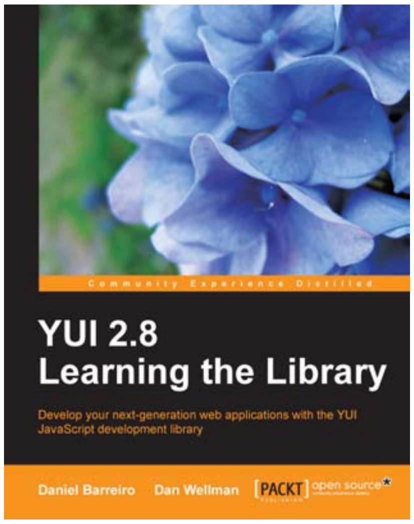
YUI 2.8 Learning the Library

Dan Wellman
Daniel Barreiro


Community Experience Distilled
YUI 2.8
Learning the Library
Develop your next-generation web applications with the YUI JavaScript development library
Daniel Barreiro
Dan Wellman
PACKT open source

Chapter No.7
"Menus"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.7 "Menus"

A synopsis of the book’s content

Information on where to buy this book

About the Authors

Daniel Barreiro (screen name Satyam) has been around for quite some time. The ENIAC was turned off the day before he was born, so he missed that—but he hasn't missed much since. He's had a chance to punch cards, program 6502 chips (remember the Apple II?), own a TRS-80, and see some fantastic pieces of operating equipment in his native Argentina, which might have been in museums elsewhere.

When globalization opened the doors to the world, his then barely usable English (plus an Electrical Engineering degree) put him on the career path that ended in a 5-year job in the Bay Area back in the days of NCSA Mosaic. Totally intrigued by the funny squiggles a friend of his wrote in his plain text editor, full of <'s and >'s, he ended up learning quite a lot about the world of frontend engineering.

It's been a long journey since COBOL and Fortran. Now he lives quite happily semi-retired in the Mediterranean coast close to Barcelona, Spain. When he's not basking in the Mediterranean sun, Satyam can be found among the most prolific and knowledgeable participants in the YUI community on the YUI developer forum. He has also authored the ProgressBar component and took over responsibility for the TreeView component of the YUI Library.

This book wouldn't exist had the guys in the YUI Team not done and continued doing such a magnificent work. Thanks to them for sharing their work with the community and for having invited me to participate; this has taught me lots about good programming practices.

Dan Wellman is an author and web developer from the UK. He works full-time for Design Haus, a local family-run digital agency, and specializes in creating fast, effective frontends using the latest technologies and solutions. In the evenings he writes books and tutorials on many aspects of web development, and spends time with his wife and four children. This is his fourth book.

Many thanks to Eamon O'Donoghue for the use of his stunning artwork that helped bring several of the examples in this book to life.

YUI 2.8 Learning the Library

The YUI Library has grown and improved since the first edition of this book. Several components came out of beta or experimental status and most of them will be covered in new chapters or added to related chapters, which will now have more than the two components originally presented.

The coding style has changed and that will be reflected in the examples, which have been modified accordingly.

New developer tools have appeared to make code more reliant, to reduce loading time, to debug, test, and do performance profiling and we will take a look at them.

The biggest change is the release of version 3 of the YUI Library. YUI3 has recently had its first General Availability (GA) release which includes the most basic components, the infrastructure of the whole library. The Widget component, the basis for all the components that will have a User Interface (UI) is still Beta so there are no visual components in GA. YUI3 still has some way to go to achieve the variety of UI components of YUI2 and, above all, the maturity and reliability of its code base. Thus, this book will be based on the current YUI 2.8 release of the library. YUI2 is not dead by any measure; development continues and future releases will come when and if any significant number of changes warrant it.

Loading YUI2 components in YUI3

JavaScript programming is not what it used to be. As we ventured into more complex client-side applications and learned more about the language and the environment it lives in, the programming style has changed. All the examples in the first edition of this book have been updated to the new coding style. The components in the YUI Library show that same change in style, though some of the original ones now feel aged and the newer ones are more flexible and capable. However, for the sake of the innumerable applications out there, the old components cannot be upgraded as backward compatibility must be preserved. To get out of this tight spot, the YUI team decided to branch off with YUI3, incorporating all the techniques learned over these years. However, YUI3 is not backward compatible with YUI2. Version 2 of the library will still be maintained and upgraded and, in fact, since the two versions branched, there have been as many releases in the YUI2 branch as in the YUI3 branch.

Although YUI3 is a much better programming environment than YUI2 and it has a more consistent and capable interface, it still lacks components such as "widgets", that is, those that have a visual interface. Recently, however, the YUI3 loader (and at this point it is not necessary to go deep into what the loader does) has been enabled to load YUI2 components as well. YUI2 can and regularly does coexist with other non-YUI library components and so does YUI3; there was no reason for them to not coexist with each other. Now, this has been made much easier with the ability of the YUI3 environment to load YUI2 components.

This means two things. First, YUI2 still has a long life. By ensuring that they can run in the YUI3 environment, we can mix and match both versions. All the complex visual components in YUI2 that had no equivalent yet in YUI3 can be used in the YUI3 environment. And second, it means that the YUI team can relax and take their time migrating the YUI2 components, ensuring that when they do get released as YUI3 native, they will be high quality, as per our expectations.

WAI-ARIA

The Web Accessibility Initiative's committee of the World Wide Web Consortium has developed the ARIA (Accessible Rich Internet Application) suite of documents to make the rich applications we can build with YUI accessible to people with disabilities. The degree of support for ARIA is not even across all YUI components. Only components with a visual interface the visitor can interact with need ARIA support. Of those, most YUI components do support ARIA—some have it built-in, others need extra subcomponents, and some do not support ARIA at all. Those that do support ARIA often have a few configuration options, though the defaults rarely need any change. In other words, we as programmers usually don't need to concern ourselves with ARIA besides loading the required subcomponents, if any. We won't cover WAI-ARIA in this book just as we don't cover "beta" components, they are both a work-in-progress with its state still changing and, even when it is available, we rarely need to concern ourselves with it beyond activating it.

What This Book Covers

In Chapter 1, Getting Started with YUI, we look at the library as a whole covering subjects such as how it can be obtained, how it can be used, its structure and composition, and the license it has been released under. We also look at a coding example featuring the Calendar control.

In Chapter 2, Creating Consistency with the CSS Tools, we cover the extensive CSS tools that come with the library, specifically the Reset and Base Tools, the Fonts Tool, and the extremely capable Grids Tool. Examples on the use of each tool are covered. We also see how skinning works in the YUI Library.

In Chapter 3, DOM Manipulation and Event Handling, we look at the all-important DOM and Event utilities. These two comprehensive utilities can often form the backbone of any modern web application and are described in detail. We look at the differences between traditional and YUI methods of DOM manipulation and how the Event Utility unites the conflicting Event models of different browsers. Examples in this chapter include how the basic functions of the DOM Utility are used. We also look at the Element Utility, which is the basis of most of the new UI components and any a developer might build.

In Chapter 4, Calling Back Home, client-server communication is the subject, where we look at how the Connection Manager handles all of our XHR requirements. Then we look at the JSON data interchange format, which is becoming a popular alternative to XML, and at the Get Utility, which allows us to break out of the same-origin policy that limits XHR. Examples include obtaining remote data from external domains and the sending and receiving of data asynchronously to and from our own servers.

In Chapter 5, Animation, the Browser History Manager, and Cookies, we first look at how the Animation Utility can be used to add professional effects to your web pages. The chapter then moves on to cover how the Browser History Manager re-enables the back and forward buttons and book marking functionality of the browser when used with dynamic web applications. Finally, we take a brief look at the Cookies Utility, which allows us to persist information at the browser, helping us save user preferences across our site and in between visits.

In Chapter 6, Content Containers and Tabs, we look at the Container family of controls as well as the TabView control. Each member of the Container family is investigated and implemented in the coding examples. We also look at and implement the visually engaging and highly interactive TabView control.

In Chapter 7, Menus, we look at one of the most common parts of any website—the navigation structure. The example looks with the ease at which the Menu control can be implemented. In

Chapter 8, Buttons and Trees, the focus is on the Button family of controls and the TreeView control. We first cover each of the different buttons and look at examples of their use. We then implement a TreeView control and investigate the methods and properties made available by its classes.

In Chapter 9, DataSource and AutoComplete, we deal with the DataSource Utility, which allows us to fetch and parse tabular data from various sources, both local and remote and in various formats and deliver it to other components in a consistent manner. We will then look at one of those components, AutoComplete.

In Chapter 10, DataTable, we will see another user of DataSource, the DataTable control that allows us to display tabular information and efficiently operate on it on the client side with minimal server intervention.

In Chapter 11, Rich Text Editor, we see the Rich Text Editor, which allows our users to create enhanced text as one of the most popular forms of user interaction over the Internet is user-created content.

In Chapter 12, Drag-and-Drop with the YUI, the first part is Drag-and-Drop, one of DHTML's crowning achievements wrapped up in an easy-to-use utility. In the second part of this chapter we look at the related Slider control and how this basic but useful control can be added to pages with ease. We also briefly mention the Resizer and Layout components that allow us to move and resize dynamic panels within the browser.

In Chapter 13, Everyday Tools, we cover several developer tools. We see how the Logger Control is used to view the event execution of other controls and how it can be used to debug existing controls and custom classes. We also look at other resources such as JSLint and the YUI Compressor. We briefly mention other more advanced tools such as YUI Test, Performance Analyzer, and YSlow.

7 Menus

In this chapter, we're going to look at a very common web page element: navigation menus. The Menu widget provides a timesaving and code-efficient solution to common website application requirements.

The skills that you will take away from this chapter include:

- How to implement a basic navigation menu
- How to override the default `sam` skin
- How to create an application-style menu bar
- How to use the `ContextMenu` type

Common navigation structures

All but the most limited of websites must have a mechanism by which visitors can navigate around the pages of the site from the home page. In order to meet accessibility guidelines, several methods of navigation will usually be available, including at least a navigation menu and a site map.

There have been many different implementation styles that have been popular over the years. Before anyone really worried about accessibility or standards compliance, a common way of designing a navigation menu was to use a series of images that linked to other pages of the site, and there was also the popular frame-based navigation structure. While these methods saved the designer a lot of time, effort, and any real skill, they led to hugely increased page load times and a legacy of bad coding practice.

Thankfully, those days have long since passed, and with the continued development of CSS, it's now possible to design an effective navigation structure based on semantic HTML and styled with CSS.

Designing a navigation menu that is effective, robust, and presented effectively can still pose a challenge, and troubleshooting the compatibility of a menu between different browsers can be a very time-consuming process. This is where the YUI steps in.

Instant menus—just add water (or a Menu Control)

The Menu Control is used to add one of several different menus to your website, saving you the chore of adding this almost essential feature yourself. It's another control that takes a complex, difficult, or time-consuming task, and one which is an almost inherent requirement of any website, and packages it up into a convenient and easy-to-use module. The three different types of menu you can create are:

- A standard navigation menu
- An application-style menu bar
- A right-click context menu

The navigation menu can be implemented as either a vertical or horizontal menu and generates a clean and attractive interface, which your visitors can use to navigate to different areas of your site. The navigation model of any site is key to whether using the site is easy and enjoyable; nothing turns off visitors more than a poorly designed or inconsistent navigation structure.

Another type of menu that the Menu Control is able to create is an application-style menu bar, which stretches across the screen horizontally, building on the current trend in the online world to blur the distinction between the browser and the desktop.

As well as taking care of navigation for you, it can also be used to add right-click context (pop-up) menus to any part of your web application, which again can give a web application a definite desktop feel to it.

The Menu Control is very flexible and can be built from existing HTML markup using a clean and logical list structure, or it can be generated entirely through JavaScript and built at runtime. Each of the different menu types is also given a default appearance with the `sam` skin so there is very little that is required to generate the attractive and highly functional menus.

We'll be looking at implementing each of the different types of menu ourselves in just a moment. Before we do this, let's take a quick look at the classes that go together to make the Menu Control.

The Menu classes

This component, much like the Container that we looked at in the earlier chapter, is made up of a small family of different types of menu. There is a range of different classes that work together to bring the functionality of the different types of menu to you.

The three main classes behind the Menu family are:

- `YAHOO.widget.Menu`
- `YAHOO.widget.ContextMenu`
- `YAHOO.widget.MenuBar`

Menu is a subclass of Overlay, part of the Container family, and the other two are subclasses of Menu. Just as each TabView is made of several Tabs, each kind of Menu has a different class for its items:

- `YAHOO.widget.MenuItem`
- `YAHOO.widget.ContextMenuItem`
- `YAHOO.widget.MenuBarItem`

`ContextMenuItem` is simply an alias for `MenuItem` created just for the sake of symmetry. All types of menu items can have a `Menu` as a submenu; neither `ContextMenu` nor `MenuBar` can be nested in a menu item, they are only good at the outermost level.

All the menus are coordinated by `YAHOO.widget.MenuManager`, which listens to events at the document body level and dispatches them to the corresponding menus or menu items using the technique of Event Delegation to the limit; after all, the document body is the furthest out an event can bubble.

Like the other members of the Container family, the constructor for Menu and its subclasses take two arguments; first a reference to existing markup or, if built via code, the `id` we want the Menu to have once rendered. The second argument takes the configuration options, if any, and accepts any configuration attribute as would be expected. However, in Menu, it can also take a couple of properties: `itemData` and `lazyLoad`, while `MenuItem` can also take `value`. We will see what they can be used for shortly.

Menus can be built from existing markup or from code or any combination of both. The required markup for a Menu might seem a little complicated at first but it is intended to work in older clients or for users without JavaScript enabled. This, we know, is called Progressive Enhancement and Menu supports it very well; the CSS style sheet for Menu works whether JavaScript is active or not and by using the correct class names the Menus will look just the same for all our visitors regardless of the capabilities of their browsers.

The markup will usually consist of a series of unordered lists `<ul>`, each of their list items `<li>` containing an anchor element `<a>` that leads to the next non-JavaScript enhanced page and optionally followed by a further nested unordered list. Menus will also read `<select>` elements creating a `MenuItem` for each `<option>`.

When building a Menu from code, we may create and add each individual `MenuItem` to the Menu or we may use the `itemData` configuration option we've just mentioned, which takes an object literal with the description of the whole Menu at once. This is particularly handy for ContextMenus as they hardly make any sense without JavaScript enabled.

Just as with any container, Menus have to be rendered. ContextMenus are usually rendered into `document.body`, as they have no place in the normal flow of the page. If the menu structure is too complex and takes too long to render, the `lazyLoad` option tells the Menu to render just the first level of items, those that would be visible initially, postponing rendering the rest until needed.

Menu subclasses

The `ContextMenu` is a specialized version of the control that provides a menu hidden from view until the element that it is associated with (the trigger element) is clicked with the right mouse button (except in Opera on Windows and OS X which requires the left-click + *Ctrl* key combination). The trigger element is defined using the `trigger` configuration attribute; this is the only configuration attribute natively defined by the `ContextMenu` class, all others are inherited.

The `MenuBar` is similar to the standard Menu, but is horizontal instead of vertical. It can behave like an application-style menu bar, where the top-level menu items must be clicked in order for them to expand, or it can behave more like a web menu where the menu items expand on a simple mouse over and have submenu indicators. This is controlled with the `autosubmenudisplay` Boolean configuration attribute.

The MenuItem class

Each menu type has a subclass representing the individual menu items that form choices within the menu. They will be created automatically when a Menu is built from existing markup or by setting the `itemData` configuration option to a menu description. You will only create individual MenuItems when extending the functionality of an existing menu.

MenuItems have several interesting configuration attributes:

- `checked`: Shows a checkmark to the left of the label, useful for items that toggle in between two states.
- `classname`: Added to the existing if any further styling is required. It can read this from existing markup.
- `disabled`: This item cannot be selected and will be grayed out. When built from markup, it can read this attribute from an `<option>` tag.
- `keylistener`: The key combination (*Shift*, *Control*, or *Alt* + character) that will trigger this item. It can be read from markup.
- `onclick`: The method to be called when this item is clicked.
- `text`: A string to be shown in the label.
- `url`, `target`: The destination page for this item when not handled via code.
- `selected`: The item shows highlighted.
- `submenu`: A nested instance of Menu, an object description of a nested Menu, or a reference to the markup that would produce it.
- `value`: A value associated with this item.

MenuItem subclasses

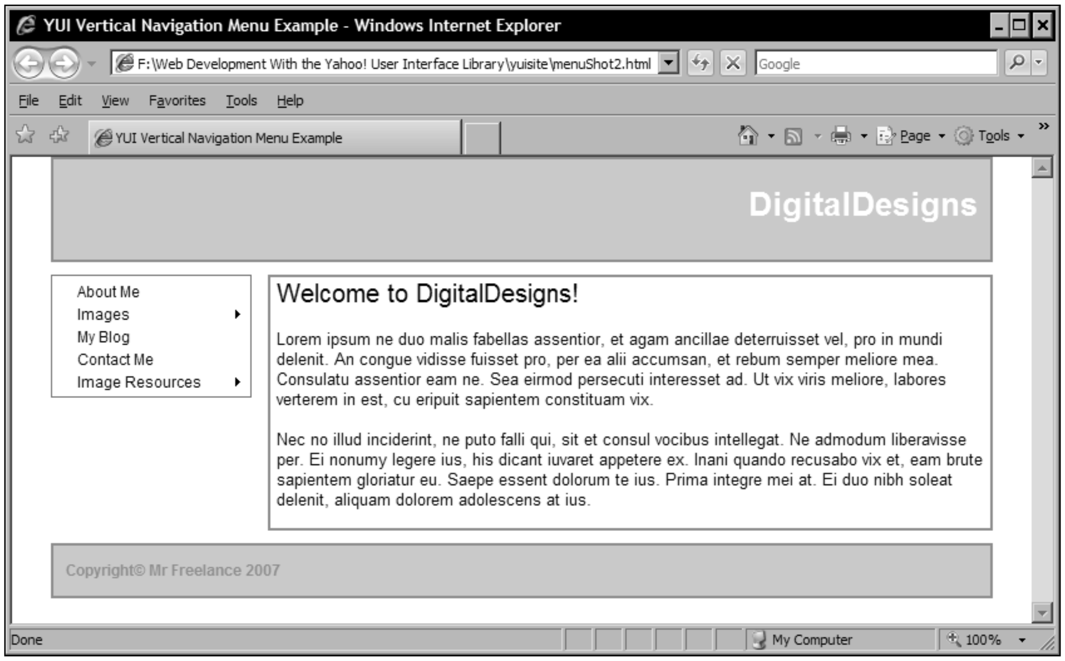
The two subclasses `YAHOO.widget.ContextMenuItem` and `YAHOO.widget.MenuBarItem` both extend the `MenuItem` class, providing a constructor and some basic properties and methods for programmatically working with individual `ContextMenu` or `MenuBar` menu items.

As a matter of fact `ConextMenuItem` is simply an alias for `MenuItem`. `MenuBarItem` has different defaults than `MenuItem` to suit the different layout of the `MenuBar`.

Creating a basic navigation menu

Let's now put together a basic navigation menu and use some of those methods and properties that we looked at in the classes. Our menu will be built from underlying HTML rather than from script.

We'll be enhancing the previous example of the image portfolio by first providing a landing, home page that will welcome us. We'll later add a context menu to the image portfolio itself so that instead of adding an extra button (like **Rate!**) for each option, we'll handle them via menus. Once complete, our landing page should appear like this:


YUI Vertical Navigation Menu Example - Windows Internet Explorer
F:\Web Development With the Yahoo! User Interface Library\yuisite\menuShot2.html
Google
File Edit View Favorites Tools Help
YUI Vertical Navigation Menu Example
Page Tools
DigitalDesigns
About Me
Images
My Blog
Contact Me
Image Resources
Copyright© Mr Freelance 2007
Done
My Computer
100%

Welcome to DigitalDesigns!

Lorem ipsum ne duo malis fabellas assentior, et agam ancillae deterruisset vel, pro in mundi delenit. An congue vidisse fuisset pro, per ea alii accumsan, et rebum semper meliore mea. Consulatu assentior eam ne. Sea eirmod persecuti interesset ad. Ut vix viris meliore, labores verterem in est, cu eripuit sapientem constituam vix.

Nec no illud inciderint, ne puto falli qui, sit et consul vocibus intellegat. Ne admodum liberavisse per. Ei nonumy legere ius, his dicant iuvaret appetere ex. Inani quando recusabo vix et, eam brute sapientem gloriatur eu. Saepe essent dolorum te ius. Prima integre mei at. Ei duo nibh soleat delenit, aliquam dolorem adolescens at ius.

The initial HTML page

Menu requires `yahoo-dom-events.js` and `container-core-min.js` but, as our image portfolio example already contains the full `container-min.js` and all its dependencies, we just need to add `menu-min.js` and its corresponding CSS file, `menu.css`. A nice feature of the Dependency Configurator is that it can also provide us with a customized link to it with our selection set, such as this:

```
http://developer.yahoo.com/yui/articles/hosting/?container&MIN
```

We can save this URL in a comment in our page so that when we need to add extra components, we can start with this URL and then add the new ones. The Dependency Configurator will immediately notice that the full Container files are loaded and will avoid any duplication.

You'll recognize the layout of the image above from *Chapter 2* when we discussed the Grids CSS Tool that was produced by this HTML:

```
<body class="yui-skin-sam">
  <div id="doc" class="yui-t1">
    <div id="hd">
      <h1>DigitalDesigns</h1>
    </div>
    <div id="bd">
      <div id="yui-main">
        <div class="yui-b">
          <h1>Welcome to DigitalDesigns!</h1>
          <p>Lorum ipsum etc...</p>
          <p>Lorum ipsum etc...</p>
        </div>
      </div>
      <div class="yui-b">
         <!-- the menu goes here -->
      </div>
    </div>
    <div id="ft">
      <p class="ftext">Copyright&copy; Mr Freelance 2007</p>
    </div>
  </div>
</body>
```

Except for the textual content in some of the sections this is a boilerplate grid with a narrow 160px sidebar on the left on a 750px document. It has been supplemented by some font choices, background coloring, and borders via CSS besides the styles provided by the `sam` skin. The menu will go where the highlighted comment shows. It was important to show this part of the HTML because it is easy to get lost once the markup for the menu is thrown in.

The underlying menu markup

There are two good reasons to create the main navigation menu from markup. The first is Progressive Enhancement because the markup Menu uses is perfectly workable for any visitor with even the most primitive browser. The second is to allow search engines to index our site. If we create the main menu via code, no search engine would be able to find the rest of the pages in our site.

We add the menu where the comment above shows:

```
<div id="navmenu" class="yuimenu">
  <div class="bd">
    <ul class="first-of-type">
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="aboutme.html">About Me</a></li>
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="images.html">My Images</a></li>
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="blog.html">My Blog</a></li>
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="contact.html">Contact Me</a></li>
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="imagelinks.html">Image Resources</a></li>
    </ul>
  </div>
</div>
```

We called it `navmenu` but any other name would do just as well. We do have to use the rest of the markup as shown. As Menu inherits from Overlay, it uses the standard SMF format, but it uses no head or footer so we just have a body `bd` section and in it, an unordered list (`<ul>`) where each list item is made of an actual link that should work both for old browsers and search engines and a label. All the class names are mandatory as the CSS file for Menu uses them, even the `first-of-type` class name, which signals that this is the top-level menu and not a submenu.

Our menu wouldn't be a proper navigation menu if there weren't, at least, a couple of submenus; it would just be a list of links. Let's add a couple of submenus now:

```
<div id="navmenu" class="yuimenu">
  <div class="bd">
    <ul class="first-of-type">
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="aboutme.html">About Me</a></li>
      <li class="yuimenuitem"><a class="yuimenuitemlabel"
          href="images.html">Images</a>
        <div id="images" class="yuimenu">
```

```
              <div class="bd">
                <ul>
                  <li class="yuimenuitem"><a class="yuimenuitemlabel"
                      href="photography.html">Photography</a></li>
                  <li class="yuimenuitem"><a class="yuimenuitemlabel"
                      href="fantasy.html">Fantasy Art</a></li>
                  <li class="yuimenuitem"><a class="yuimenuitemlabel"
                      href="Corporate.html">Corporate Logos</a></li>
                </ul>
              </div>
            </div>
          </li>
          <li class="yuimenuitem"><a class="yuimenuitemlabel"
              href="blog.html">My Blog</a></li>
          <li class="yuimenuitem"><a class="yuimenuitemlabel"
              href="contact.html">Contact Me</a></li>
          <li class="yuimenuitem"><a class="yuimenuitemlabel"
              href="imagelinks.html">Image Resources</a>
            <div id="links" class="yuimenu">
              <div class="bd">
                <ul>
                  <li class="yuimenuitem"><a class="yuimenuitemlabel"
                      href="http://www.flickr.com">Flickr</a></li>
                  <li class="yuimenuitem"><a class="yuimenuitemlabel"
                      href="http://www.b3ta.com">B3ta</a></li>
                  <li class="yuimenuitem"><a class="yuimenuitemlabel"
                      href="http://yotophoto.com">Yoto Photo</a></li>
                </ul>
              </div>
            </div>
          </li>
        </ul>
      </div>
    </div>
```

The submenus take the same format as the top-level menu—an outer container `<div>` with a class of `yuimenu` forms the basis of the submenu, followed by an inner body `<div>`. Each submenu, like the overall Menu, is built from a standard unordered list (`<ul>`) element where each menu item is composed of a single list item (`<li>`). The structure of the menu is very similar to that of the overall page because it has a distinct body section (`<div class="bd">`) and can also be given `hd` and `ft` sections if required.

Formatting options

Menu offers a few formatting options. A menu or submenu may actually be made of several consecutive unordered lists. Menu will produce a thin dividing bar between each group.

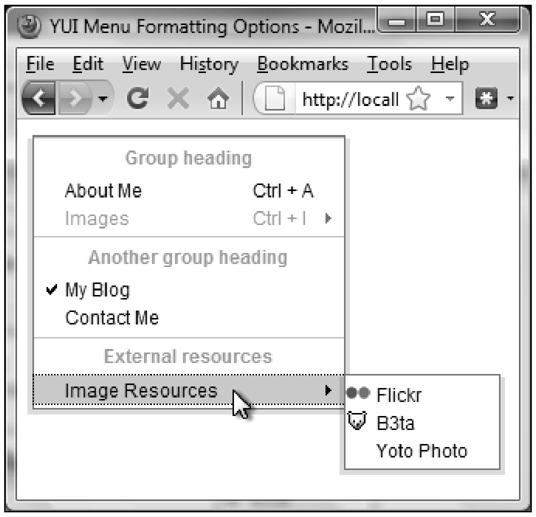
A level-6 heading (`<h6>`) can be added before any `<ul>` to create a heading for each group; Menu won't mind any other heading level but the stylesheet has a suitable style only for an `<h6>`. The heading will be bold but slightly grayed out and it will not respond to a click.

When building the menu from code, these groups, whether with headings or not, can be created by using the optional `groupIndex` argument in `.addItem()` or `.addItems()`. Headers can be added with `.setItemGroupTitle()` using those same indexes. The `.addItems()` method, instead of a simple array of menu items, can also take an array of arrays of menu items, each nested array corresponding to a group.

Hints for keyboard shortcuts can be added to each item. Menu will not automatically respond to such shortcuts as read from the markup, a little more code is required afterwards, but the shortcut will be properly presented to the visitor. For example, in the code earlier we could have added a keyboard shortcut hint to an option by doing:

```
<li class="yuimenuitem">
   <a class="yuimenuitemlabel" href="fantasy.html">
        Fantasy Art<em class="helptext">Ctrl + F</em>
   </a>
</li>
```

This image shows an assortment of such options:



The menu has been divided into three groups, each with its heading, the first two have keyboard shortcuts, and the second item is disabled. The **My Blog** option has the `checked` attribute set and the three items in the bottom submenu each have a class name so that they use the default icon (`favicon.ico`) from the target site as a non-repeating background image; the last site offers no default icon but should it offer one, the menu would immediately show it.

Menu will not automatically respond to the keyboard shortcut as displayed; it cannot parse it from the help text as that might be localized. The `MenuItem` object offers the `keylistener` configuration attribute that can be set to the key combination that will activate this item as if it had been clicked. This attribute uses the same format to describe the key combination as used by the `YAHOO.util.KeyListener` object of the Event Utility.

Creating the Menu object

Implementing a basic menu takes just a little bit of YUI-targeting JavaScript. You can add the following `<script>` block to the page directly above the closing `</body>` tag:

```
<script type="text/javascript">
   YAHOO.util.Event.onDOMReady(function() {
      var menu = new YAHOO.widget.Menu("navmenu", {
                                       position:"static" });
      menu.render();
   });
</script>
```

As soon as we detect the DOM is ready, we create the instance of `Menu` using the `id` of the markup as a reference and then we render it. We need to specify `position` as `static` because we want this menu to always be shown in the page, in contrast to `dynamic` menus that float over the contents. This is all we need; links in the terminal nodes of the Menu hierarchy will be active and navigate when clicked. Items having submenus will unfold the corresponding submenu but they won't navigate to the link destination, such links being reserved for non-JavaScript enabled browsers.

This way of navigating, though, is still too Web 1.0 style. Can we change the contents of the main panel without navigating at all? We can certainly do that. We will copy the main content of the image portfolio from the `panel.html` example of the previous chapter and paste it into the current content, enclosing each in its own `<div>`, like this:

```
<div id="yui-main">
    <div class="yui-b">
        <div id="welcome">
```

```
                <a name="welcome"></a><h1>Welcome to DigitalDesigns!</h1>
                <p>Lorem ipsum … vix.</p><br>
                <p>Nec no illud … at ius.</p>
            </div>
            <div id="fantasyart">
                <a name="fantasyart"></a><h1>Fantasy Art Images</h1>
                <div class="images">
                    <img class="thumb" src="images/image1_thumb.jpg" …>
                    <img class="thumb" src="images/image2_thumb.jpg" …>
                </div>
            </div>
        </div>
    </div>
```

The content has been slightly clipped to fit in this page as signaled by the ellipsis. Now we have a `welcome` section and a `fantasyart` section. Each has an `<h1>` heading preceded by an anchor destination. The link in the menu item for this entry is also changed from `href="fantasy.html"` to `href="#fantasyart"`. Thus, in a non-JavaScript situation, both sections would be visible and the link would serve to jump to that section. We then add the following code:

```
YAHOO.util.Event.onDOMReady(function() {
    var Dom = YAHOO.util.Dom, Event = YAHOO.util.Event;
    Dom.setStyle("fantasyart","display","none");
    var menu = new YAHOO.widget.Menu("navmenu", {
                                   position:"static" });
    menu.render();
    menu.subscribe("click",function(type, oArgs) {
        var ev = oArgs[0],
            menuItem = oArgs[1];

        switch (menuItem.cfg.getProperty("url")) {
          case "#fantasyart":
            Event.stopEvent(ev);
            Dom.setStyle("welcome","display","none");
            Dom.setStyle("fantasyart","display","");
            menu.insertItem({text:"Home",url:"#welcome"},0);
            break;
          case "#welcome":
            Event.stopEvent(ev);
            Dom.setStyle("welcome","display","");
            Dom.setStyle("fantasyart","display","none");
            menu.removeItem(0);
            break;
        }
    });
});
```

We are rendering the menu as we did before, but first we are hiding the `fantasyart` section so only the `welcome` section remains. We add a listener to the click event of the menu. We read both parts of the argument, the raw event object (`ev`) and the `menuItem` that was clicked. We read the `url` configuration attribute for that item and if it is `#fantasyart` we stop the propagation of the event (so it does not navigate on its own), toggle the visibility of both sections, and insert an extra item in the menu, which enables the visitor to return to the home page. This newly added menu item is the one handled in the next `case` where we reverse all we did in this one.

Adding menu items to a menu is easy as shown above. Methods `.addItem()` and `.addItems()` let us append new items to the end. These and `.insertItem()` take a freshly created instance of `MenuItem` (or the corresponding subclass for the other types of menu), a literal object with any configuration attributes, or a simple string to be used as the `text` attribute.

Using the ContextMenu

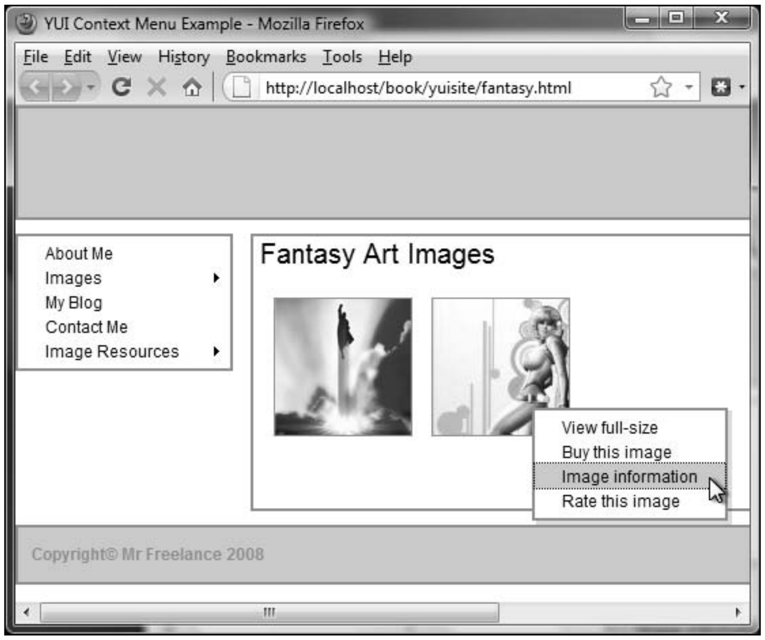
Let's move on to take a look at another menu type—the context, or right-click menu. The navigation menu on our portfolio site provides links to three different image pages. We can use one of these pages to showcase a series of thumbnail images and add right-click functionality to each thumbnail image.

We'll take the page that we used with the Container family controls to show a full view and rate the images and replace that ugly **Rate!** button with a `ContextMenu` with even more options.

ContextMenu scripting

Our custom `ContextMenu` can be created entirely programmatically, without building on existing markup (except for the images of course), making the context menu very easy and quick to implement.

The following screenshot shows how the context menu should appear by the end of this example:


YUI Context Menu Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/book/yuisite/fantasy.html
About Me
Images
My Blog
Contact Me
Image Resources
Fantasy Art Images
View full-size
Buy this image
Image information
Rate this image
Copyright© Mr Freelance 2008

Add the following code to the existing `<script>` block near the bottom of the page. It will create the `ContextMenu`:

```
var context = new YAHOO.widget.ContextMenu(Dom.generateId(), {
    trigger: Dom.getElementsByClassName("thumb"),
    itemData: [
        {text: "View full-size", value:"viewFullSize"},
        {text: "Buy this image", value:"buyImage"},
        {text: "Image information", value:"imageInfo"},
        {text: "Rate this image", value:"rateImage"}
    ],
    lazyLoad:true
});
context.render(document.body);
```

We are creating a new menu, not from markup as we did before, but completely from code. The first argument is not the `id` of an existing section of markup but the `id` we want to give our newly created menu and, as we don't really care, we let the Dom utility's `.generateId()` provide us with one.

The second argument contains the configuration options and we will use the two most important for a `ContextMenu`, `trigger` and `itemData`. `trigger` can be a reference to an HTML element or an array of them; when the visitor right-clicks on any of them, the ConextMenu will show. As all our images have the class name `thumb`, it is easy to search for them by using the Dom method `.getElementsByClassName()`, which returns just the kind of array we need.

The next property is `itemData`, which takes a definition of the menu; its argument is the same as method `.addItems()` would take, an array of strings, an array of object literals with menu properties, an array of instances of `MenuItem`, or an array of arrays, if we wanted grouped items. We have also set the `lazyLoad` property so that the HTML for the menu is not generated until the menu is requested. If we know the menu will be used only occasionally, it is worth saving the time it takes to render it until it is needed.

Finally, there is nothing left but to render it and as it has no place in the normal layout of the page, we do it in `document.body`, which is as good as anywhere else. It's interesting to note that the default context menu supplied by the browser is completely replaced by our YUI context menu; right-clicking on an image and anywhere else in the page will produce different results, which is a small step towards protecting the images from being downloaded if this is something that you wanted to do.

As you can see, the context menu has picked up the custom styling of our main navigation menu without any intervention from us. None of the menu items will actually do anything at this stage because we haven't wired in any additional functionality other than displaying the context menu itself.

Wiring up the backend

Adding functionality for the `ContextMenu` items is extremely easy. The first option in our context menu is to view a full size version of the thumbnail image. We already have this from the Panel example; we triggered it by clicking on the image itself. We'll simply extract the part that actually shows the Panel with the full-size image and call it `showFullSize()`. We also have the code that shows the ratings Dialog. We'll also extract the part that shows the Dialog and call it `askRating()`. We then add the following:

```
context.subscribe("click",function(type, oArgs) {
    var ev = oArgs[0],
        menuItem = oArgs[1],
        image = this.contextEventTarget;

    switch (menuItem.value) {
        case "viewFullSize":
            showFullSize(image);
            break;
        case "rateImage":
            askRating(image);
            break;
        default:
            alert("You've just clicked: " + menuItem.value);

    }
});
```

We listen to the `click` event on the `ContextMenu`. First we extract vital information. The event (`ev`) that we don't use this time as there is no default action to prevent from happening, the `menuItem` to find out what option was selected by the visitor, and finally, what element caused the `ContextMenu` to pop up in the first place, in this case an `image`, upon which to act.

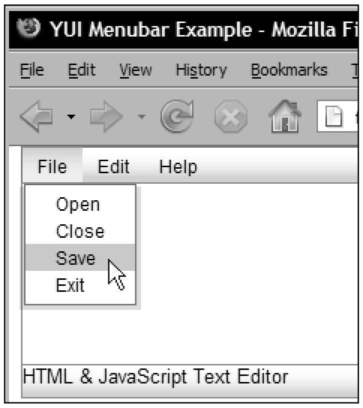
From the `menuItem` we read the `value` property that we assigned to each `menuItem` when we created them. If you were wondering what the purpose of the `value` property was, this is it. It is more reliable than `url` or `text` as any of them might change due to external reasons. The `value` property is our own; if the site is rearranged and URLs changed, or if the site is localized and menu labels get translated, `value` will still remain untouched.

We fork on `value` and call `showFullSize()` or `askRating()` providing the reference to the `image` we want to show or ask the rating about. We really don't have anything to do for all the menu options so we end up with a final `default` to catch the rest of the options.

The application-style MenuBar

The last member of the menu family is the `MenuBar`, which creates a horizontal application-style menu bar. Like both of the other menu types, the menu bar is exceptionally easy to create and work with.

For this example, we can create a basic user interface for a web-based text editor. Creating a fully working online application that can be used to create or open `.txt` files is beyond the scope of this chapter, but we can at least see how easy it would be to create the interface itself. The final page will look like this:


YUI Menubar Example - Mozilla Fi
File Edit View History Bookmarks
File Edit Help
Open
Close
Save
Exit
HTML & JavaScript Text Editor

There is not the slightest chance that this application could possibly work without JavaScript enabled so we'll dispense with using any markup. The body of the page should contain:

```
<body class="yui-skin-sam">
   <div id="doc">
      <div id="hd">
      </div>
      <div id="bd">
        <textarea class="ed" cols="50" rows="12"></textarea>
      </div>
      <div id="ft">
        HTML &amp; JavaScript Text Editor
      </div>
   </div>
</body>
```

The JavaScript code for this example is:

```
YAHOO.util.Event.onDOMReady(function() {
    var Dom = YAHOO.util.Dom, Event = YAHOO.util.Event;

    var menu = new YAHOO.widget.MenuBar(Dom.generateId(), {
        autosubmenudisplay: true ,
        itemdata: [
            {text:"File", submenu:{id:Dom.generateId(), itemdata: [
                {text:"Open",  value:"Open"},
                {text:"Close", value:"Close"},
                {text:"Save",  value:"Save"},
                {text:"Exit",  value:"Exit"}
            ]}},
            {text:"Edit", submenu:{id:Dom.generateId(), itemdata: [
                {text:"Cut",   value:"Cut"},
                {text:"Copy",  value:"Copy"},
                {text:"Paste", value:"Paste"}
            ]}},
            {text:"Help", submenu:{id:Dom.generateId(), itemdata: [
                {text:"About", value:"About"},
                {text:"Help",  url:"help.html", target:"_new"}
            ]}}
        ]
    });

    menu.render("hd");
    menu.subscribe("click",function(type, oArgs) {
        var value = oArgs[1].value;
        if (value) {
            Event.stopEvent(oArgs[0]);
            alert("You've just clicked: " + value);
        }
    });
});
```

When the DOM is ready, we create an instance of `MenuBar`; as we are not using existing markup, we provide generated IDs for all menu instances. In the configuration attributes argument we make submenus display automatically instead of having to click on the main menu for the submenu to show. We then provide the description of our menu structure as an array of object literals, which we assign to `itemdata`.

It is worth noting that `itemdata` and `itemData` are both valid in this context. The property is `itemData` and the configuration attribute, when used with methods `.cfg.setProperty()` or `.cfg.getProperty()` is `itemdata`. When passed as argument in the initial configuration object, it can be either. This is an anomaly that a property, which should use camel-case, can be set as a configuration attribute, which is all lower-case. A similar thing happens with `lazyLoad` and `lazyload`.

All menu items have a `text` property that the visitor will see. Each main item has a `submenu` property, which being a new instance of `Menu` (not of `MenuBar` as neither MenuBars nor ContextMenus can be nested) requires its own `id` and its own set of `itemdata`, each a new array of menu options. The final elements of the menus have a `value` property so we can branch to suitable actions based on them, or a `url` if we simply navigate elsewhere, such as the **Help** item, which will pop up a help text in a new window.

We finally render the menu in the head element of the page, identified with an `id` of `hd` in the three-section SMF format.

To act upon the menu items, we listen for the `click` event, read the `value` property of the item clicked and, if it has any value, we stop the event from propagating and, in this case, we simply show it in an alert box, though in practice this would be a big `switch()` statement or, even more probably, a search into a hash of helper functions indexed by the value.

Keyboard shortcuts can be easily provided, for example, the menu item description for **Edit | Cut** could be given as:

```
{text: "Cut <em class=\"helptext\">Ctrl + X</em>",
            value:"Cut", keylistener: {ctrl:true, keys:88}}
```

Summary

The Menu control is a versatile and easy-to-use control that you'll probably want to implement in many of your web creations. As we've seen, there are three distinct menu types: the standard vertical or horizontal navigation menu, the right-click context menu, and the application-style menu bar. Customizing these different types of menu is as easy as overriding the default styling, if you don't want to stick with the standard skin.