fn)**: Handler to process file upload response.

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 <code><body></code> 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 <code>argument</code> 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);
```

Check asynchronous call status:

```
var cObj = YAHOO.util.Connect.asyncRequest('GET',
  'myServer.php', callback);
//status will be true (in progress) or false
var bStatus =
  YAHOO.util.Connect.isCallInProgress(cObj);
```

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; Event Utility is optional and will be used by CM for event attachment if present.

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

Y! YUI Library: CSS Reset, Fonts, and Grids

2007-2-20

v2.2

Recommended Doctype and Render Mode

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

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

YUI CSS Reset: Level the Playing Field

YUI Reset creates a level playing field upon which to explicitly declare your intentions by **normalizing the default rendering of all HTML elements**. It sets margin, padding, and border to 0; font sizes to YUI Font’s default; italic & bold styles to normal; list-style to none, etc.

YUI CSS Fonts: Setting Font Size and Family

Font-size: While still allowing users to adjust their font size, the YUI Fonts package renders all text at 13px by default. To preserve this user feature while maintaining maximum consistency, **specify other sizes using percentages only** (see chart in right column).

```
<style>selector {font-size:122%;} /*16px
*/</style>
```

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

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

Base Page Format

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

```
<body>
  <div id="hd"><!--header / masthead --></div>
  <div id="bd"><!--body--></div>
  <div id="ft"><!--footer--></div>
</body>
```

Inside #bd, if two blocks (.yui-b) exist make one primary by wrapping in an ID yui-main:

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

YUI CSS Grids: Nomenclature

#doc – #doc3	Define the overall width of the page.
.yui-t1 – .yui-t7	Choose the secondary column's width and orientation with one of seven <i>templates</i> .
.yui-g	Standard <i>grids</i> (.yui-g) instruct child <i>units</i> to share available space evenly and can be nested inside other <i>grids</i> for additional subdivision.
.yui-gb – .yui-gf	Five special <i>grids</i> (.yui-gbyui-gf.) are used when child <i>units</i> should occupy space unevenly and when dividing into three sections instead of the standard two. (See chart in right column.)
.yui-u	A <i>unit</i> inside a <i>grid</i> ; generic; obeys parent <i>grid</i> .
.first	Overload the class attribute with “first” to indicate first of a series of <i>grids</i> or <i>units</i> to facilitate use of floats and margins.

YUI CSS Grids: Page Widths

Standard Page Widths: 750px and 950px centered, and 100%-fluid:

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

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

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

Example: Nested 4-Column w/ “first” Identified

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

Font-size Chart

To equal this px size:	Use this percent:
10	77
11	85
12	92
13	100
14	107
15	114
16	122
17	129
18	136
19	144
20	152
21	159
22	167
23	174
24	182
25	189
26	197

Grids: Nesting Grids (yui-g's)

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

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

Grids: Templates (yui-t's)

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

Y! YUI Library: DataTable

2007-4-4

v2.2

Simple Use Case: YAHOO.widget.DataTable

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

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

Script:

```
var myColumnSet = new YAHOO.widget.ColumnSet([
  {key:column1,text:"Column 1"}]);
var myDataSource =
  new YAHOO.util.DataSource(myDataArray);
var myDataTable = new YAHOO.widget.DataTable(
  "myContainer", myColumnSet, myDataSource);
```

Creates a DataTable instance from scratch or existing markup.

Constructor: YAHOO.widget.ColumnSet

```
YAHOO.widget.ColumnSet(array column defs);
```

Arguments:

- (1) **column defs**: JavaScript array of object literal Column definitions.

Constructor: YAHOO.util.DataSource

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

Arguments:

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

Constructor: YAHOO.widget.DataTable

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

Arguments:

- (1) **el**: HTML ID or HTMLElement for a DataTable container. May be empty or already contain <table> markup.
- (2) **ColumnSet**: ColumnSet instance.
- (3) **DataSource**: DataSource instance.
- (4) **configuration object**: An optional object literal defines property values of an DataTable instance.

Key Interesting Moments in DataTable

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

Event:	oArgs Properties:
cellClickEvent, cellDoubleClickEvent, cellMouseDownEvent, cellMouseoutEvent, cellMouseoverEvent	oArgs.event (HTMLElement) oArgs.target (el)
columnResizeEvent	oArgs.target (el)
columnSortEvent	oArgs.column (YAHOO.widget.Column) oArgs.dir (string) "asc" "desc"
headCellClickEvent, headCellDoubleClickEvent, headCellMouseDownEvent, headCellMouseoutEvent, headCellMouseoverEvent	oArgs.event (HTMLElement) oArgs.target (el)
cellEditEvent	oArgs.target (el) oArgs.newData (object) oArgs.oldData (object)
rowDeleteEvent	oArgs.rowIndexes (number[])
rowSelectEvent, rowUnselectEvent	oArgs.row (el) oArgs.record (YAHOO.widget.Record)
rowUpdateEvent	oArgs.rowIds (string[])
tableCellClickEvent, tableCellDoubleClickEvent, tableCellMouseDownEvent, tableCellMouseoutEvent, tableCellMouseoverEvent	oArgs.event (HTMLElement) oArgs.target (el)
All DataTable events are Custom Events (see Event Utility docs); subscribe to these events using "subscribe": (e.g. <code>myDataTable.subscribe("rowSelectEvent", fn);</code>).	

Solutions

Custom cell formatting:

```
var myFnc = function(elCell,oRecord,oColumn,oData)
  { elCell.innerHTML = oData; };
var cols = [ {key:"coll", formatter:myFnc} ];
var myColumnSet = new YAHOO.widget.ColumnSet(cols);
```

Row selection:

```
myDataTable.subscribe("cellClickEvent",
  myDataTable.onEventSelectRow);
```

Basic pagination (defaults to 500 rows per page):

```
var myDataTable = new
  YAHOO.widget.DataTable("myContainer",
    myColumnSet, myDataSource, {paginator:true});
```

YAHOO.widget.Column: Key Properties

abbr
children
className
editor
formatter
key
parser
resizable
sortable
sortOptions.ascFunction
sortOptions.descFunction
text
type
width

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

YAHOO.widget.DataTable: Key Properties

caption
contextMenu
dataSource
fixedWidth
initialRequest
paginator
paginatorOptions
rowSingleSelect
scrollable
sortedBy
summary

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

Dependencies

DataTable requires the YAHOO Object, Event, Dom, and DataSource.

Y! YUI Library: Dialog & SimpleDialog

2007-2-20

v2.2

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 and appended to the document body.
- (2) **Configuration Object:** JS object defining configuration properties for the Dialog. See Configuration section for full list.

The postmethod Property: Dialog & SimpleDialog

postmethod:	Charateristics:
"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: <code>myDlg.hideEvent.subscribe(fnMyHandler);</code> .	

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 (<i>SimpleDialog only</i>).
icon	"none"	Sets url for graphical icon. Six icons are provided: ICON_BLOCK, ICON_WARN, ICON_HELP, ICON_INFO, ICON_ALARM, and ICON_TIP. (<i>SimpleDialog only</i> .)
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: <code>text</code> label for button, <code>handler</code> function to process button click, and <code>isDefault</code> 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, Connection Manager and Drag And Drop are optional (though required for specific features).

YAHOO.widget.Dialog & SimpleDialog: Key Properties

body (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()

Y! YUI Library: Dom Collection

2007-2-20

v2.2

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).
array	getElementsBy (fn method, str tag, str el ref root) Returns a array of HTMLElements that pass the test applied by supplied boolean method. For optimized performance , include a tag and/or root node when possible.
array	getElementsByClassName (str className, str tag, str el ref root) Returns a array of HTMLElements with the given class. For optimized performance , include a tag and/or root node when possible.
obj	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 the current Y position of the element(s) based on page coordinates.
b/array	hasClass (str el ref arr el, str className) Determines whether the element(s) has the given className.

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

Solutions

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

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

Get all elements by attribute:

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

Set element's opacity using setStyle:

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

Dependencies

The Dom Collection requires the YAHOO object.

Useful Dom Methods:

appendChild()
 click()
 cloneNode()
 contains()
 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.

Y! YUI Library: Drag & Drop

2007-2-20

v2.2

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

Methods:

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

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

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

YAHOO.util.DD Properties: scroll (b)	YAHOO.util.DDProxy Properties: centerFrame (b) resizeFrame (b)
--	---

Interesting Moments in Drag & Drop

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

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

Solutions

Add a drag handle to an 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)
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.

Y! YUI Library: Event Utility & Custom Event

2007-4-16

v2.2

Simple Use Case: Adding Event Listeners

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

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

Invocation (addListener)

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

Arguments:

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

Event Utility Solutions

Using `onAvailable`:

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

Using Event's **convenience methods**:

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

Prevent the event's default behavior from proceeding:

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

Remove listener:

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

Dependencies

Event Utility requires the YAHOO Global Object.

Simple Use Case: Custom Event

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

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

Constructor (Custom Event)

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

Arguments:

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

Subscribing to a Custom Event

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

Arguments for **subscribe**:

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

Arguments received by your callback function:

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

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

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

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

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

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

Event Utility Methods:

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

DOM Event Object Properties & Methods:

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

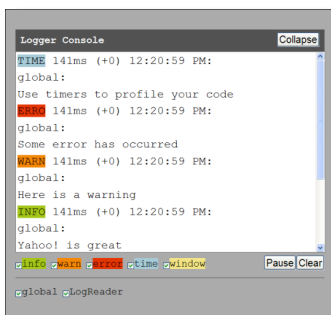
Y! YUI Library: Logger

Simple Use Case (LogReader)

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

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

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



Constructor (LogReader)

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

Arguments:

- (1) **HTML element (string or object):** An optional reference to an HTML id string or element object binds the LogReader to an existing page element.
- (2) **Configuration object (object):** An optional object literal defines LogReader settings. All properties of a LogReader instance can be set via the constructor by using this object.

Logging via console.log()

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



Dependencies

Logger requires the YAHOO object, Dom, Event, and the Logger CSS file; 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 MyModule");
var myInstance = new MyClass();
myInstance.myLogWriter.log("This log message can now
  be filtered by its source, MyClass."); // "MyClass
  of MyModule", the full name of the source, will
  be prepended to the actual log message
```

2007-4-4

v2.2

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 (Pass in other categories to log() to add to this list.)
warn
error
time
window

Y! YUI Library: Menu

2007-2-20a

v2.2

Simple Use Case: YAHOO.widget.Menu

Markup (optional, using standard module format):

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

Script:

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

Creates, renders and shows a menu using existing markup and all default Menu settings.

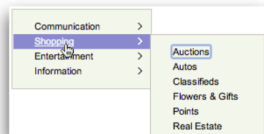
Constructor: YAHOO.widget.Menu

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

Arguments:

- (1) **Element ID:** HTML ID of the element being used to create the Menu. If this element doesn't exist, it will be created and appended to the document body.
- (2) **Configuration Object:** JS object defining configuration properties for the Menu instance. See Configuration section for full list.

Three Types of Menus



Classic Menu

YAHOO.widget.Menu

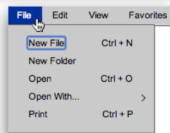
A classic menu is defined by a list of menu items, visible on pageload, each of which can contain sub-menus that fly out on mouseover or on click.



Context Menu

YAHOO.widget.ContextMenu

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



Menu Bar

YAHOO.widget.MenuBar

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

Key Interesting Moments in Menu

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

beforeRenderEvent	renderEvent
renderEvent	beforeShowEvent
showEvent / hideEvent	beforeHideEvent

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

Key Menu Configuration Options

See online docs for complete list of Menu options.

Option (type)	Default	Description
constrain toviewport (b)	true	Forces a menu to remain inside the confines of the viewport.
itemData (a)	null	Array of MenuItem objects 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 milliseconds.
trigger (s o a)	Null	The id(s) or node reference(s) for the element(s) whose contextmenu event triggers the context menu's display.
maxHeight (n)	0	The maximum height (in pixels) for a menu before the contents of the body are scrolled.

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

Key MenuItem Configuration Options

See online docs for complete list of MenuItem options.

Option (type)	Default	Description
checked (b)	false	Renders the item with a checkmark.
disabled (b)	false	If set to true the MenuItem will be dimmed and will not respond to user input or fire events.
helptext (s el)	null	Instructional text to accompany the text for an item.
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

Methods

`addItem(o || s [,i])`
`addItems(o || s [,i])`
`getItem(i [,i])`
`getItems()`
`getItemGroups()`
`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.

Y! YUI Library: Panel

2007-2-20

v2.2

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 and appended to the document body.
- (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",
  "t1", "br"]);
```

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

```
alertMove = function(type, args) {
  alert(args[0] + ", " + args[1]);
}
myPanel.moveEvent.subscribe(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 representing new body content (Note: there are corresponding Header and Footer change events, too).

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

Key Panel Configuration Options

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

Option (type)	Default	Description
close (b)	null	Display close icon.
draggable (b)	null	Make the Panel draggable.
modal (b)	null	Use a modal mask behind Panel when Panel is visible.
visible (b)	true	Sets the "display" style property to "block" (true) or "none" (false).
x, y, and xy (int, int, ar)	null	These properties can be used to set the Panel's "top" and/or "left" styles.
context (ar)	null	Anchors Panel to a context element; format: <code>[el contextEl, s panelCorner, s contextCorner]</code> 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.
constraintto 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: <code>{effect: YAHOO.widget.ContainerEffect.FADE, duration:1}</code>

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()

Dependencies

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

Y! YUI Library: Slider

2007-2-20

v2.2

Simple Use Case

Markup:

```
<div id="sliderbg">
  <div id="sliderthumb"></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:

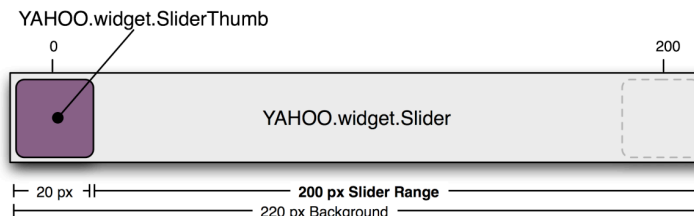
```
mySlider.subscribe("slideEnd", function() {
  alert(this.getValue()); //alerts offset from start
});
```

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} <small>offset from the starting position, one offset per slider dimension</small>

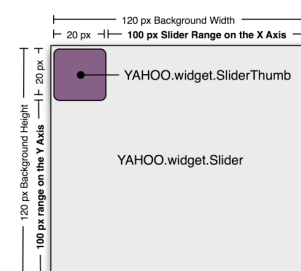
Slider events are Custom Events; subscribe to them by name using the following syntax: `mySlider.subscribe("change", fn);`.

Slider Design Considerations



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

Region Sliders:



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

Dependencies

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

YAHOO.widget.Slider: Factory Methods

`getHorizSlider()`
`getVertSlider()`
`getSliderRegion()`

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

YAHOO.widget.Slider: Properties

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

YAHOO.widget.Slider: Methods

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

YAHOO.widget.SliderThumb:

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

CSS Notes:

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

Y! YUI Library: TabView

2007-2-20

v2.2

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 <code>label</code> .
contentEl (el)	null	The HTMLElement containing the <code>contentEl</code> .
dataSrc (s)	null	Url to use for retrieving content.
cacheData (b)	false	Whether or not data retrieved from <code>dataSrc</code> 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.

Y! YUI Library: TreeView

Simple Use Case

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

Places a Tree control in the HTML element whose ID attribute is "treeDiv1"; adds one node to the top level of the Tree and renders.

Constructor: YAHOO.widget.TreeView

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

Arguments:

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

Nodes: TextNode, MenuNode, HTMLNode

TextNode (for simple labeled nodes):

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

Arguments:

- (1) **Associated data:** A string containing the node label or an object containing str `label`, str `href`, and any other custom members desired. If no `oData.href` is provided, clicking on the TextNode's intrinsic `<a>` tag will invoke the node's `expand` method.
- (2) **Parent node:** The node object of which the new node will be a child; for top-level nodes, the parent is the Tree's root node.
- (3) **Expanded state:** A boolean indicating whether the node is expanded when the Tree is rendered.

MenuNode (for auto-collapsing node navigation):

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

HTMLNode (for nodes with customized HTML for labels):

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

Arguments:

- (1) **HTML:** A string containing markup for the node's label; no event handlers are provided by default for this markup.
- (2) **Parent node:** See TextNode.
- (3) **Expanded state:** See TextNode.
- (4) **Has Icon:** Stipulates whether the expanded/contracted icon (and its horizontal space) should be rendered for this node.

Interesting Moments in TreeView see docs for complete list

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

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

TreeView DOM Structure

Outer <div> (or other block-level element)

This is the element whose ID you passed in when you instantiated the tree

Root Node Wrapper <div>

Every node is wrapped in a <div>. Upon instantiation, the tree's root node is created; as part of this process, a <div> for the root node is added to the DOM.

Bounding <div> for Root Node's Children

When any node has children, those children are wrapped within a <div> that is a child element of the parent node's bounding <div>.

First Child Node's Wrapper <div>

<table> for first child node's display

<td> icon	<td> label
--------------	---------------

Bounding <div> for First Child Node's Children

First Grandchild's Wrapper <div>

<table> for first child node's display

<td> spc	<td> icon	<td> label
-------------	--------------	---------------

Solutions:

Dynamically load child nodes:

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

Dependencies

TreeView requires the YAHOO global object and the Event Utility.

2007-2-20

v2.2

YAHOO.widget.

TreeView: Properties

id (str)
nodeCount (int)

YAHOO.widget.

TreeView: Methods

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

YAHOO.widget.Node:

Properties

Inherited by Text, Menu, & HTML nodes

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

YAHOO.widget.Node:

Methods

Inherited by Text, Menu, & HTML nodes

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

Y! YUI Library: The YAHOO Global Object

2007-02-20b

v2.2

Simple Use Case: YAHOO Object

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

Usage: YAHOO.namespace()

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

Arguments:

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

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

Usage: YAHOO.log()

```
YAHOO.log(str msg, str category, str source)
```

Arguments:

- (1) **msg**: The message you want to log.
- (2) **category**: Category of message ("info", "warn", "error", and "time" are built-in; new categories can be added on the fly).
- (3) **source**: The source of the message, used for filtering.

Note: YAHOO.log is an alias to YAHOO.widget.Logger.log; it exists in the YAHOO Global Object so that Logger statements don't throw errors when the Logger Control is removed from the page. See Logger for more details.

Usage: YAHOO.augment()

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

Arguments:

- (1) **receiver**: The function whose prototype is going to be augmented.
- (2) **supplier**: The function whose prototype is the source of the augmentation.
- (3-n) **properties**: By default, YAHOO.augment will apply all members of the supplier's prototype to the receiver; arguments 3 through *n* can be used to supply string member names that designate the specific members to be augmented from the supplier to the receiver.

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

YAHOO Object: Default Members

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

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

Usage: YAHOO.extend()

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

Arguments:

- (1) **subclass**: The object you're using to extend the base object ("superclass").
- (2) **superclass**: The base object being extended by the "subclass".
- (3) **overrides**: An object whose members will be added to the subclass prototype, overriding members of the same name if they exist on the superclass prototype.

Solutions

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

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

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

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

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

YAHOO.lang Methods

isArray(any) returns boolean
isBoolean(any) returns boolean
isFunction(any) returns boolean
isNull(any) returns boolean
isNumber(any) returns boolean
isObject(any) returns boolean
isString(any) returns boolean
isUndefined(any) returns boolean
hasOwnProperty(obj, property) returns boolean
augment() see usage section
extend() see usage section

YAHOO.env Method:

getVersion(str yuimodulename) returns module info; see below

Module Info

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

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

YUI module names: animation, autocomplete, calendar, button, connection, container, container_core, datasource, datatable, dom, dragdrop, event, history, logger, menu, slider, tabview, treeview, yahoo.

Dependencies

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

Note: If you wish to use **YAHOO_config**, you must define that object prior to loading the YAHOO Global Object.