The Yahoo! User Interface Library

version 0.11.3

http://developer.yahoo.com/yui
http://yuiblog.com
http://tech.groups.yahoo.com/group/ydn-javascript/
http://sourceforge.net/projects/yui/

YUI Library: Animation

2006-08-25 **v0.11**

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 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()

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

2006-09-13 **v0.11**

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

Subscribe to AutoComplete Custom Events on your AutoComplete instance:

```
myAC.containerExpandEvent.subscribe(myFn[, myObj, bScope]);
```

Simple Use Case (DataSource)

```
var myDataSource = new YAHOO.widget.DS_JSArray(
   ["a", "b", "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 <li> element in the container.

```
myAC.formatResult = function(aResultItem, sQuery) {
   var sKey = aResultItem[0]; //the query match key
   // Additional data mapped by schema
   var attribute1 = aResultItem[1];
   var attribute2 = aResultItem[n];
   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)
prehighlightClassName (string)
useShadow (b)
useIFrame (b)
forceSelection (b)
typeAhead (b)
allowBrowserAutocomplete (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)

YUI Library: Calendar

2006-08-25 **v0.11**

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, Calendar2up

```
YAHOO.widget.Calendar(str newElID, str containerID,
   [str monthYear, str selectedDates]);
```

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) **Month/year:** The month to display upon rendering ("MM/YYYY").
(4) **Selected dates:** Dates to be pre-selected. Dates and date ranges can be comma-separated in this string argument (e.g. "MM/DD, MM/DD-MM/DD,MM/DD/YYYY"). Omit spaces.

Solutions

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

```
myCal = new YAHOO.widget.Calendar("myCalEl",
   "container", "12/2008");
myCal.customConfig = function() {
 this.Config.Locale.MONTHS_LONG = ["Jenero",
   "Febrero", … , "Diciembre"];
 this.Config.Locale.WEEKDAYS_SHORT = ["Lu", "Ma",
   "Mi", "Ju", "Vi", "Sa", "Do"];
}
myCal.setupConfig();
myCal.render();
```

Add renderer for Mexican holidays using a preexisting CSS class named `calHolidays`:

```
var renderHolidays = function(workingDate, cell) {
YAHOO.util.Dom.addClass(cell, "calHolidays");
}
myCal.addRenderer("1/1,1/6,2/5,2/14,2/24,3/5,3/21,4/
   2/2007-4/8/2007,5/1,5/5,5/10,5/18 …",
   renderHolidays);
myCal.render();
```

Key Interesting Moments in Calendar, Calendar2up

Event	Fires...	Arguments:
onSelect	...after a date is selected.	Array of date fields selected by the current action (e.g., `[[2008,8,1],[2008,8,2]]`)
onBeforeSelect	...when a date is selected, before processing change.	none
onChangePage	...when the Calendar navigates to a new month.	none
onClear	...when the Calendar is cleared.	none
onDeselect	...after a date is deselected.	Array of date fields deselected by the current action (e.g., `[[2008,8,1],[2008,8,2]]`)

When using YAHOO.widget.Calendar, event methods can be accessed on the instance: `myCal.onSelect = myFn;`. When using Calendar2up, event methods are set using the setChildFunction method: `myCal.setChildFunction("onSelect", myFn);`.

Calendar Options

Configure options before rendering, or call setupConfig() and re-render; see docs for use with Calendar2up.

Calendar objects include an Options member which is configurable: `myCal.Options.MULTI_SELECT = true;` key properties include:

SHOW_WEEKDAYS	SHOW_WEEK_HEADER	MULTI_SELECT HIDE_BLANK_WEEKS

Localizing Calendar

See docs for use with Calendar2up.

Calendar objects include a Locale member which is configurable for internationalization; Locale's key properties include:

MONTHS_SHORT MONTHS_LONG	WEEKDAYS_1CHAR WEEKDAYS_SHORT	WEEKDAYS_MEDIUM WEEKDAYS_LONG

Applying Locale properties requires overriding the customConfig method (on your instance or in the prototype):

```
myCal = new YAHOO.widget.Calendar("calEl",
   "container");
myCal.customConfig = function() {
 this.Config.Locale.MONTHS_SHORT =
   ["Jan", "Feb", "Mär", "Apr", "Mai", "Jun", "Jul",
    "Aug", "Sep", "Okt", "Nov", "Dez"];
}
myCal.setupConfig();
myCal.render();
```

Dependencies

Calendar requires the YAHOO object, Event, and Dom.

YAHOO.widget.Calendar & Calendar_Core: Properties

minDate (d)
maxDate (d)
pageDate (d) current month year; read-only

YAHOO.widget.Calendar & Calendar_Core: Methods

addMonthRenderer(int *month*, fn *renderer*)
addMonths(int) navigates to current month + int months
addRenderer(s *dates*, fn *renderer*)
addWeekdayRenderer (int *wkd*, fn *renderer*)
addYears(int) navigates int years forward
clear() removes all selected dates
getSelectedDates() returns 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)
setupConfig() initializes configuration options; always do this after changing config, then (re)render
setYear(int)
subtractMonths(int)
subtractYears(int)

YAHOO.widget.Calendar2up & CalendarGroup

Calendar2up shares many of the navigation methods of Calendar. Its key additional methods are these:

setChildFunction(str *fn name*, fn *function*) use this to set event methods with Calendar2up
callChildFunction(str *fn name*) use this to call Calendar methods from a Calendar2up instance

YUI Library: Connection Manager

2006-09-13 **v0.11**

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. Usually GET or POST.
(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 HTTP form data, this string holds the post 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.
getResponse Header[]	Method returns the string value of the specified header label.
getAllResponse Headers	Method returns all available HTTP headers as a string.
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);
```

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*)
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: Dom Collection

2006-08-10 **v0.11**

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()
getElementsBy TagName()
hasAttribute()
hasChildNodes()
insertBefore()
removeAttribute()
removeChild()
replaceChild()
scrollIntoView()
setAttribute()
setInterval()
setTimeout()

Dom Node Properties:

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

Note: These are not exhaustive lists.

YUI Library: Dialog & SimpleDialog

2006-09-20 **v0.11**

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="textbox" 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:	Charteristics:
"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, 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()

YUI Library: Drag & Drop

2006-07-20 **v0.11**

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 target[, str group name, obj
   configuration]);
```

Arguments:

(1) **Element id:** HTML ID 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
endDrag	e	e	mouseup
onMouseUp	e	e	mouseup

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

Solutions

Add a drag handle to an exsting DD object:

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

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

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

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

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

Override an interesting moment method for a DD object instance:

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

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

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

Lock Drag and Drop across the whole page:

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

Switch to Intersect Mode:

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

Drag & Drop Manager: Properties

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

Drag & Drop Manager: Methods

oDD=instance of DragDrop object

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

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

Dependencies

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

YUI Library: Event Utility & Custom Event

2006-06-27 **v0.11**

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 requires the YAHOO 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 is set to listen to the Custom Event and alert the first argument, "hello world", when the event is fired.

Constructor (Custom Event)

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

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.

Subscribing to a Custom Event

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

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

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.

DOM Events

Mouse Events:
click
dblclick
mousedown
mouseout
mouseover
mouseup
mousemove

Keyboard Events:
keydown
keypress
keyup

HTML Events:
load
unload
abort
error
select
change
submit
reset
resize
scroll
blur

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(…**)**
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)
button (i)
cancelable (b)
cancelBubble (b)
***charcode** (i)
clientX (i)
clientY (i)
ctrlKey (b)
currentTarget (el)
detail (i)
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)

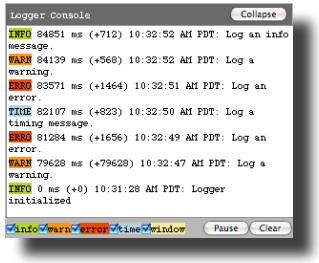
[**use Event Utility method*]

YUI Library: Logger

2006-09-13 **v0.11**

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 or 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 and 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", "mycategory", "my
   source");
```

Logs a message to the default console and to `console.log()`, if enabled. Categories (like "info") 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");
```

Hide and show the logging console:

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

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
```

YAHOO.widget.Logger Static Properties:

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:

logReaderEnabled (b)
footerEnabled (b)
verboseOutput (b)
newestOnTop (b)
thresholdMax (int)
thresholdMin (int)

LogReader Methods:

hide()
show()
pause()
resume()
setTitle()

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

2006-08-30 **v0.11**

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

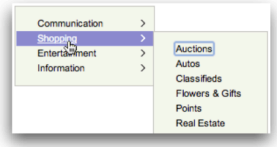
```
YAHOO.widget.Menu(str elId[, obj config]);
```

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.

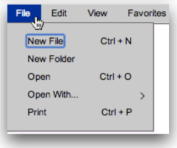
(2) **Configuration Object:** JS object defining configuration properties for the Menu instance. See Configuration section for full list.

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	beforeHideEvent
hideEvent	

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; see Solutions (bottom left) for how to set your options.

Option (type)	Default	Description
zIndex (i)	null	Sets "z-index" style property.
iframe (b)	false (true for IE6 and below)	Prevents background elements from poking through.
constrain toviewport (b)	false	Forces a menu to remain inside the confines of the viewport.
position (s)	"dynamic" ("static" for MenuBar)	Defines how a menu should be positioned. Static menus are visible by default and reside in the normal flow of the document. Dynamic menus are hidden by default, reside out of the normal flow of the document, and can overlay other elements on the screen.
trigger (s || o || a)	null	Defines the DOM element to which a ContextMenu instance is bound. Accepts an element, element id or an array of either.

Key MenuItem Configuration Options

See online docs for complete list of Menu options.

Option (type)	Default	Description
text (s)	null	Text label for the item.
helptext (s || el)	null	Instructional text to accompany the text for an item.
url (s)	"#"	URL for the item's anchor's "href" attribute.
target (s)	null	Value for the "target" attribute of the item's anchor element.
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.
checked (b)	false	Renders the item with a checkmark.

YAHOO.widget.Menu: Properties

parent
element
id

YAHOO.widget.Menu: Methods

addItem(o || s [,i])
getItem(i [,i])
getItemGroups()
insertItem(o || s [,i] [,i])
removeItem(o || i [,i])
setItemGroupTitle(s [,i])
show()
hide()
render([el])

YAHOO.widget.MenuItem: Properties

element
parent
groupIndex
index
value

YAHOO.widget.MenuItem: Properties

focus()
blur()

Dependencies

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

YUI Library: Panel

2006-09-20 **v0.11**

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",
   "tl", "br"]);
```

Subscribe to a Panel Custom Event, listening for changes to the Panel's postion, alerting it's 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: `[el contextEl, s panelCorner, s contextCorner]` with corners defined as "tr" for "top right" and so on.
fixedcenter (b)	false	Automatically center Panel in viewport?
width (s)	null	Sets "width" style property.
height (s)	null	Sets "height" style property.
zIndex (int)	null	Sets "z-index" style property.
constrainto viewport (b)	false	When true, prevents the Panel from being dragged out of the viewport.
underlay (s)	"shadow"	Type of underlay: "shadow", "none", or "matte".
effect (obj)	null	Object defining effect (FADE or SLIDE) to use in showing and hiding Panel: `{effect: YAHOO.widget.ContainerEffect.FADE, duration:1}`

YAHOO.widget.Panel: Properties

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

YAHOO.widget.Panel: Methods

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

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.

YUI Library: Slider

2006-07-12 **v0.11**

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:

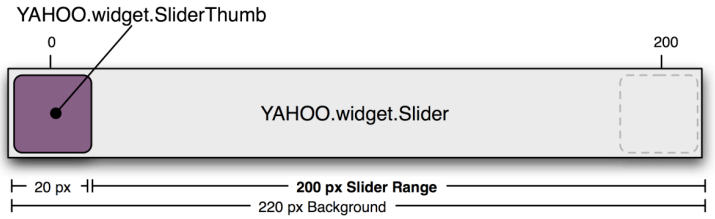
```
mySlider.onSlideEnd = function() {
  alert(this.getValue); //alerts offset from start
}
```

Interesting Moments in Slider

Event	Fires...	Arguments
onSlideStart	...at the **beginning** of a user-initiated change in the thumb position.	none
onSlideEnd	... at the **end** of a user-initiated change in the thumb position.	none
onChange	...each time the thumb position changes during a user-initiated move.	int *firstOffset* [, int *secondOffset*] the offset from the starting position, one offset per slider dimension

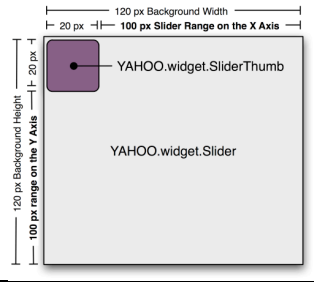
These are methods of Slider's protoype object; you will most commonly use them by overriding them on your instance: `mySlider.onSlideEnd = myFn;`

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

YAHOO.widget.Slider: Properties

animate (b)

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. In a typical implementation of Slider you will not interact directly with the SliderThumb object.

CSS Notes:

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

YUI Library: TreeView

2006-09-13 **v0.11**

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]);
```

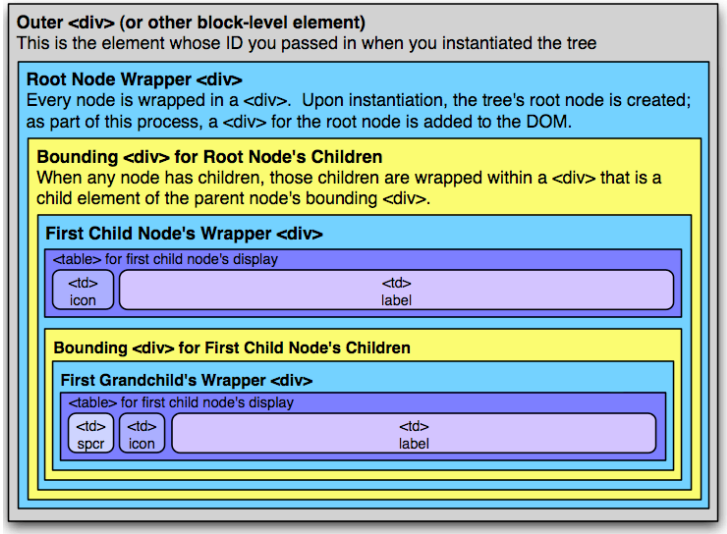
Arguments:
(1) **HTML:** A string containing markup for the node's label; no event handlers are provided by default for this markup.
(2) **Parent node:** See TextNode.
(3) **Expanded state:** See TextNode.
(4) **Has Icon:** Stipulates whether the expanded/contracted icon (and its horizontal space) should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

Interesting Moments in TreeView

Event	Fires...	Arguments
onExpand	...before a node expands; return false to cancel.	Node obj *expanding node*
onCollapse	...before a node collapses; return false to cancel	Node obj *collapsing node*

These are methods of TreeView's protoype object; you will most commonly use them by overriding them on your instance: `tree.onExpand = myFn;`

TreeView DOM Structure



Solutions:

Dynamically load child nodes:

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

Dependencies

TreeView requires the YAHOO global object.

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)
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()
getHTML() includes children
getNodeHTML() sans children
hasChildren()
insertBefore()
insertAfter()
isDynamic()
isRoot()
setDynamicLoad()
toggle()