SAMPLE CHAPTER

EXT JS IN ACTION

Jesus Garcia

MANNING

Ext JS in Action
by Jesus Garcia

Chapter 1

brief contents

1 A framework apart

This chapter covers

- A holistic view of Ext JS
- Learning about what's new in 3.0
- Downloading and unpacking the framework source code
- Exploring an Ajax-based "Hello world" example

Envision a scenario where you're tasked to develop an application with many of the typical UI (user interface) widgets such as menus, tabs, data grids, dynamic forms, and styled pop-up windows. You want something that allows you to programmatically control the position of widgets, which means it has to have layout controls. You also desire detailed and organized centralized documentation to ease your learning curve with the framework. Finally, this application needs to look mature and go into beta phase as quickly as possible, which means you don't have a lot of time to toy with HTML and CSS. Before entering the first line of code for the prototype, you need to decide on an approach for developing the frontend. What are your choices?

You do some recon on the common popular libraries on the market and quickly learn that all of them can manipulate the DOM, but only two of them have a mature UI library, YUI (Yahoo! User Interface) and Ext JS.

At first glance of YUI, you might think you need not look any further. You toy with the examples and notice that they look mature but are not exactly professional quality, which means you'll need to modify CSS. No way. Next, you look at the documentation at http://developer.yahoo.com/yui/docs. It's centralized and technically accurate, but it's far from user friendly. You look at all of the scrolling required to locate a method or class. Some classes are even cut off because the left navigation pane is too small. What about Ext JS? Surely it has to be better, right? What alternatives do we have?

In this chapter, we'll take a good look at Ext JS, and you'll learn about some of the widgets that compose the framework. After we finish the overview, you'll download Ext JS and take it for a test drive.

1.1 Looking at Ext JS

Having to develop an RIA with a set of rich UI controls, you turn to Ext JS and find that, out of the proverbial box, Ext JS provides a rich set of DOM utilities and widgets. Although you can get excited about what you see in the examples page, it's what is under the hood that's most exciting. Ext JS comes with a full suite of layout management tools to give you full control over organizing and manipulating the UI as requirements dictate. One layer down exists what's known as the Component model and Container model, each playing an important role in managing how the UIs are constructed.

> **Component and Container models**
>
> The Component and Container models play a key role in managing UIs with Ext JS and are part of the reason Ext JS stands out from the rest of the Ajax libraries and frameworks. The Component model dictates how UI widgets are instantiated, rendered, and destroyed in what's known as the component lifecycle. The Container model controls how widgets can manage (or *contain*) other child widgets. These are two key areas for understanding the framework, which is why we'll be spending a lot of time on these two topics in chapter 3.

Almost all UI widgets in the framework are highly customizable, affording you the option to enable and disable features, override functions, and use custom extensions and plug-ins. One example of a web application that takes full advantage of Ext JS is Conjoon. Figure 1.1 shows a screenshot of Conjoon in action.

Conjoon is an open source personal information manager and can be considered the epitome of web applications developed with Ext JS. It uses just about all of the framework's native UI widgets and demonstrates how well the framework can integrate with custom extensions such as `YouTubePlayer`, `LiveGrid`, and `ToastWindow`. You can get a copy of Conjoon by visiting http://conjoon.org.

You've learned how Ext JS can be used to create a full-page web application. But what if you have an application that's already in production? Next, you'll learn how Ext JS can be integrated into existing applications or websites.

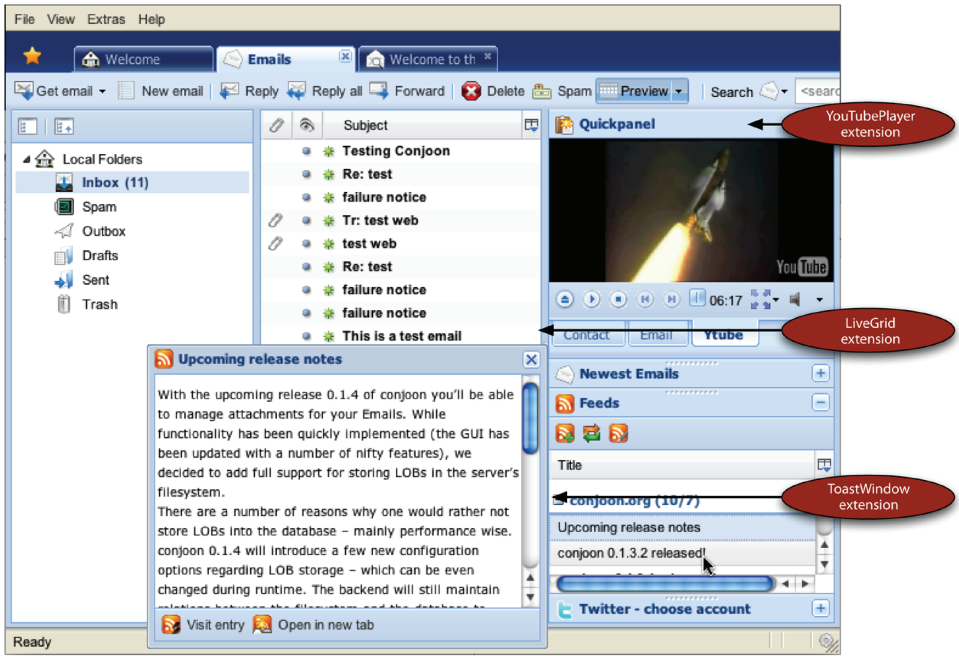

Figure 1.1 Conjoon is an open source personal information manager that's a great example of a web application that uses the framework to manage a UI that leverages 100 percent of the browser's viewport. You can download it at http://conjoon.org/.

1.1.1 Integration with existing sites

Any combination of widgets can be embedded inside an existing web page and or site with relative ease, giving your users the best of both worlds. An example of a public-facing site that contains Ext JS is the Dow Jones Indexes site, http://djindexes.com. Figure 1.2 shows Ext JS integrated with a page at djindexes.com.

This Dow Jones Indexes web page gives its visitors rich interactive views of data by utilizing a few of the Ext JS widgets such as the `TabPanel`, `GridPanel`, and `Window` (not shown). Its visitors can easily customize the view of the stocks by selecting a row in the main `GridPanel`, which invokes an Ajax request to the server, resulting in an updated graph below the grid. The non-Ext JS graph view can be modified as well by clicking one of the time period buttons below it.

You now know that Ext JS can be leveraged to build single-page applications or can be integrated into existing multipage applications. But we still haven't satisfied the requirement of API documentation. How does Ext JS solve this?

1.1.2 Rich API documentation

When opening the API documentation for the first time (figure 1.2), you get a sense of the polish that the framework has. Unlike competing frameworks, the Ext JS API Documentation leverages its own framework to present a clean and easy-to-use documentation tool that uses Ajax to provide the documentation.

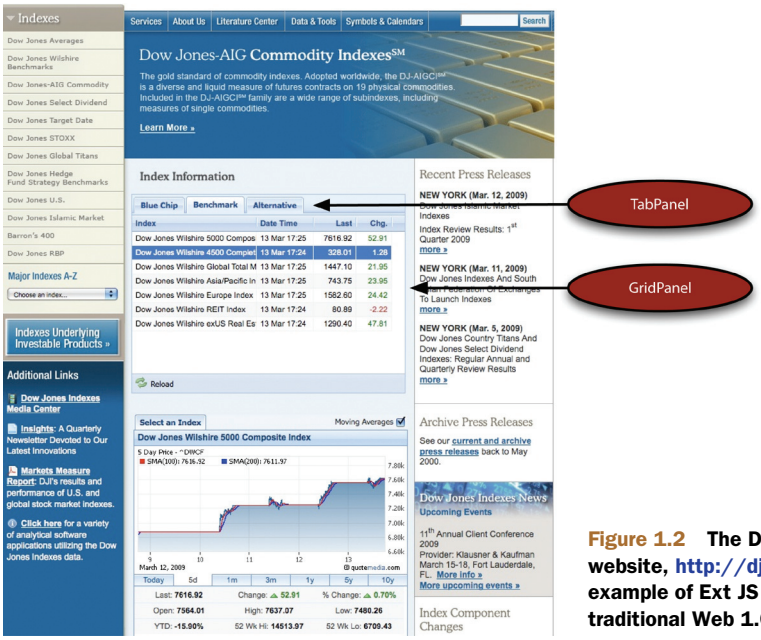

Figure 1.2 The Dow Jones Indexes website, http://djindexes.com, is one example of Ext JS embedded in a traditional Web 1.0 site.

We'll explore all of the features of the API and talk about some of the components used in this documentation tool. Figure 1.3 illustrates some of the components used in the Ext JS API Documentation application.

The API Documentation tool is chock-full of gooey GUI goodness and uses six of the most commonly used widgets, which include the `Viewport`, `TreePanel`, `Toolbar` with an embedded `TextField` and `Toolbar Buttons`, and `TabPanel`. I'm sure you're wondering what all of these are and what they do. Let's take a moment to discuss these widgets before we move on.

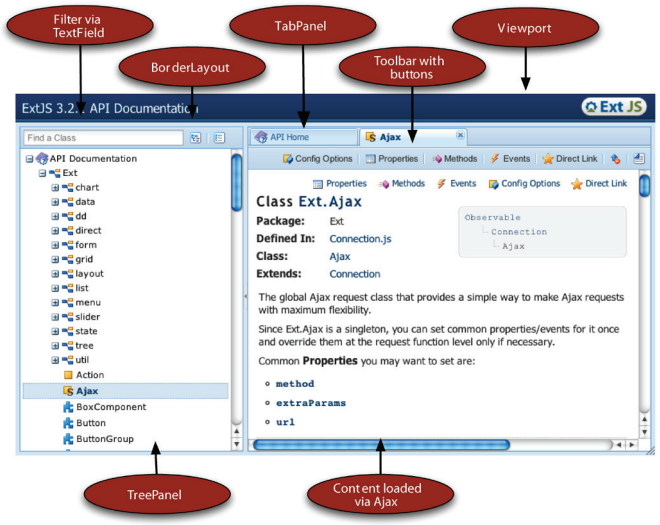

Figure 1.3 The Ext JS API Documentation contains a wealth of information and is a great resource for learning more about components and widgets. It contains most of what you need to know about the API, including constructor configuration options, methods, events, properties, component hierarchy, and more.

Looking from the outside in, the `Viewport` is a class that leverages all of the browser viewing space to provide an Ext JS–managed canvas for UI widgets. It's the foundation from which an entire Ext JS application is built upon. The `BorderLayout` layout manager is usually used, which divides the `Viewport` (or any container) into five regions. In this example three are used: North (top) for the page title, link, and `Toolbar`; West (left), which contains the `TreePanel`; and the Center (right) region, which contains the `TabPanel` to display documentation.

The `Toolbar` class provides a means to present commonly used UI components such as `Buttons` and `Menus`, but it can also contain, as in this case, any of the `Ext.form.field` subclasses. I like to think of the `Toolbar` as the common file-edit-view menus that you see in popular operating systems and desktop applications. The `TreePanel` widget displays hierarchical data visually in the form of a tree much like Windows Explorer displays your hard drive's folders. The `TabPanel` provides a means to have multiple documents or components on the canvas but allows only one to be active at a time.

Using the API is a cinch. To view a document, click the class node on the tree. This will invoke an Ajax request to fetch the documentation for the desired class. Each document for the classes is an HTML fragment (not a full HTML page). With the `TextField` in the `Toolbar`, you can easily filter out the class tree with a few strokes on the keyboard. If you're connected to the internet, you can search for API documentation in the API Home tab.

So the documentation is thorough. But what about rapid application development? Can Ext JS accelerate your development cycles?

1.1.3 Rapid development with prebuilt widgets

Ext JS can help you jump from conception to prototype because it offers many of the required UI elements already built and ready for integration. Having these UI widgets prebuilt, instead of having to engineer them, means that you save a lot of time. In many cases, the UI controls are highly customizable and can be modified to your application needs.

1.1.4 Works with Prototype, jQuery, and YUI, and inside AIR

Even though we discussed how Ext JS stands apart from other libraries such as YUI, it can easily be configured to leverage these frameworks as a base. This means if you're using these other libraries, you don't have to give them up to enjoy Ext JS UI goodness.

Although we won't cover development of Ext JS applications in Adobe AIR, it's worth noting that the framework has a complete set of utility classes to help with the integration of AIR. These utility classes include items like sound management, a video player panel, and access to the desktop clipboard. We won't go into AIR in this book, but we'll cover many of the important parts of the framework you'll need to know when developing with Ext JS in Adobe AIR.

> **NOTE** Because Adobe AIR is a development environment with features such as a sandbox, we won't be discussing how to develop applications with it. To learn more about Adobe AIR, visit http://www.adobe.com/devnet/air/.

Before we talk any more about Ext JS, we should set the playing field and discuss what types of skills are necessary to utilize the framework.

1.2 What you need to know

Although being an expert in web application development isn't required to develop with Ext JS, developers should have some core competencies before attempting to write code with the framework.

The first of these skills is a basic understanding of Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS). It's 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. Although 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 holds Ajax together, a solid foundation in JavaScript programming is suggested. Again, you need not be an expert, but you should have a good grasp of key concepts such as arrays, reference, and scope. It's a plus if you're familiar with object-oriented JavaScript fundamentals such as objects, classes, and prototypal inheritance. If you're new to JavaScript, you're in luck. JavaScript has existed since nearly the dawn of the internet. An excellent place to start is W3Schools.com, which offers a lot of free online tutorials and even has sandboxes for you to play with JavaScript online. You can visit them at http://w3schools.com/JS/.

If you're 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. To persist data, you'll need to know how to interact with a database or filesystem with your server-side language of choice.

Naturally, the range of solutions available is quite large. For this book, we won't focus on a specific language. Instead, we'll use online resources at http://extjsinaction.com, where I've done the server-side work for you. This way, all you have to focus on is learning Ext JS.

We'll begin our exploration of Ext JS with a bird's-eye view of the framework, where you'll learn about the categories of functionality.

1.3 A bird's-eye view of the framework

The Ext JS framework not only provides UI widgets but also contains a host of other features. These fall into six major areas of purpose: Core, UI components, web remoting, data services, drag and drop, and general utilities. Figure 1.4 illustrates the six areas of purpose.

Knowing what the different areas of purpose are and what they do will give you an edge when developing applications, so we'll take a moment to discuss them.

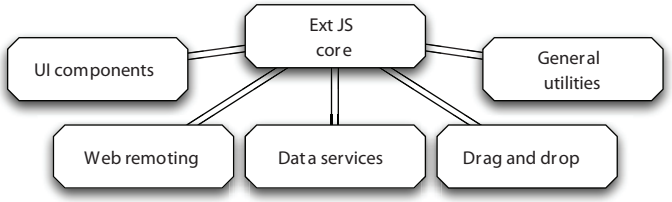

Figure 1.4 The six areas of purpose for Ext JS classes: Ext JS Core, UI components, web remoting, data services, drag and drop, and general utilities

CORE

The first feature set is the Ext JS Core, which comprises of many of the basic features such as Ajax communication, DOM manipulation, and event management. Everything else is dependent on the Core of the framework, but the Core isn't dependent on anything else.

> **Learn more about Ext Core**
>
> Ext Core is a library, which is a subset of the Ext JS base functionality and can be considered on par with jQuery, Prototype, and Scriptaculous. To learn more about Ext Core, visit http://extjs.com/products/core/.

UI COMPONENTS

The UI components contain all of the widgets that interface with the user.

WEB REMOTING

Web remoting is a means for JavaScript to (remotely) execute method calls that are defined and exposed on the server, which is commonly known as a Remote Procedure Call, or RPC. It's convenient for development environments where you'd like to expose your server-side methods to the client and not worry about all of the fuss of Ajax method management.

> **Learn more about Ext JS Direct**
>
> Because Direct is a server-side-focused product, we won't be covering it in this book. Ext JS has many online resources where you can learn about Direct, including examples for many of the popular server-side solutions. To learn more about Direct, visit http://extjs.com/products/direct/.

DATA SERVICES

The data services section takes care of all of your data needs, which include fetching, parsing, and loading information into stores. With the Ext JS data services classes, you can read Array, XML, and JSON (JavaScript Serialized Object Notation), which is a data format that's quickly becoming the standard for client-to-server communication. Stores typically feed UI components.

DRAG AND DROP

Drag and drop is like a mini framework inside Ext JS, where you can apply drag-and-drop capabilities to an Ext JS component or any HTML element on the page. It includes all of the necessary members to manage the entire gamut of all drag-and-drop operations. Drag and drop is a complex topic. We'll spend the entirety of chapters 13 and 14 on this subject alone.

UTILITIES

The utilities section comprises cool utility classes that help you perform some of your routine tasks easier. An example would be `Ext.util.Format`, which allows you to format or transform data easily. Another neat utility is the CSS singleton, which allows you to create, update, swap, and remove stylesheets as well as request the browser to update its rule cache.

Now that you have a general understanding of the framework's major areas of functionality, let's take some time to look at some of the more commonly used UI widgets that Ext JS has to offer.

1.3.1 Containers and layouts at a glance

Even though we'll cover these topics in detail in chapter 3, we should spend a little time here talking about containers and layouts. The terms *container* and *layout* are used extensively throughout this book, and I want to make sure you have at least a basic understanding of them before we continue. Afterwards, we'll begin our exploration of visual components of the UI library.

CONTAINERS

Containers are widgets that can manage one or more child items. A child item is generally any widget or component that's managed by a container or parent, thus the parent-child paradigm. You've already seen this in action in the API. The `TabPanel` is a container that manages one or more child items, which can be accessed via tabs. Please remember this term, because you'll be using it a lot when you start to learn more about how to use the UI portion of the framework.

LAYOUTS

Layouts are implemented by a container to visually organize the child items in the container's content body. Ext JS has 12 layouts from which to choose, which we'll go into in great detail in chapter 5 and shows the ins and outs of each layout. Now that you have a high level of understanding of containers and layouts, let's look at some containers in action.

In the figure 1.5, you see two subclasses of `Container`, `Panel` and `Window`, each engaged in parent-child relationships.

The `Panel` (left) and `Window` (right) in figure 1.5 each manage two child items. Child Panel 1 of each parent container *contains* HTML, whereas the children with the title Child Panel 2 manage one child panel each using the simple `ContainerLayout`, which is the base class for all other layouts. This parent-child relationship is the crux

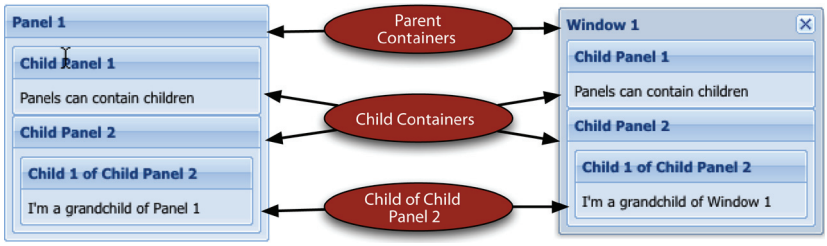

Figure 1.5 Here, you see two parent `Containers`, `Panel` (left) and `Window` (right), managing child items, which include nested children.

of all of the UI management of Ext JS and will be reinforced and referenced repeatedly throughout this book.

You learned that `Containers` manage child items and use layouts to visually organize them. Now that you have these important concepts down, we'll move on to see and discuss other `Containers` in action.

1.3.2 Other Containers in action

You saw `Panel` and `Window` being used when you learned about `Containers`. Figure 1.6 shows some other commonly used subclasses of `Container`.

In figure 1.6, you see the `FormPanel`, `TabPanel`, `FieldSet`, and `QuickTip` widgets. The `FormPanel` works with the `BasicForm` class to wrap fields and other child items with a `form` element.

Looking at this from a parent-child perspective, the `FormPanel` is being used to manage three child items: two instances of `FieldSet` and one `instance` of `TabPanel`.

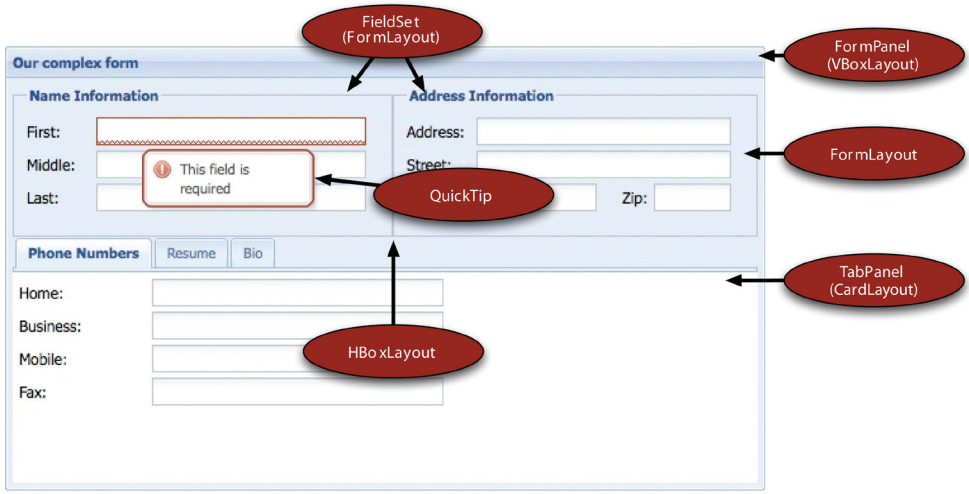

Figure 1.6 Commonly used subclasses of `Container`—`FormPanel`, `TabPanel`, `FieldSet`, and `QuickTip`—and the layouts used to compose this UI `Panel`. We'll build this in chapter 6, where you learn about forms.

FieldSets are generally used to display fields in a form much like a typical FieldSet tag does in HTML. These two FieldSets are managing child items, which are text fields. The TabPanel here is managing three direct children (tabs); in the first tab (Phone Numbers) it's managing many children, which are text fields. The QuickTip, which is used to display helpful text when the mouse is hovering over an element, is being displayed, but it isn't a child of any Ext JS component.

We'll spend some time building this complex UI in chapter 6, where you'll learn more about FormPanels. For now, let's move on to see what data presentation widgets the framework has to offer.

1.3.3 Grids, DataView, and ListView

You've already learned that the Data Services portion of the framework is responsible for the loading and parsing of data. The main consumers of the data that the data store manages are the GridPanel and the DataView and its subclass, the ListView. Figure 1.7 is a screenshot of the Ext JS GridPanel in action.

The GridPanel is a subclass of Panel and presents data in a table-like format, but its functionality extends far beyond that of a traditional table, offering sortable, resizable, and movable column headers and selection models such as RowSelectionModel and CellSelectionModel. You can customize its look and feel as you desire and couple it with a PagingToolbar to allow large datasets to be segmented and displayed in pages. It also has subclasses such as the EditorGridPanel, which allows you to create a grid where users can edit data on the grid itself, leveraging any of the Ext JS form data input widgets.

The grid is great for displaying data but is somewhat computationally expensive because of the many DOM elements that each row contains. To combat this, you can

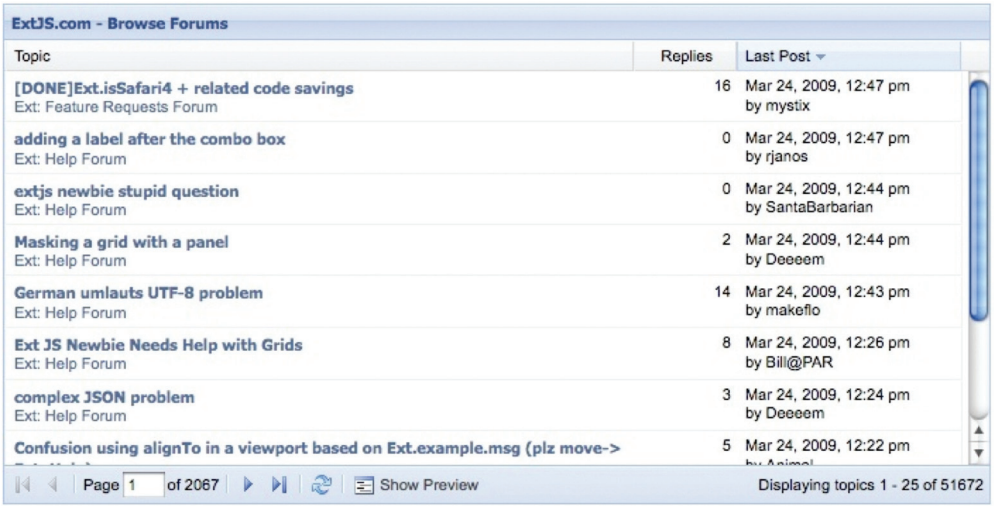

Figure 1.7 The GridPanel as seen in the Buffered Grid example in the Ext JS SDK

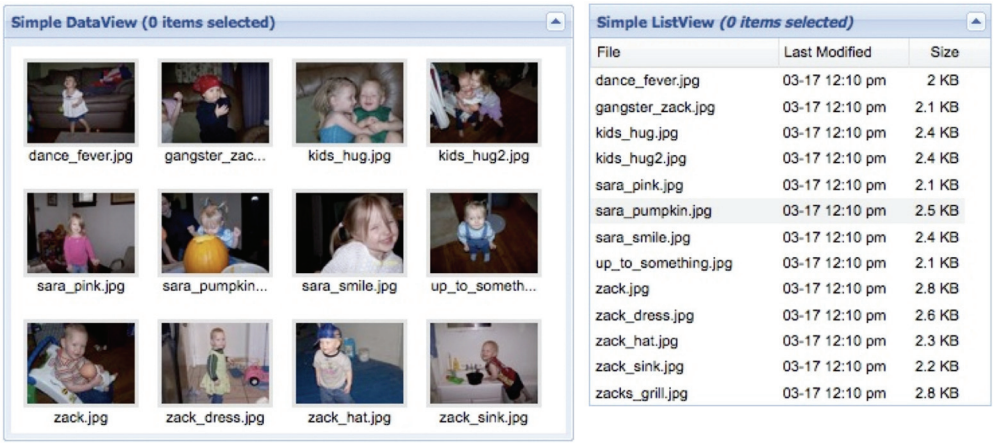

Figure 1.8 The `DataView` (left) and `ListView` (right) as shown in the Ext JS SDK examples

couple the `GridPanel` with a `PagingToolbar` or use one of the lighter-weight widgets to display data from a store, including the `DataView` and its subclass, the `ListView`, as shown in figure 1.8.

The `DataView` class consumes data from a store, paints it onscreen using a `Template`, and provides a simple selection model. An Ext JS `Template` is a DOM utility that allows you to create a *template*, with placeholders for data elements, which can be filled in by individual records in a store and stamped out on the DOM. In figure 1.8, the `DataView` (left) is displaying data from a store, which includes references to images. It uses a predefined template, which contains image tags, where the individual records are leveraged to fill in the location of the images. The `Template` then stamps out an image tag for each individual record, resulting in a nice collage of photos. The `DataView` can be used to display anything in a data store.

The `ListView`, as pictured in figure 1.8 (right), is displaying data from a store in a grid-like fashion but is a subclass of `DataView`. It's a great way to display data in a tabular format without the weight of the `GridPanel` if you're not looking to use some of the `GridPanel` features like sortable and resizable columns.

The `GridPanel` and `DataView` are essential tools for painting data onscreen, but they do have one major limitation. They can only show lists of records. They can't display hierarchical data. This is where the `TreePanel` fills the gap.

1.3.4 Make like a TreePanel and leaf

The `TreePanel` widget is an exception to the list of UI widgets that consume data, in that it doesn't consume data from a data store. Instead, it consumes hierarchical data via the usage of the `data.Tree` class. Figure 1.9 shows an example of an Ext JS `TreePanel` widget.

In figure 1.9, the `TreePanel` is being used to display the parent-child data inside the directory of an installation of the framework. The `TreePanel` can leverage a

`TreeLoader` to fetch data remotely via Ajax or can be configured to use data stored on the browser. It can also be configured to use drag and drop and has its own selection model.

You already saw `TextFields` in a form when we discussed `containers` a short while ago. Next, we'll look at some of the other input fields that the framework has to offer.

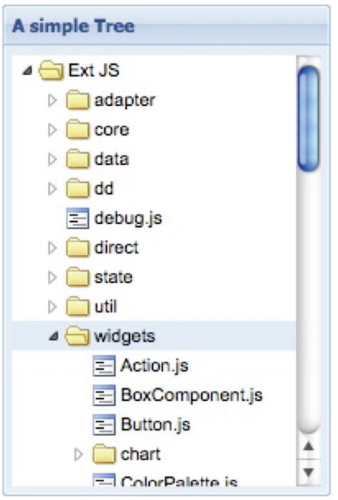

Figure 1.9 An Ext JS Tree, which is an example from the Ext JS SDK

1.3.5 Form input fields

Ext JS has a palette of eight input fields. They range from simple `TextFields`, as you've already seen, to complex fields such as the `ComboBox` and the `HtmlEditor`. Figure 1.10 is an illustration of some of the available Ext JS form field widgets that come out of the box.

As you can see in figure 1.10, some of the form input fields look like stylized versions of their native HTML counterparts. The similarities end here, however. With the Ext JS form fields, there's much more than meets the eye!

Each of the Ext JS fields (minus the `HtmlEditor`) includes a suite of utilities to perform actions like get and set values, mark the field as invalid, reset, and perform validations against the field. You can apply custom validation to the field via `regex` or custom validation methods, allowing you complete control over the data being

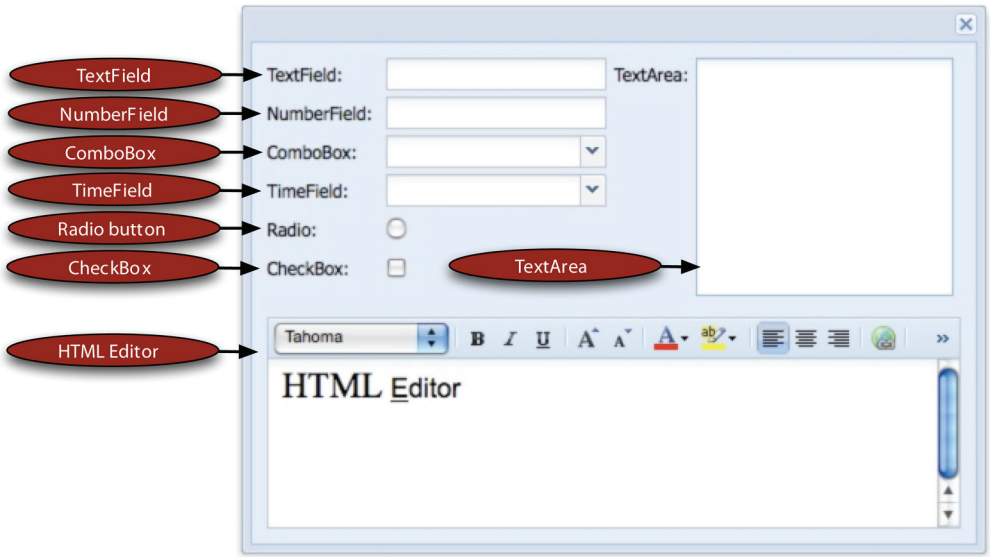

Figure 1.10 All of the out-of-the-box form elements displayed in an encapsulating `Window`

entered into the form. The fields can validate data as it's being entered, providing live feedback to the user.

TEXTFIELD AND TEXTAREA

The `TextField` and `TextArea` classes can be considered extensions of their generic HTML counterparts. `NumberField`, however, is a subclass of the `TextField` and is a convenience class, which utilizes regular expressions to ensure users can enter only numbers. With `NumberField`, you can configure decimal precision as well as specify the range of the value entered. The `ComboBox` and `TimeField` classes require a little extra time relative to the other fields, so we'll skip these two for now and jump back to them in a bit.

RADIO AND CHECKBOX

Like `TextField`, the `Radio` and `Checkbox` input fields are extensions of the plain old `Radio` and `Checkbox` but include all of the Ext JS `Element` management goodness and have convenience classes to assist with the creation of `Checkbox` and `RadioGroups` with automatic layout management. Figure 1.11 shows a small sample of how the Ext JS `Checkbox` and `RadioGroup` classes can be configured with complex layouts.

See all of the Radio and Checkbox examples

To see the entire set of examples, visit http://extjs.com/deploy/dev/examples/form/check-radio.html.

HTMLEDITOR

The HTML editor is WYSIWYG, like the `TextArea` on steroids. The `HtmlEditor` leverages existing browser HTML editing capabilities and can be considered somewhat of a black sheep when it comes to fields. Because of its inherent complexities (using `IFrames` and the like), it doesn't have a lot of the abilities like validation and can't be marked as invalid. There's much more to discuss about this field, which we're going to save for chapter 6. But for now, let's circle back to the `ComboBox` and its subclass, the `TimeField`.

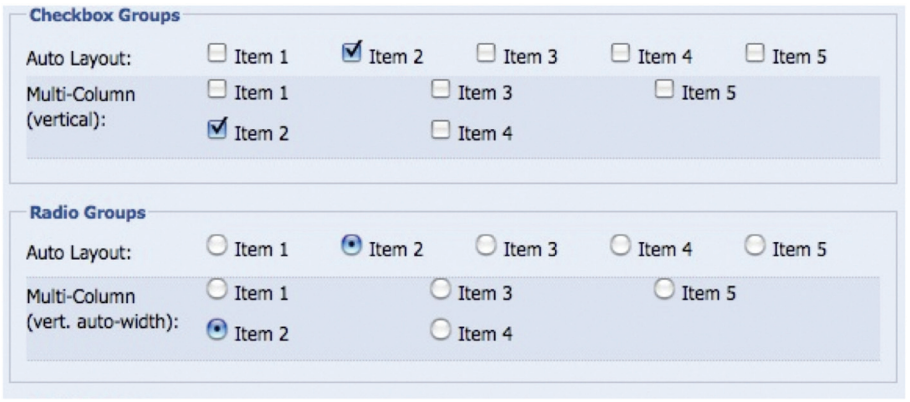

Figure 1.11 An example of the `Checkbox` and `RadioGroup` convenience classes in action with automatic layouts.

Figure 1.12 A custom `ComboBox`, which includes an integrated paging toolbar, as shown in the downloadable Ext JS examples

COMBOBOX AND TIMEFIELD

The `ComboBox` is easily the most complex and configurable form input field. It can mimic traditional option drop-down boxes or can be configured to use remote datasets via the data store. It can be configured to autocomplete text, known as *type-ahead*, entered by the user and perform remote or local filtering of data. It can also be configured to use your own instance of an Ext JS `Template` to display a custom list in the drop-down area, known as the `ListBox`. Figure 1.12 is an example of a custom `ComboBox` in action.

In figure 1.12, a custom combo box is being leveraged to search the Ext JS forums. The `ComboBox` here shows information like the post title, date, author, and a snippet of the post in the list box. Because some of the dataset ranges are so large, it's configured to use a paging toolbar, allowing users to page through the resulting data. Because the `ComboBox` is so configurable, we could also include image references to the resulting dataset, which can be applied to the resulting rendered data.

Here we are, on the last stop of our UI tour. Now we'll take a peek at some of the other UI components that work anywhere.

1.3.6 Other widgets

A bunch of UI controls stand out, which aren't major components but play supporting roles in a grander scheme of a UI. Look at the illustration in figure 1.13, and follow along as we discuss what these items are and what they do.

Here's a breakdown of the various widgets.

TOOLBAR

The `Toolbar` is a widget that allows you to place in it just about any widget that fits. Generally, developers place menus and buttons in it. When we discussed the custom `ComboBox`, you saw the `Toolbar`'s subclass, the `PagingToolbar`. `Panels` and just about any subclass thereof can use these toolbars on the top or bottom of their content body. The `Button` widget is a stylized version of the generic HTML button, which can include icons as well as text.

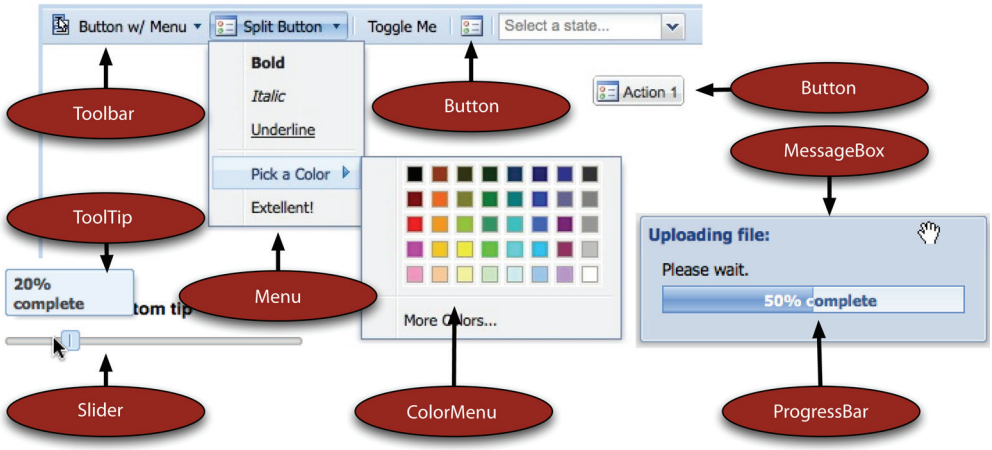

Figure 1.13 **Miscellaneous UI widgets and controls**

MENU

`Menus` can be displayed by clicking a button on the toolbar or shown on demand at any X and Y coordinates on screen. Although they typically contain menu items, such as the items shown and the color menu item, they can contain widgets such as `ComboBoxes`.

MESSAGEBOX

The `MessageBox` is a utility class, which allows you to easily provide feedback to the user without having to craft up an instance of `Ext.Window`. In this illustration, it's using an animated `ProgressBar` to display the status of an upload to the user.

SLIDER

The `Slider` is a widget that leverages drag and drop to allow users to change a value by repositioning the knob. `Sliders` can be styled with images and CSS to create your own custom look. The movement of the knob can be restricted so it moves only in increments. In this example, the slider has a `ToolTip` above the knob, displaying the value of the slider as the user moves it. In addition to the default horizontal orientation, `Sliders` can be configured as vertical.

You've learned how Ext JS can help you get the job done through a large palette of widgets. You've learned that you could elect to use Ext JS to build an application without touching an ounce of HTML or integrate it with existing sites. You also got a top-down view of the framework, which included a UI tour. All of the material discussed thus far existed for Ext JS 2.0. Let's take a moment to discuss what's new in Ext JS 3.0.

1.4 New Ext JS 3.0 goodies

Ext JS 2.0 introduced some radical changes, which made upgrading from 1.0 rather difficult. This was mainly because of the introduction to the more modern layout manager and a new, robust component hierarchy, which broke most of the user-developed Ext JS 1.x code. Thankfully, due to the great engineering of Ext JS 2.0, the migration from Ext JS 2.0 to 3.0 is much easier. Although the additions to Ext JS 3.0

aren't as drastic, there's excitement about this latest release, and it's certainly worthwhile to discuss some of the additions.

1.4.1 Ext JS does remoting with Direct

Web remoting is a means for JavaScript to easily execute method calls that are defined on the server side. It's convenient for development environments where you'd like to expose your server-side methods to the client and not have to worry about all of the muck with Ajax connection handling. `Ext.Direct` takes care of this for us by managing Ajax requests and acts as a bridge between the client-side JavaScript and any server-side language.

This functionality has great advantages, which include method management in a single location as well as unification of methods. Having this technology inside the framework will ensure consistency across the consumers, such as the data classes. Speaking of which, let's look at how the addition of `Ext.Direct` brings new classes to the data classes.

1.4.2 Data class

The `Ext.data` class is the nerve center for just about all data handling in the framework. The data classes manage every aspect of data management, including fetching, reading, and parsing data to create records, which are loaded into a store. With the addition of `Direct`, Ext JS has added additional convenience `Data` classes, `DirectProxy` and `DirectStore`, to facilitate ease of integration with your web remoting needs.

Now that we've reviewed some of the behind-the-scenes changes and additions to the framework, take a gander at some of the UI widgets that have been added.

1.4.3 Meet the new layouts

A total of six new layouts make their way into the Ext JS 3.0 framework, including `AutoLayout`, `MenuLayout`, `ToolbarLayout`, `BoxLayout`, `VBoxLayout`, and `HBoxLayout`. `MenuLayout` is an improvement on the way menu items were organized in the 2.0 version of the framework. Similarly, `ToolbarLayout` adds important features to the `Toolbar`, like overflow management, as illustrated in figure 1.14. Neither of these two layouts is designed to be implemented by their intended widgets, and neither is for end-developer use.

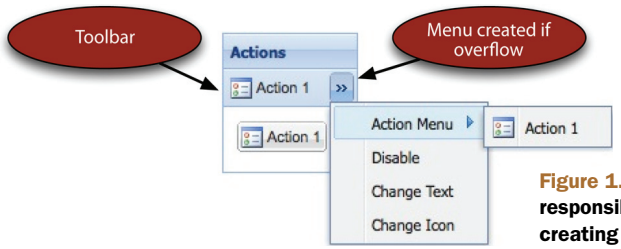

Figure 1.14 The new `ToolbarLayout` is responsible for detecting the toolbar's size and creating menu stubs if menu items overflow.

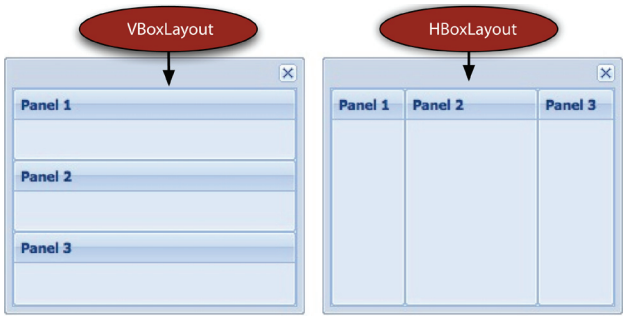

Figure 1.15 An example of the `VBox` and `HBox` layouts in action

As shown in figure 1.14, `ToolbarLayout` will detect the overflow of toolbar items in a toolbar and will automatically create a menu that lists and contains the rest of the items. The changes to `MenuLayout` help support this effort.

The `BoxLayout` class is an abstract class that's meant to provide base functionality for the `VBoxLayout` and `HBoxLayout` classes and is not designed for end-developer use. The `VBoxLayout` and `HBoxLayout` are great additions to the list of end-developer-usable layouts. The `HBoxLayout` allows you to divide a container's content body into horizontal slices, whereas the `VBoxLayout` does the same except vertically, as illustrated in figure 1.15.

Many experienced Ext JS developers might think that the `HBoxLayout` looks like the `ColumnLayout` in practice. Although it does provide similar functionality, it extends way beyond the `ColumnLayout`'s capabilities, where it will vertically and horizontally stretch children based on weights, which is known as `flex`. In contrast to the `ColumnLayout`, however, child items will never wrap around inside the `Container`'s content body. These two layouts usher in a whole new era of layout capabilities within the framework.

In addition to the layout changes, the `ColumnModel`, a supporting class for the `GridPanel`, has undergone some fundamental changes. Let's look at some of these changes and see why they're going to help us in our development efforts.

1.4.4 Grid ColumnModel enhancements

The grid `ColumnModel` is a class that models how the columns are organized, sized, and displayed for the `GridPanel` widget. Prior to Ext JS 3.0, individual columns were generally configured as a list of configuration objects in an array, which is consumed by `ColumnModel`.

For each column in the `ColumnModel`, you could enhance or modify the way the data is displayed by creating a custom renderer, which is a method that's called for each data point for

Configuration objects

Many of the Ext JS constructors and some methods accept parameters listed in a configuration object. These configuration objects are nothing more than plain objects that contain a set of key-value pair parameters that will influence how a widget is rendered onscreen or how a method behaves.

that column and returns the desired formatted data or HTML. This means that if you wanted, let's say, a date to be formatted or displayed a certain way, you had to configure it, which many people found themselves doing a lot. In this release, the `ColumnModel` changed somewhat to make our jobs that much easier.

The individual `Column` has been abstracted from the `ColumnModel` and an entirely new class created called the `grid.Column`. From here, many convenience `Column` subclasses have been created, which include `NumberColumn`, `BooleanColumn`, `TemplateColumn`, and `DateColumn`, each of which allows you to display your data as you desire. To display formatted dates, you could use the `DateColumn` and specify a format in which the dates are to be displayed. The `TemplateColumn` is another welcome change because it allows you to leverage `XTemplates` and display them in a `GridPanel`, which are convenience methods to create and stamp out HTML fragments based on data. To use any of these `Column` subclasses, no custom renderers are required, though you can use them if you wish.

Many applications require data to be displayed in tabular format. Although the `GridPanel` is a great solution, it's computationally expensive for generic data displays that require little or no user interaction. This is where `ListView`, an extension of `DataView`, comes to the rescue.

1.4.5 *ListView, like GridPanel on a diet*

With this new addition to the framework, you can now display more data in a grid-like format without sacrificing performance. Figure 1.16 shows the `ListView` in action. Although it looks similar to the `GridPanel`, in order to achieve better performance, we sacrifice features such as drag-and-drop column reordering as well as keyboard navigation. This is mainly because the `ListView` doesn't have any of the elaborate, feature-rich supporting classes, such as the `ColumnModel` we discussed a moment ago.

Using `ListView` to display your data will ensure that you have faster response from DOM manipulation, but remember that it doesn't have all of the features of the `GridPanel`. Choosing which one to use will depend on your application requirements.

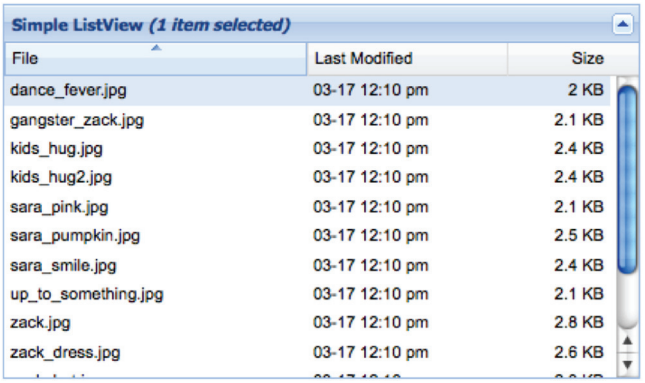

Figure 1.16 The new `Ext.ListView` class, which is like a lightweight `DataGrid`

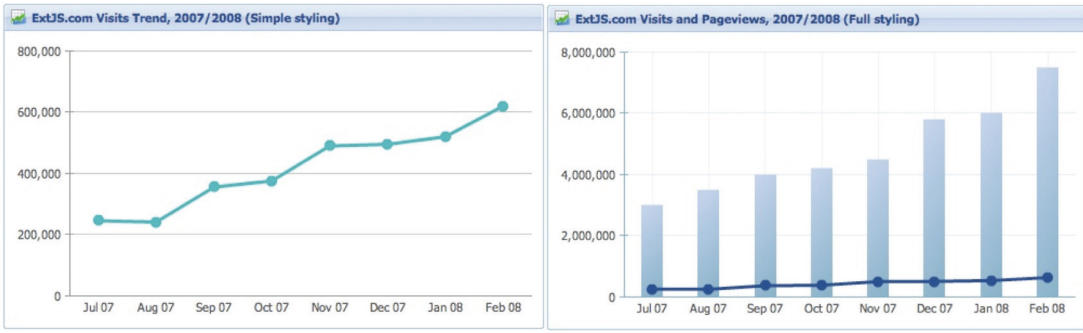

Figure 1.17 These charts now bring rich graphical views of trend data to the framework. It's important to note that this new widget requires Flash.

Ext JS has always been excellent at displaying textual data on screen, but it lacked a graphical means of data representation. Let's take a quick look at how this has changed with Ext JS 3.0.

1.4.6 Charts come to Ext JS

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

Use of these charts, however, requires Adobe Flash to be installed for the browser you're using, which you can download at http://get.adobe.com/flashplayer/. In addition to the `Line` and `Column` charts shown in figure 1.17, the framework has `Bar`, `Pie`, and `Cartesian` charts available for your data visualization needs.

You now have just about everything you need to lay down some code. Before you start on your path to becoming Ext JS Jedis, you must first download and set up the framework, and we'll have a discussion about development.

1.5 Downloading and configuring

Even though downloading Ext JS is a simple process, configuring a page to include Ext JS isn't as simple as referencing a single file in HTML. In addition to configuration, you'll learn about the folder hierarchy and what folders are and what they do.

The first thing you need to do is get the source code. Visit the following link: http://extjs.com/products/extjs/download.php.

The downloaded file will be the SDK in a zip file, which weighs in at over 6 MB in size. I'll explain why this file is so large in a bit. Now, extract the file in a place where you serve JavaScript. In order to leverage Ajax, you're going to need a web server. I typically use Apache configured locally on my computer, which is free and cross-platform, but IIS for Windows will do. Let's peek at what you just extracted.

1.5.1 Looking at the SDK contents

If you're like me, you probably checked the size of the files extracted from the downloaded SDK zip file. If your jaw dropped, feel free to pick it back up. Yes, over 30 MB is rather large for a JavaScript framework. Pay no attention to the size for now; figure 1.18 shows what was extracted.

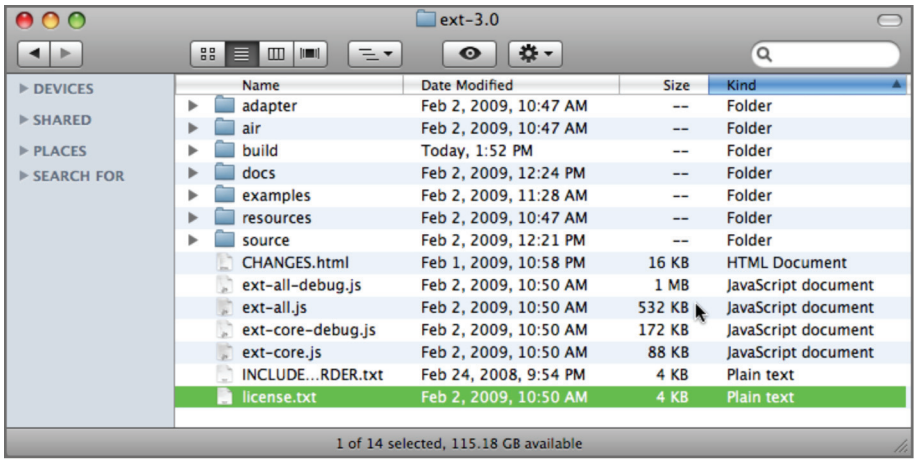

Figure 1.18 A view of the Ext JS SDK contents

Looking at the contents of the SDK, you see a lot of stuff. The reason there are so many folders and files is that the downloadable package contains a few copies of the entire code base and CSS. It's this way because you get the freedom to build or use Ext JS any way you see fit. Table 1.1 explains what each of the folders is and what it does.

Table 1.1 The contents of the Ext JS SDK

Folder	What it does
adapter	Contains the ext-base.js, which is the base Ext JS library, which is used for an all-Ext JS setup. It also contains necessary adapters and supported versions of Prototype, jQuery, or YUI libraries if you want to use any of those as a base.
air	Contains all of the required libraries to integrate Ext JS with Adobe AIR.
build	Has each of the files that compose the Ext JS framework with all of the unnecessary whitespace removed. It's a deflated version of the source directory.
docs	Holds the full API documentation.
examples	Holds all of the example source code, which is great source to learn by example (no pun intended).
resources	Contains all of the necessary images and CSS required to use the UI widgets. It contains all of the CSS files broken down by widget and an ext-all.css, which is a concatenation of all the framework's CSS.

Table 1.1 The contents of the Ext JS SDK *(continued)*

Folder	What it does
src	Holds the entire framework, including all of the comments.
ext-all.js	This is a minified version of the framework, which is intended for production applications.
ext-core.js	If you wanted to use the core library, which means none of the UI controls, you'd set up your page with ext-core.js.
*debug.js	Anything with *debug* in the name means that the comments are stripped to reduce file space, but the necessary indentation remains intact. When we develop, we're going to use ext-all-debug.js.

Although there are quite a few files and folders in the distribution, you need only a few of them to get the framework running in your browser. Now is a good time to talk about how to set up Ext JS for use.

1.5.2 Setting up Ext JS for the first time

In order to get Ext JS running in your browser, you need to include at least two required JavaScript files and at least one CSS file:

```
<link rel="stylesheet" type="text/css"
     href="extjs/resources/css/ext-all.css" />

 <script type="text/javascript" src="extjs/adapter/ext/ext-base-debug.js">
 </script>

 <script type="text/javascript" src="extjs/ext-all-debug.js">
 </script>
```

Here we're linking the three core files for an all-Ext JS configuration. The first thing we do is link to the ext-all.css file, which is all of the CSS for the framework in one file. Next, we include ext-base-debug.js, which is the underlying base of the framework. Last, we include ext-all-debug.js, which we'll use for development. When setting up your initial page, be sure to replace extjs in your path with wherever you plan to reference the framework on your development web server.

What if you want to use any of the other base frameworks? How do you include those?

1.5.3 Configuring Ext JS for use with others

In order for Ext JS to work with the previously mentioned frameworks, an *adapter* must be loaded after the external base framework. The adapter maps ext-base methods to the external library of choice, which is crucial. You can use the following patterns to use any of the other three base frameworks in addition to Ext JS:

First up, the Prototype library:

```
<link rel="stylesheet" type="text/css"
     href="extjs/resources/css/ext-all.css" />
```

```
<script type="text/javascript"
    src="extjs/adapter/prototype/prototype.js">
</script>

<script type="text/javascript"
    src="extjs/adapter/prototype/scriptaculous.js?load=effects.js">
</script>

<script type="text/javascript"
    src="extjs/adapter/prototype/ext-prototype-adapter-debug.js">
</script>

<script type="text/javascript"
     src="extjs/ext-all-debug.js"></script>
```

As you can see, this is like the generic Ext JS setup with two additional JS files. The Prototype and Scriptaculous libraries take the place of ext-base, and ext-prototype-adapter.js maps the external library methods to Ext. Note that we're still loading ext-all-debug.js. We'll continue to do so for the other two cases.

Next is jQuery:

```
<link rel="stylesheet" type="text/css"
    href="extjs/resources/css/ext-all.css" />

<script type="text/javascript"
    src="extjs/adapter/jQuery/jQuery.js">
</script>

<script type="text/javascript"
    src="extjs/adapter/jQuery/ext-jquery-adapter-debug.js">
</script>

<script type="text/javascript"
     src="extjs/ext-all-debug.js"></script>
```

Configuring jQuery is similar to the Prototype setup. The YUI configuration will be similar, the difference being that we're loading different base library and adapter files.

Lastly, YUI:

```
<link rel="stylesheet" type="text/css"
    href="extjs/resources/css/ext-all.css" />

<script type="text/javascript"
    src="extjs/adapter/yui/yui-utilities.js">
</script>

<script type="text/javascript"
    src="extjs/adapter/yui/ext-yui-adapter-debug.js">
</script>

<script type="text/javascript"
     src="extjs/ext-all-debug.js"></script>
```

And there you have it, the recipes for an Ext-all setup and the other three supported base JS libraries. Moving forward, we'll be using the Ext-all configuration, but you're free to use whichever base library you wish. Before we move on to coding, we need to talk about one final crucial step to setting up Ext, and that's configuring the reference for s.gif.

Place `BLANK_IMAGE_URL` configuration wisely

It's recommended that you set this parameter immediately after the inclusion of the Ext JS files or immediately before your application code is parsed. I'll show you an example of where to place it when we take Ext JS for a test drive.

1.5.4 Configuring BLANK_IMAGE_URL

The configuration of the `Ext.BLANK_IMAGE_URL` is one of those steps that developers often overlook and may lead to issues with the way the UI renders for your application. The `BLANK_IMAGE_URL` property specifies a location for the 1x1–pixel clear s.gif graphic (known as a spacer), which is an essential piece of the UI portion of the framework and is used to create items like icons. Out of the box, `BLANK_IMAGE_URL` points to http://extjs.com/s.gif. For most users, that's okay, but if you're in an area where extjs.com isn't accessible, this will become an issue. This also becomes a problem if you're using SSL, where s.gif is being requested via HTTP instead of HTTPS, which will invoke security warnings in browsers. In order to prevent these issues, set `Ext.BLANK_IMAGE_URL` to s.gif locally on your web server, like this:

```
Ext.BLANK_IMAGE_URL = 'extjs/resources/images/default/s.gif';
```

If you're like me, after all of this discussion, you're probably itching to start using Ext JS. So, what are you waiting for? Go ahead and dive in.

1.6 Take it for a test drive

For this exercise, you'll create an Ext JS `Window`, and you'll use Ajax to request an HTML file for presentation in the content body of the `Window`. You'll start by creating the main HTML file, from which you'll source all of your JavaScript files.

Listing 1.1 Creating our helloWorld.html

```
<link rel="stylesheet" type="text/css"
     href="/extjs/resources/css/ext-all.css" />

<script type="text/javascript"
  src="/extjs/adapter/ext/ext-base-debug.js">
</script>

<script type="text/javascript"
     src="/extjs/ext-all-debug.js"></script>

<script type="text/javascript">
     Ext.BLANK_IMAGE_URL = '/extjs/resources/images/default/s.gif';
</script>

<script type="text/javascript" src='helloWorld.js'>
</script>
```

1 Include ext-all.css

2 Load ext-base.js and ext-all-debug.js

3 Configure Ext. BLANK_IMAGE_URL

4 The soon-to-be-created helloWorld.js

Listing 1.1 includes the HTML markup for a typical Ext-only setup, which includes the concatenated CSS file, ext-all-css (1) and the two required JavaScript files, ext-base.js

and ext-all-debug.js (2). Next, you create a JavaScript block (3), where you set the ever-important `Ext.BLANK_IMAGE_URL` property. Last, you include our soon-to-be-created helloWorld.js file (4).

If you haven't noticed it, we're using /extjs as the absolute path to our framework code. Be sure to change it if your path is different. Next, you create our helloWorld.js file, which will contain your main JavaScript code.

Listing 1.2 Creating helloWorld.js

```
function buildWindow() {
   var win =  new Ext.Window({
      id       : 'myWindow',
      title    : 'My first Ext JS  Window',
      width    : 300,
      height   : 150,
      layout   : 'fit',
      autoLoad : {
         url     : 'sayHi.html',
         scripts : true
      }
   });
   win.show();
}

Ext.onReady(buildWindow);
```

(1) A new instance of Ext.Window

(2) An autoLoad configuration object

(3) Calling upon our window to show()

(4) Passing buildWindow to Ext.onReady

In listing 1.2, you create the function `buildWindow`, which will be passed to `Ext.onReady` for later execution. Inside this `buildWindow`, you create a new instance of `Ext.Window` and set up a reference to it called `win` (1). You pass a single configuration object to `Ext.Window`, which has all of the properties required to configure the instance you're instantiating.

In the configuration object, you specify an `id` of `'myWindow'`, which will be used in the future to look up the window using the `Ext.getCmp` convenience method. You then specify a title for the window, which will appear as blue text on the topmost portion of the window, known as the title bar. Next, you specify the height and width of the window. You then go on to set the `layout` as `'fit'`, which ensures that whatever child item is managed by your window is stretched to the dimensions of its `Content-Body`. You then move on to specify an `autoLoad` configuration object (2), which will instruct the window to automatically fetch an HTML fragment (specified via the `url` property) and execute a JavaScript if found (specified via `scripts : true`).

This ends the configuration object for your instance of `Ext.Window`. Next, you call on `win.show` (3), which renders your window. This is where you find the conclusion of the `buildWindow` method. The last thing you do is call `Ext.onReady` (4) and

HTML fragments

An HTML fragment is HTML that isn't enclosed by head and body tags and isn't considered a full page. Ext JS loads fragments because only one `HEAD` and `BODY` tag can exist in a page.

pass your method, `buildWindow`, as a reference. This ensures that `buildWindow` is executed at the right time, which is when the DOM is fully built and before any images are fetched. Let's see how your window renders. Go ahead and request helloWorld.html in your browser. If you coded everything properly, you'll see a window similar to the one in figure 1.19, with a spinning icon next to the "Loading…" text, known as a loading indicator.

Figure 1.19 Our first Ext JS window attempting to load content via Ajax

Why do you see this message? Because you haven't created sayHi.html, which you referenced in the `url` property of the `autoLoad` configuration object. Essentially, you instructed Ext JS to load something that wasn't on the web server. Next, you'll construct sayHi.html, where you'll create an HTML fragment, which will include some JavaScript.

Listing 1.3 Creating sayHi.html

```
<div>Hello from the <b>world</b> of Ajax!</div>

<script type='text/javascript'>
  function highlightWindow() {
    var win     = Ext.getCmp('myWindow');
    var winBody = win.body;
    winBody.highlight();
  }

  highlightWindow.defer(1000);
</script>
```

1 "Hello world" DIV tag

2 Highlight body of window

3 Delay execution by one second

In listing 1.3, you create an HTML fragment file, sayHi.html. It contains a div 1 from which we have our "Hello world" message. After that, you have a `script` tag with some JavaScript, which will get executed after this fragment is loaded by the browser. In our code, we create a new function called `higlightWindow` 2, which will be executed after a delay of one second. Inside that function, you perform a `highlight` effect on the content body of the window. The execution of `highlightWindow` is delayed by one second 3. Here's how this method works.

You start by creating a reference to the `Window` you created in our helloWorld.js file by using a utility method called `Ext.getCmp`, which looks up an Ext JS component by `id`. When you created your window, you assigned it an `id` of `'myWindow'`, which is what you're passing to `Ext.getCmp`. This works because all components (widgets) are registered with the `ComponentMgr` upon instantiation. `Ext.getCmp` is a way to retrieve a reference by `id` from any context within your application.

After you get the reference of your `Window`, you create a reference, `winBody`, to its content body via the `body` property. You then call its highlight method, which will

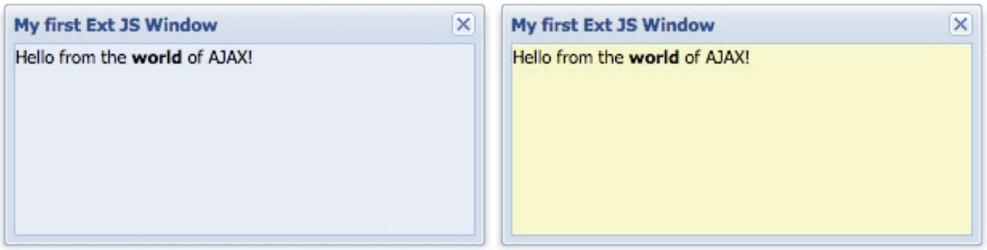

Figure 1.20 Our Ext JS `Window` loading the our HTML fragment (left) and the highlight effect performed on the `Window`'s content body (right)

perform the highlight (fade from yellow to white) operation on the element. This is where you conclude the `highlightWindow` method.

The last thing you do in this JavaScript block is call `highlightWindow.defer` and pass a value of `1000`, which defers the execution of `highlightWindow` by one thousand milliseconds (or one second).

If you've never heard of `defer` in the JavaScript language, that's because you're using an Ext-introduced method. Ext JS leverages JavaScript's extensibility to add convenience methods to important core language classes, such as `Array`, `Date`, `Function`, `Number`, and `String`. This means every instance of any of those classes has the new convenience methods. In this case, you're using `defer`, which is an extension of `Function`. If you're an old-timer, you're probably asking, "Why not use `setTimeout`?" The first reason is because of ease of use. Call `.defer` on any method and pass the length of time to defer its operation. That's it. Another reason to use it is because it allows us to control the scope from which the deferred method is being executed and pass custom parameters, which `setTimeout` lacks.

You then end your HTML fragment, which can now be fetched by your `Window`. Refresh helloWorld.html and you should see something like figure 1.20.

If you did everything correctly, your results should be exactly like those shown in figure 1.20, where the content body is being populated with the HTML fragment (left), and exactly one second later, the content body of the window highlights yellow (right). Pretty cool, huh? I suggest that you take some time to modify the example and use the API to do things like changing the color of the highlight effect. Here's a hint: Look under Ext JS > Fx for the list of effects and their parameters.

1.7 Summary

In this introduction to Ext JS, you learned how it can be used to build robust web applications or integrated into existing websites. You also learned how it measures up against other popular frameworks on the market and that it's the only UI-based framework to contain UI-centric support classes such as the `Component`, `Container`, and `Layout` models. Remember that Ext JS can ride on top of jQuery, Prototype, and YUI.

We explored many of the core UI widgets that the framework provides and showed that the number of prebuilt widgets helps rapid application development efforts. In

doing that, we talked about some of the changes that Ext JS 3.0 has brought forth, such as Flash charts.

Last, we discussed where to download and how to set up the framework with each individual base framework. We created a "Hello world" example of how to use an Ext JS `Window` to retrieve an HTML fragment via Ajax with a few simple lines of JavaScript.

In the chapters to follow, we'll explore how Ext JS works from the inside out. This knowledge will empower you to make the best decisions when building well-constructed UIs and better enable you to leverage the framework effectively. This will be a fun journey.

JAVASCRIPT/WEB DEVELOPMENT

EXT JS IN ACTION

Jesus Garcia

This cross-browser JavaScript library provides an extensive collection of high-quality widgets, an intuitive and extensible component model, and a rich API that enterprise developers find especially comfortable to use. And they have used it to build rock-solid web applications, in many very different companies including Adobe, Aetna, Amazon, Best Buy, Hallmark, Panasonic, Pixar, Siemens, Sony, and Visa.

Ext JS in Action is a comprehensive guide to Ext JS. By following its rich examples, patterns, and best practices, you'll achieve the kinds of results you only see in top JavaScript applications. This book thoroughly explores every class, component, and model, and shows you how to build rich, dynamic user interfaces and responsive applications. You will learn Ext JS inside and out—and your apps will stand out from the crowd.

What's Inside

- Explore the depths of Ext JS 3.0
- Create rich and dynamic UIs
- Extend the framework and write plug-ins
- Watch the author develop an Ext JS app

This book assumes a reader with a foundation in JavaScript, but no previous exposure to Ext JS.

Jesus "Jay" Garcia is an Ext JS community leader. Since 2006, he has deployed and optimized Ext JS world-class applications for many corporations.

For online access to the author and a free ebook for owners of this book, go to manning.com/ExtJSinAction

"Outstanding resource for using Ext JS!"
—Dan McKinnon
MITRE Corporation

"An easy-to-understand walk-through of Ext JS 3."
—Mitchell Simoens
Senior Web Developer

"Takes the complexity out of a complex interface."
—Ric Peller
Management Dynamics

"Really EXTends your knowledge!"
—Jeroen Benckhuijsen
Atos Origin

"This is a handy book!"
—Orhan Alkan, Oracle

ISBN 13: 978-1-935182-11-5
ISBN 10: 1-935182-11-0
54999
9 781935 182115

MANNING

$49.99 / Can $57.99 [INCLUDING eBOOK]