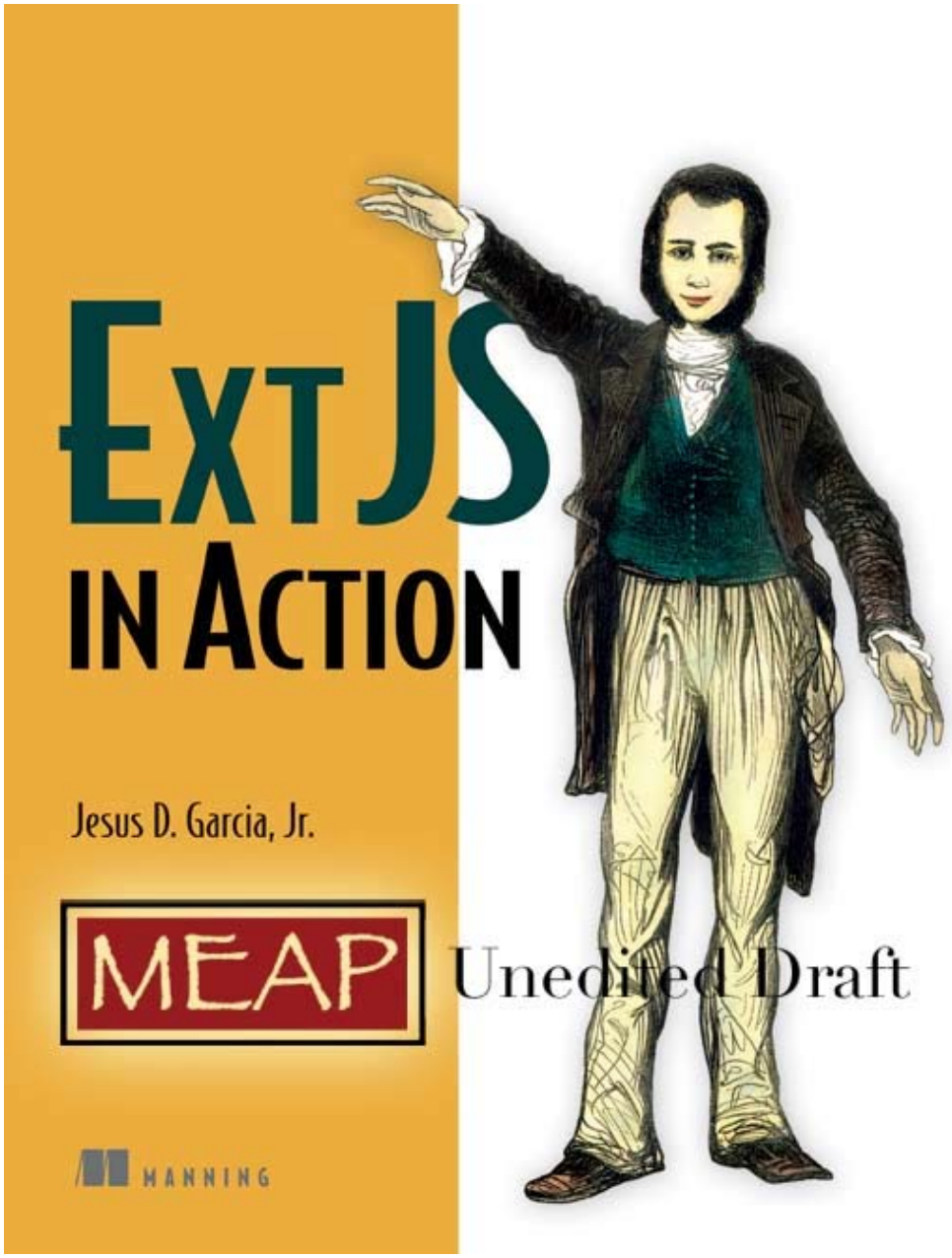

MEAP Edition
Manning Early Access Program

For more information on this and other Manning titles go to
www.manning.com

1

A framework apart

1.1 Why Ext JS

When I was "shopping" for a JavaScript framework, I was looking for one that was easy to learn, had great documentation and a set of prebuilt UI widgets. I spent a lot of time experimenting with quite a few popular frameworks such as Dojo, jQuery, Yahoo User Interface (YUI) and Prototype with Scriptaculous.

After a lot of late nights (and a large number of caffeinated beverages), I found that documentation was one of the biggest problems with the other frameworks, which made it hard to learn. Also, the number of prebuilt widgets was limited and many of them looked immature. With Ext JS, I found everything I needed. With Ext, I found that with the great documentation, I could easily learn the framework. It also included a large variety of mature enterprise-looking widgets. I found myself really enjoying web development again.

1.1.1 It's fun and easy to learn

To me, Ext JS is by far, one of the easiest client side frameworks to learn to use. While it is strongly advised that you are very well familiar with the technologies used in browsers, you need not be an expert JavaScript, HTML or CSS to take advantage of the framework.

One of the things I disliked most about older JavaScript frameworks or 'widgets' is them having a method or constructor that required a long list of parameters, resulting in ugly, hard to read, and difficult to learn code. In the following fictional example, an `Ext.Window` constructor takes the arguments; `title`, `html`, `height`, `width`, `resizable`, `closable`, `modal`, `minHeight` and `minWidth`:

```
var myWindow = new Ext.Window('An ugly panel title',
'This is not the best way of coding', 200, 150, true, true,
false, 400, 400 );
```

You can quickly see how it is hard to decipher by just a glance. Without looking at the API, how does one know what the strings, numbers and Boolean parameters mean? Ext JS comes to the rescue by utilizing a modern approach, where most widget constructor calls use a single parameter, an object, which contains all of the properties or methods you want to set or override. Here is a *real world example* using the same parameters as before:

```
var myWin = new Ext.Window({
   title  : 'An ugly panel title',
   html   : 'This is not the best way of coding',
   height : 200,
   width  : 150
});
myWin.show();
```

The first noticeable advantage is readability. I can quickly detect what attributes are being passed, which speeds up my digestion of the code. The next is not so noticeable, which is the order of parameters. Because we're passing an object, the order of properties becomes irrelevant. With this method, I find it easier to remember what needs to be passed to a constructor. The main disadvantage is the number of lines of code that are required.

Some experienced Ext JS developers enjoy developing extensions and plug-ins, both of which can enhance Ext JS widgets. Here, we can push the envelope with the framework and exercise our creativity. A great example of a plugin is the Window Drawer, shown in Figure 1.1.

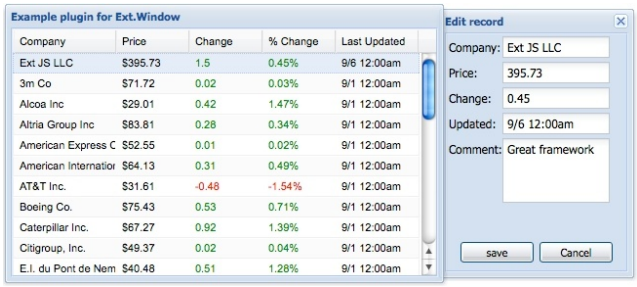

Figure 1.1 This window drawer is a great example of a plug-in that adds to an existing widget, the Ext.Window. When called upon, the drawer (on the right) will slide out behind it's associated parent, providing web applications a similar functionality to using the Mac OS X, which uses window drawers.

In short, an extension is a subclass of a particular widget that generally adds or changes functionality to the base class. A plug-in adds functionality to a component and gets called upon when the parent component is ready. We will go deeper into these topics in greater later in the book where we'll have fun creating our own extensions and plug-ins.

1.1.2 A rich set of widgets

Out of the proverbial box, you're provided a rich set of layout models, widgets, DOM manipulation and event management tools, all of which give you unprecedented control of your application from the widgets down to the browser events. Most widgets are highly customizable, affording you the option to enable, disable features, override, and use custom plug-ins.

Arguably, the most configurable widget is the `GridPanel`, which presents data in a table format much like Microsoft Excel. Because of its design, you can control the `GridPanel` to meet most requirements for a web-based application. The column management component, known as the `ColumnModel`, provides features such as sorting, hiding, drag and drop reordering, resizing and custom cell renderers. The component that manages selection, known as the `SelectionModel`, can be configured to allow the user to select any row or cell and multi or single. What if you have thousands of records to display? This requirement can quickly become a usability and technology nightmare. The `GridPanel` can have the `PagingToolbar` embedded in the widget, which allows users to flip through virtual pages of data, lightening the load on the server, client and user.

The `EditorGridPanel`, a descendant (subclass) of `GridPanel`, can be integrated with any of the Ext form input field widgets to allow your users to modify and submit data asynchornously in the `EditorGridPanel` itself without having to use a separate form.

1.1.3 Quality through centralized development

Centralized development has been one of the key ingredients to the success of Ext, where the development team, which is employed by the Ext JS Company, pushes the framework through the various development cycles with quality control processes in place.

In contrast other JavaScript frameworks are developed or kept up to date by loosely coupled volunteers or independent individuals, where quality control measures may not exist or may not be enforced.

1.1.4 Rich API documentation

The Ext API is a clear example on how you can build a robust RIA using the framework. When opening the API documentation for the first time (Figure 1.2), you get a sense that this unlike any other to date. You can have multiple class documents open at the same time and quickly flip through them without a single page to reload. You can easily filter out the class tree with just a few strokes of the keyboard. If you're connected to the Internet, API documentation itself can be searched easily as well using a filter input box. Aside from the looks of the Ext API, the documentation is clear, concise and ever evolving thanks to the core developers and the Ext JS community volunteers.

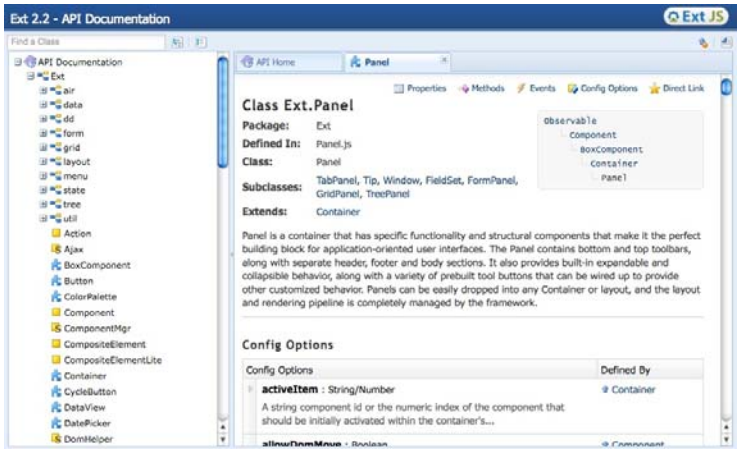

Figure 1.2 The Ext API documentation contains a wealth of information and is a great resource to learn more about components and widgets. It contains most of what you need to know about the API including constructor configuration options, methods, events, properties, component hierarchy and more.

1.1.5 Involved community

The Ext JS online community is a great place to ask peer developers questions regarding the framework and can be accessed via http://extjs.com/forum. It continues to grow and thrive on content submitted by the Ext JS staff as well as volunteers such as myself.

The community promotes some of the best coding techniques and practices, which originates from the framework itself. A great result of this effort is an expanding number of plug-ins and extensions, which plays to the spirit of the framework.

Many people, whom are new to the framework ask, "How does it fit with what I'm doing?" Ext is a flexible framework, but like any tool, knowing where and how to use it is paramount to the success of the tool and the people using it.

1.2 Main course or side dish

Ext JS can be used as a stand-alone client side tool for developing web applications. With its advanced layout management models, developers have full control to manipulate the UI as requirements dictate. A popular option is to use the `BorderLayout` in the `Viewport`, which takes advantage of 100% of the browsers available display space (also called a viewport). Using the `BorderLayout`, the UI can be divided in up to five manageable regions, four of which can be configured to be collapsible and resizable.

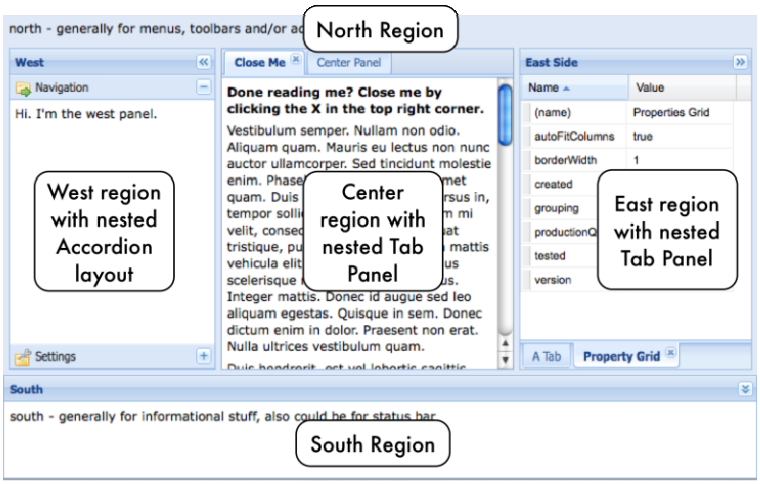

Figure 1.3 This rich Ext example is a great place to start to get yourself familiar with the Border Layout structure. It also demonstrates how you can nest another managed layout such as the 'Accordion' (west region) and tab panels (center and east regions).

If you're not looking to build an entire web application with Ext, you're in luck. Any combination of widgets can be embedded inside an existing web page and or site with relative ease, giving your visitors the best of both worlds.

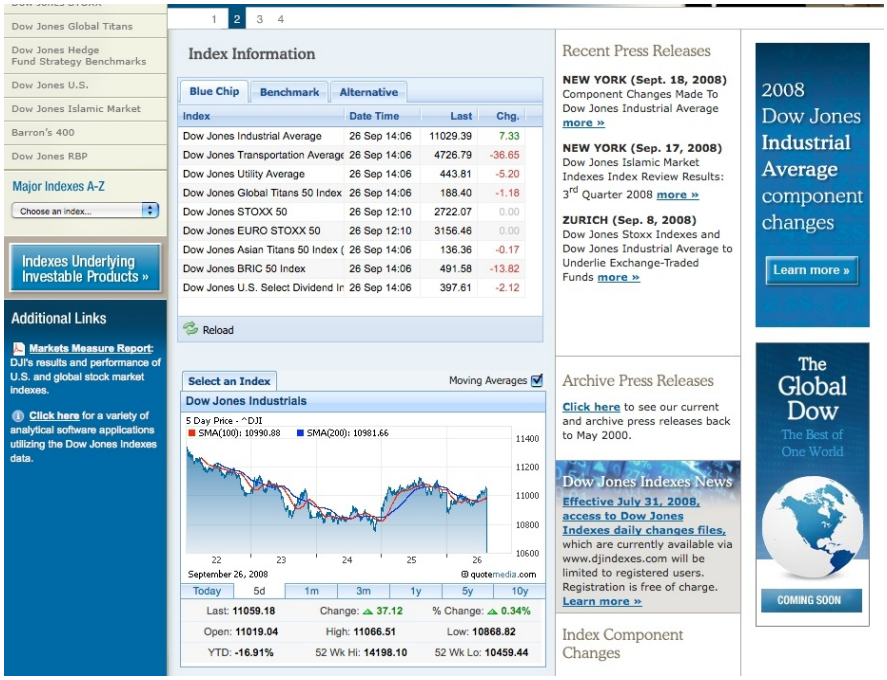

Figure 1.4 The Dow Jones indexes web site, http://djindexes.com, is a great example of Ext JS embedded in a traditional Web 1.0 site.

This Dow Jones indexes web page gives its visitors rich interactive views of data by utilizing a few of the Ext widgets such as the `TabPanel`, `DataGrid`, and `Window`. Their visitors can easily customize the view of the stocks by selecting a row in the main data grid, which invokes an AJAX request to the server, resulting in an updated graph below the grid. The graph view can be modified as well by clicking on one of the time period buttons below it.

1.3 Additions for 3.0

Ext 2.0 introduced some radical changes, which made upgrading from 1.0 absolutely painful. This was mainly because of the introduction to the more modern layout manager and new robust component hierarchy, which broke most of the user developed Ext 1.x code. Thankfully, due to the great engineering of Ext 2.0, the migration from Ext 2.0 to 3.0 is much easier. While the additions to Ext 3.0 are not as drastic, there is excitement about this latest release and is certainly worthwhile discussing some of the additions.

1.3.1 Ext Direct

Due to the communication model of browsers, in order for an application to get updated data from the server, the browser has to poll the server at a predetermined set of intervals. The web server has no way of pushing data to the client, which is one of the major limitations of any RIA that requires up-to-date data. Adding the new `Direct` class, Ext JS has solved this problem by integrating a small Flash object, bringing an enhanced communication mechanism to the browser and allowing the browser to create a sustainable socket connection to a server, which brings the RIA one step closer to a true desktop application. This, of course, requires a supporting server side technology to leverage.

1.3.2 ListView and Charts

A lot of applications require data to be displayed in tabular format. While the `GridPanel` is a great solution, it is expensive for generic data displays that require little or no user interaction. This is where `ListView`, an extension of one of `DataView`, comes to the rescue. Now you can display more data in a tabular format without sacrificing performance.

One of thing that was missing in the 2.0 version of Ext JS, was charts. Thankfully the development team listened to the community and has introduced them for version 3.0. These are a great addition, which adhere to the Ext Layout models.

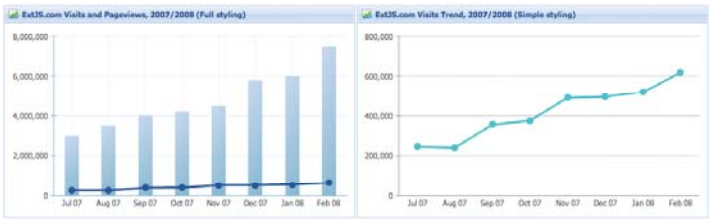

Figure 1.5 These charts now bring rich graphical views of trend data to the framework. It is important to note that this new widget requires Flash.

1.4 What you need to know

While being an expert in web application development is not required to develop with Ext JS, there are some core competencies that developers should have before attempting to write code with the framework.

The first of these skills is a basic understanding of Hyper Text Markup Language (HTML) and Cascading Style Sheets (CSS). It is important to have some experience with these technologies because Ext JS, like any other JavaScript UI library, uses HTML and CSS to build its UI controls and widgets. While its widgets may look like and mimic typical modern Operating System controls, it all boils down to HTML and CSS in the browser.

Because JavaScript is the 'glue' that ties AJAX together, a solid foundation in JavaScript programming is suggested. Again, you need not be an expert, but you should have a solid

grasp on key concepts such as strings, arrays, references, properties and methods. It is a plus if you are very well familiar with Object Oriented JavaScript fundamentals such as objects, classes, and prototypal inheritance.

If you are required to develop code for the server side, you're going to need a server side solution for Ext JS to interact with as well as a way to store data. Most developers choose databases, but some choose direct file system storage as well as well. Naturally, the range of solutions available is quite large. For this book, we'll be using PHP for the server side interpreter and MySQL for the database, as they are the most widely used free code and database combination.

Ext JS is a framework that is built upon modern Object Oriented JavaScript patterns that are widely used by many of the top JavaScript development teams. Since the very beginning, Jack Slocum, the original developer of Ext JS, has encouraged the use of these design patterns for use of the framework.

Before we can begin to use any web 2.0 JavaScript framework, we must take step back and consider how these technologies transform the web page viewing experience.

1.5 Rethinking the web

I can recall my first experience with an AJAX web site, Google maps. Before then, all of the other map websites that were available required a full-page refresh to redraw the map. The result was a site that was usable, not great by any means. Along came Google maps, where I could modify the map without the page refreshing. For me, it was an amazing feeling of freedom to watch the updates to the map live, without the dreaded page refresh.

To understand and appreciate why AJAX development is both exciting and revolutionary, we must discuss and dissect, at the highest levels, how traditional and AJAX web browsing works. After all, the reduction (or elimination) of page reloads is one of the greatest benefits of AJAX.

1.5.1 Traditional web browsing

Since the beginning of the Internet, the basic semantics behind browsing has stayed the same. In its most simplified form, the page request communication is as follows: the browser sends a request, the server responds and the browser redraws the page. While this model works, it is far from a rich experience for the users, which are accustomed to the rapid response of desktop applications.

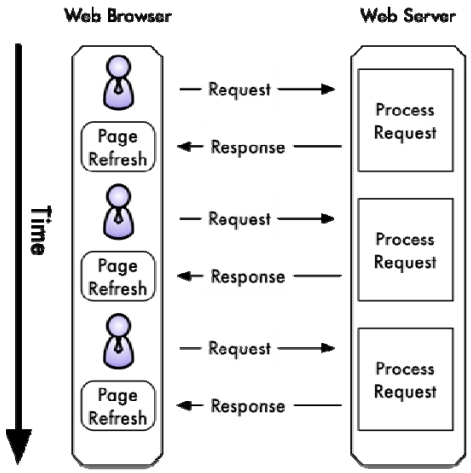

Figure 1.6 Through the life cycle of a traditional web page usage, every interaction that requires server processing by the user, results in a page refresh, which places undue load on the entire browser-server communication model. This is where AJAX comes to the rescue.

Throughout the lifespan of the Internet, there have been technological advances such as the browser cache and proxies that decrease the time for each page load. While these technologies have brought forth drastic improvements (especially for page reloads), they still do not remedy the issue of the page having to be refreshed in order to update content. If we peek under the covers of each page response, we can quickly see why this communication model is inefficient.

1.5.2 The content request cycle

As the browser parses the document that was served, it searches for content that has an external reference, such as images, sound or flash presentations (SWFs) that need to be retrieved and displayed. The browser will then spawn a request for each externally referenced resource until the list is exhausted.

Browser content caches and proxies speed up this interaction by reducing over all network utilization. Browser caches speed up page loads for each page requested by eliminating the need to download redundant content for each site. Proxies speed up the process by intercepting the requests and responding according to whether or not it thinks the content has changed, only fixing the problem of external network communication.

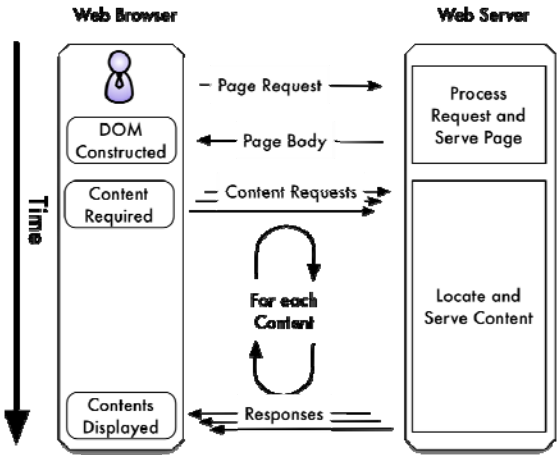

Figure 1.7 A high-level view of a single page request cycle.

If you're a broadband customer, the content retrieval portion of a page load may seem lightning fast. Just because the network is fast, doesn't mean that the system, as a whole, is efficient. There is a heck of a lot that is going on behind the scenes to provide you the non-text content, which must not be overlooked. Each resource requested requires computations to occur throughout the entire client server paradigm.

At a high level, for every external resource, the browser must first check to see if the desired content is in the cache (on disk). If the content is not in the cache, then, it fires off a request to the serve. Each request contains a payload of any parameters required to access the content and headers, which describe a lot of aspects of the browser to the server, which includes what type of content it can accept.

On the flip side, if the content is on disk, the browser must gather information from its local copy of that resource, such as content length or the date it was last modified. It then adds that to the request headers for the server to process.

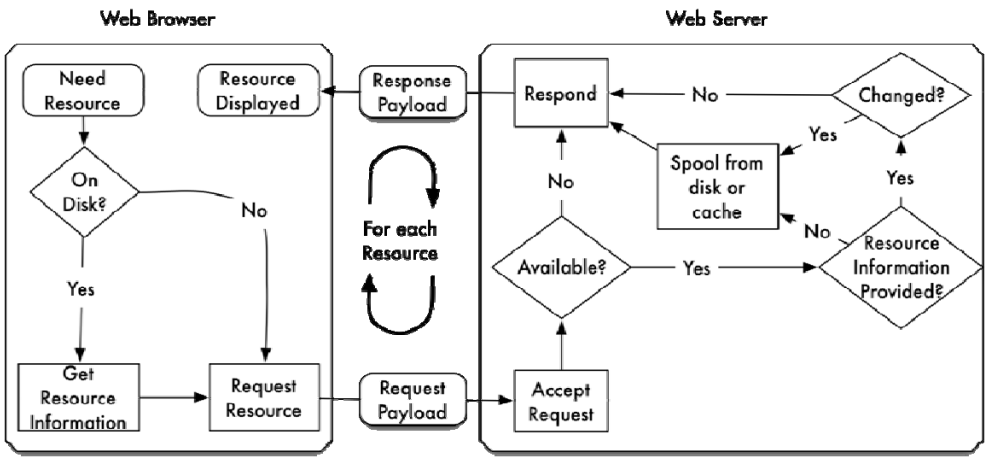

Figure 1.8 A single resource request produces a load across every machine across the wire. Though we don't discuss it, this load includes the network infrastructure that the client and server are using to communicate over. A request, no matter how small, requires a level of routing computation and bandwidth, thus a load on the network infrastructure.

The server accepts the request and processes the headers. The server must first see if the requested content is available. Assuming the server side code doesn't choke; if the resource is not available, the server returns a response with the payload of only headers. This could include any of the 400s codes, such as the common codes "404 Not Found" or "403, Forbidden."

If the client is requesting a resource that is available and it provided information such as the resource's last modified date, the server will check to see if it has changed. If it has not changed, the server will respond with only the payload of only headers with the code "304, Not Modified," which will instruct the browser to use what is on disk. If the resource has changed, the server will stream the resource back to the browser. Multiply this process by the number of items to be requested and it can be easier to visualize how this system leaves room for further improvements.

Now that we understand how traditional browsing works, let's see how adding AJAX to a page can make it more efficient, thus providing a better experience for the user and less of a load on the infrastructure.

1.5.3 AJAX to the rescue!

I cannot think of a single technology to flip the world of web development like AJAX. AJAX is an acronym for Asynchronous JavaScript and XML, which was originally meant to describe the usage of the XML Http Request (XHR) objects in browsers, but is now known to describe a composite of technologies in the browser that make RIAs possible.

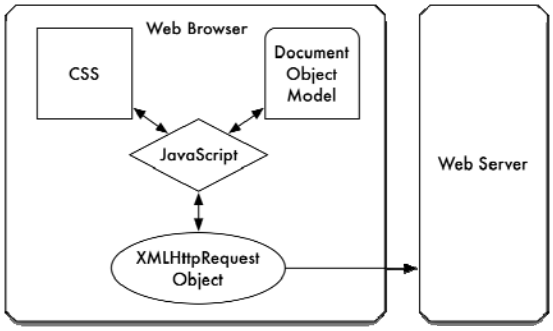

Figure 1.9 AJAX components at a glance.

These technologies comprise of four layers, presentation (XHTML and CSS), content display and user interaction (Document Object Model), data interchange (XML) and JavaScript, which is the glue that ties these technologies together. JavaScript acts as the controller for the XHR Object, manage the DOM, and dynamically modify CSS rules, which change the presentation of content.

NOTE

For all that JavaScript enables it is also its own Achilles Heel. If JavaScript is disabled in the browser, the AJAX model breaks entirely. But being that it is the centerpiece of the AJAX technologies, any AJAX web site will be reduced to the old browsing paradigm.

1.5.4 AJAX applied to browsing

AJAX communication is asynchronous, which means "out of sync." This is important to remember because it shifts the paradigm of web browsing all together. Because the requests are asynchronous, JavaScript can fork off a request and move on to accomplish other tasks while the server processes and responds to the request. Once the server does respond, JavaScript processes the response and provides the appropriate feedback in the form of some content modification. The net result is a reduction in page refreshes, which provides a better experience for the user.

A great example of AJAX interaction can be found on the right side navigation bar of my own site, http://tdg-i.com, as seen in Figure 1.10. When you visit the site for the first time, you get an initial page load, which contains the necessary JavaScript to provide the desired interaction. To achieve the live search, I leveraged the Ext JS ComboBox widget from which my visitors can invoke queries with just a few keystrokes. The act of entering text automatically invokes requests to the web server to be processed. The server then returns simple text and Ext uses CSS and provides the necessary DOM manipulation, which provides excerpts of article content for preview.

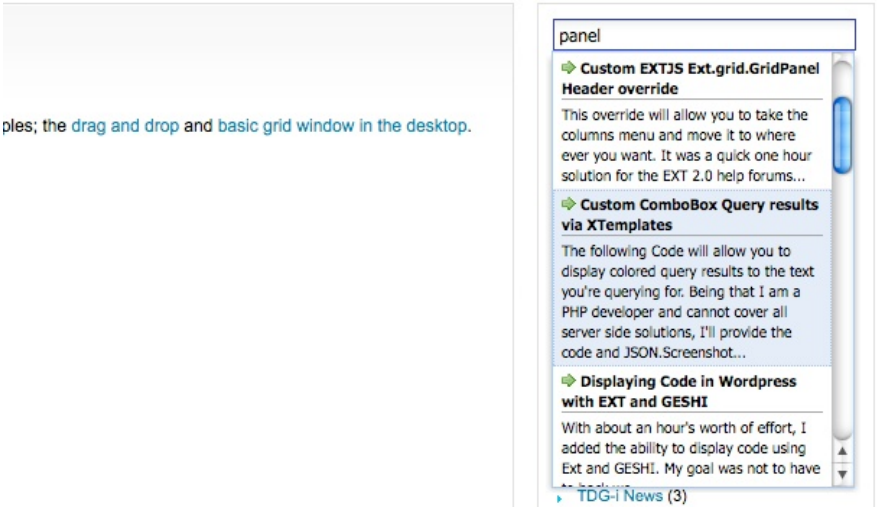

Figure 1.10 The flexible ComboBox widget can be customized via Templates to provide a great way to allow users to input query strings, resulting in very rich and information filled responses.

If the visitor wants to modify the query, they simply modify the text in the input field, triggering another request.

Because the requests are asynchronous, as the visitor is entering text, JavaScript invokes an XHR to the server, resulting in another refresh of the drop down box. The page does not refresh until the visitor actually selects an article to view.

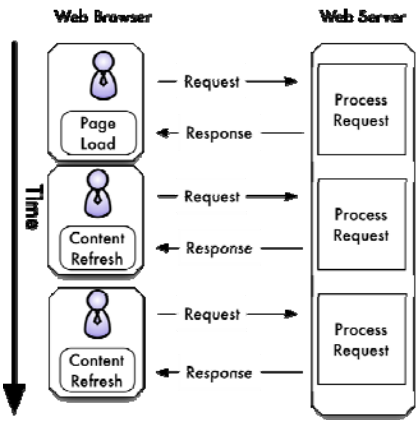

Figure 1.11 AJAX requests are independent of each other and are typically faster than traditional queries

because they only request small amounts of data.

The additional benefits to this AJAX based search are over-all reduced server load and overall network utilization. The server load is reduced because the server only has to transmit text with the results of the query. Because the number of network transactions is down to one – per request, the amount of network utilization decreases as well.

1.5.5 XML or JSON

While the "X" in AJAX means XML, most Ext JS developers leverage JavaScript Object Notation (JSON) to exchange data from the server to the browser. JSON, pronounced like the name JSON, is a lightweight data interchange format published by Douglas Crockford in July 2006 who is, arguably, one of the fathers of modern JavaScript development.

While there are countless debates online on which format is better, in my opinion each exchange format has its own advantages and disadvantages and its own special applications. While some will claim format superiority over the other, I believe that it all boils down to readability and familiarity. I, like a lot of Ext JS Developers, choose JSON for this reason.

1.6 Take it for a test drive

If you're like me, after all of this discussion, you're probably itching to start using Ext JS. So, let's go ahead and dive in.

For this exercise, we're going to create an Ext JS `Window` with one child, a `TabPanel` that will contain two tabs each of which will be an instance of `Panel`. We will leverage the `Panel`'s ability to use AJAX to asynchronously request data, in the form of HTML fragments, from the server to display in its body.

1.6.1 Building the foundation

As a foundation for this test exercise, we'll need to create the two files that contain the fragments to be served up by the server. First of which is a PHP script aptly named "ajaxPage1.php", which will output our first set of HTML fragments to the browser when called upon:

```
<?PHP
sleep(1);
$date = date("m/d/Y");
$time = date("h:j:s A");
$server = $_SERVER['HTTP_HOST'];

echo "<div>Hello from server <b>$server</b>.</div>\n";
echo "<div>My first AJAX page was served on $date at $time.</div>";
?>
```

In `ajaxPage1.php`, first call `sleep`, which will pause the execution of the PHP script, simulating typical network latency. We do this because if you're on a local development web

server like me, responses will be lightning fast. In `ajaxPage1.php`, we assign `$date` and `$time` as references to the server's respective date and time. The date and time will change each time the page is requested, which provide visual confirmation that the fragment is dynamically loading. Next, we assign `$server` to the web server's HTTP host name. We then echo out two `div` tags with the references embedded, resulting in rendered HTML fragments like the following:

```
<div>Hello from server <b>ext2play</b>.</div>
<div> This is my first AJAX page, which was served on 09/30/2008 at
02:30:59 PM.</div>
```

Next, let's create our second file, `ajaxPage2.php`, which will invoke the sleep command. This fragment is different as it contains JavaScript code that will interact with the `TabPanel` itself, dynamically add a tab and apply a cool highlight effect to the page:

```
<?PHP sleep(1); ?>

<div>This fragment contains some magic.</div>

<script type="text/javascript">
Ext.getBody().highlight();

Ext.getCmp('myTabPanel').add({
        title    : 'Peek-a-boo',
        closable : true,
        html     : 'Ext is awesome!'
});

</script>
```

When our second Panel requests this page via AJAX, it will digest the embedded JavaScript. The JavaScript code will first highlight the `document.body`, which will provide visual indication that the included code is being executed by the browser's JavaScript engine. Next, we call `Ext.getCmp` (get component), passing in a string component ID of '`myTabPanel`' and chain a method call, `add`, to the result of that call. We pass in a shortcut object definition for a third tab, which will get translated into a `Panel` that will appear to the right of the second panel, 'AJAX Panel 2' with the title of 'Peek-a-boo!'.

1.6.2 Building our window

We'll now create our base html file, which contains the code for the collection of widgets. Because this is our first real exposure to JavaScript with Ext JS, we're going to take it slow, as I will guide you through each bit as we produce it. It's important to note, however, that all of the broken up code belongs in one file, `chapter01.html`.

We will first create the panels that will be children of the `TabPanel`.

```
function initTabWindow() {
     var ajaxPanel1 = new Ext.Panel({
```

```
        title    : 'AJAX Panel 1',
        autoLoad : 'ajaxPage1.html'
});
```

We first declare the function `initTabWindow`, which will be called upon by Ext.onReady, which is the platform from which we launch all of our Ext code. When `initTabWindow` is called, we first reference a new instance of `Ext.Panel` to `ajaxPanel1`. When calling `new Ext.Panel`, we actually are passing an object as the only parameter. This is commonly known as a configuration object. In that configuration object, we set both `title` and `autoLoad` properties as strings. When `Panel` is initializing, it checks for the `autoLoad` property. If it's present, the `Panel` will fetch the page from the URL in the `autoLoad` property after it's rendered. Let's create the second panel in the set.

```
var ajaxPanel2 = new Ext.Panel({
        title    : 'AJAX Panel 2',
        autoLoad : {
                url      : 'ajaxPage2.html',
                scripts  : true
        }
});
```

Here, we again instantiate a new instance of `Ext.Panel`, but this time, our autoLoad configuration is a nested object, setting the `url` parameter and `scripts` to `true`, which tells the updater to parse embedded JavaScript in the HTML fragment page.

To summarize: `ajaxPanel1` only needs to fetch the page, so we set `autoLoad` to a simple string for the URL of the page, whereas `ajaxPanel2` will fetch and parse the JavaScript, so we set `autoLoad` as an object so we can instruct it to digest the code embedded in the HTML fragment.

Please take a look at the code for `ajaxPanel1` and `ajaxPanel2` can you spot the difference between them? I can wait a moment if you like… Ready? In the first panel, `ajaxPanel1`, we define `autoLoad` as a string whereas in `ajaxPanel2`, we define `autoLoad` as an object. How could this be? How can we define two data types as a parameter for the same property?

This answer highlights on of the areas of flexibility for Ext JS. Just like the autoLoad parameter, many parameters can be set as a string, list (array) or an object. If you are unsure about what to set, consulting the API is a great place to start.

If you're scratching your head right now, don't worry. We'll walk through other configuration options where you have this level of flexibility, and it will make more sense then. Let's move on to create the TabPanel.

```
var tabPanel = new Ext.TabPanel({
        activeTab : 0,
        items     : [
                ajaxPanel1,
                ajaxPanel2
        ]
```

```
    });
```

Here, we create the `TabPanel`. In the configuration object, we set `activeTab` to `0 (zero)`, which instruct the `TabPanel` to activate the first tab after rendering. Next, we set `items` to an array with the references for `ajaxPanel1` and `ajaxPanel2`. The `items` configuration parameter describes the children for any component that will contain or manage any other components within it.

Lastly, we'll create our window, which will contain the `TabPanel`:

```
var win =  new Ext.Window({
            layout   : 'fit',
            title    : 'Ajax tab panel test window',
            width    : 300,
            height   : 150,
            closable : false,
            items    : tabPanel
    });
    win.show();
});

Ext.onReady(initTabWindow);
```

Here, we reference `win` to new instance of `Ext.Window`. Because some of the parameters are self-explanatory, we won't go into all of them. We first set the `layout` to `fit`, which instructs the `Window` to stretch its only child to the dimensions of its body. Next, we set `closable` to false, which instructs the window not to render the close button. We then pass `TabPanel` as the `items` property. After the window is instantiated, we call `win.show`, which renders the window into the browser's viewport and displays it. We then set the closure for the `initTabWindow` with the final `});` . Lastly we call `Ext.onReady`, passing it the name of the method, `initTabWindow`.

1.6.3 Using our window

Now that we've got the coding out of the way, let's open Firefox and browse to the `chapter01.html` file. If there are no errors in your code, you should immediately see a window displayed similar to the one in Figure 1.10. Clicking on the tab for "AJAX Panel 2" will invoke a request for `ajaxPage2.php`. Once `ajaxPage2.php` is loaded, the JavaScript is parsed and executed causing the background to highlight and a new tab will be added with the title of 'Peek-a-boo!'

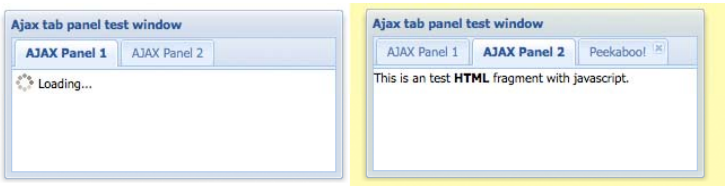

Figure 1.12 This example demonstrates the flexibility of Ext, delivering a powerful AJAX-enabled tabbed Window.

There you have it! Building a UI with Ext, in many ways, is as easy and fun as building this Tab panel window.

1.7 Summary

In this brief introduction, we've covered quite a lot of subjects in preparation for our journey with Ext and learned how flexible it is, allowing for integration with traditional web pages as well as building complex rich Web 2.0 UI based applications. We've learned about some of the exciting new additions to the 3.0 version of the framework and discussed how AJAX plays a role in the modern World Wide Web. By building a fairly complex tab panel window, we got a chance to flex a little of Ext's muscle and (hopefully) had fun doing it.

In the chapters to follow, we will explore how Ext works from the inside out. This knowledge will empower you to make the best decisions when building UIs and better enable you to leverage the framework effectively.