Displaying Hierarchical Data with NestedList

The Data package is a group of classes that gives Sencha Touch the ability to fetch and read data from a myriad of sources, including mobile WebKit’s HTML5 Session, Local, and Database Storage methods. Sencha Touch can read data in a variety of formats including XML, Array, JSON, and Tree (nested). Views are widgets that implement data Stores to display data. If you’re an Ext JS developer, you might be surprised to learn that Sencha Touch only provides three data-bound views and that this list excludes a GridPanel. What we have at our disposal is a DataView, List, and NestedList.

The DataView and List widgets are designed to display data in a linear set. However, there are times where you want to display nested data. For that, you'll need to use the NestedList widget. Here is a NestedList widget in action.
In the illustration above, I have set up a NestedList for the selection of a food item. There are two main categories, Drinks and Food. I selected Drinks (left), which brought me to three sub-categories. I then chose Sports Drinks (center), which led me to the last section of items, which is a list of sports drinks (right). All of this is done with the slide animation.

If you have flat data, DataView or List is what you should use, but if you have hierarchical data, then they are not equipped to handle that data. They will show the first level, also called the root level, of data, but not any of the child data levels. Not to leave you out in the cold, Sencha Touch has a component equipped to handle hierarchical data called NestedList. As the name suggests, the NestedList component will show a List for each nested level of data. NestedList extends Container and uses the Card layout. Each item is a List and the number of items depends on the number of levels in your hierarchical data. NestedList also has a top docked TitleBar to show a back Button when not on the root level, allowing you to traverse backwards. This TitleBar will also show the title text, usually the same text that is shown on the row you tapped on.

To get the data, you don't just use a regular Store like we used for DataView and List; Store also doesn't support hierarchical data. For this we have to use a TreeStore.

**Understanding the hierarchical data**

Before we go diving into the NestedList and TreeStore, we need to look at what the hierarchical data should look like. Hierarchical data is nested data where each record is known as a node. The data becomes hierarchical when a node contains child nodes, and those can contain child nodes, and so on. A node that doesn't have any children is known as a leaf. I’m sure you are familiar with the file system on a computer where folders can have folders and files. A node that has children is like a folder, and a leaf is like a file.

Here is what a basic JSON sample of hierarchical data would look like:

```json
{
  "children" : [
    { "text" : "Mitchell Simoens",
      "children" : [
        { "text" : "@msims84",
          "leaf" : true
        },
        { "text" : "http://www.linkedin.com/in/mitchellsimoens",
          "leaf" : true
        }
      ]
    },
    { "text" : "Jay Garcia",
      "children" : [
        { "text" : "@_jdg",
          "leaf" : true
        },
        { "text" : "http://www.linkedin.com/in/tdginnovations",
          "leaf" : true
        }
      ]
    },
    { "text" : "Anthony De Moss",
      "children" : [
        { "text" : "@ademoss1",
          "leaf" : true
        },
        { "text" : "http://www.linkedin.com/in/ademoss",
          "leaf" : true
        }
      ]
    }
  ]
}
```

For source code, sample chapters, the Online Author Forum, and other resources, go to [http://www.manning.com/garcia2/](http://www.manning.com/garcia2/)
Looking at the JSON above, you can see the hierarchy. The top level is the root of the data, which has a single property, named children. Each level will have a children property to specify what children that item has, unless that item has the leaf property. It is assumed each item has children unless the leaf property is set to true, meaning that it will not have children; it’s the end of the road. Each item, leaf or not, will have a text property which will be displayed.

**Using TreeStore**

As I’ve mentioned at the beginning of this article, we need to use TreeStore, not a regular Store, to consume hierarchical data. There isn’t a difference when creating a TreeStore, but the TreeStore is able to handle hierarchical data where Store cannot.

Listing 1 is a sample TreeStore with the associated Model.

### Listing 1 Sample TreeStore

```javascript
Ext.define('Author', {
    extend : 'Ext.data.Model',
    config : {
        fields : [
            'text',
            'link'
        ]
    }
});
var store = Ext.create('Ext.data.TreeStore', {
    model : 'Author',
    autoLoad : true,
    proxy : {
        type : 'ajax',
        url : 'authors-tree.json'
    }
});
```

- **#1** Creates the Author Model
- **#2** Specifies the text field
- **#3** Specifies additional fields
- **#4** Creates the TreeStore
- **#5** Specifies the JSON file

Creating a TreeStore is easy. Like a regular Store, we need to create a Model class definition; in this case we are naming it Author #1. The only thing required is that we specify which field we intend to display, which in this case is text #2. We also specified another field that we will use later, named link #3. Now we create the TreeStore #4 and give it the model, autoLoad, and proxy configurations, and within the proxy configuration we are going to use the Ajax proxy with the url to the authors-tree.json file #5 that contains the JSON from section 7.4.1.

Run the code from listing 7.12 and the TreeStore will load the JSON from the authors-tree.json file and parse the hierarchical data to be used in a NestedList.

### Creating a basic NestedList

So we can load data, but that’s no fun without being able to display the data. As we discussed earlier, we cannot use a DataView or a List to display hierarchical data. For this task, Sencha Touch has the NestedList component. As we mentioned earlier in section 7.4, NestedList is just a Container that uses Card layout to display child Lists. Listing 2 is a basic example of how to display the JSON from our earlier discussion using TreeStore.

### Listing 2 Basic NestedList

```javascript
var nestedlist = Ext.create('Ext.dataview.NestedList', {
    fullscreen : true,
});
```

For source code, sample chapters, the Online Author Forum, and other resources, go to [http://www.manning.com/garcia2/](http://www.manning.com/garcia2/)
There’s not much going on in our simple nestedlist. of course we create the NestedList #1 and give it the fullscreen configuration to add it to Ext.Viewport so it can be displayed. We also specify the store #2 for the NestedList to use. We give it the title configuration, which is a title #3 for the root level. The title configuration is actually an optional configuration, but, without it, the NestedList will look a little funny because the top docked TitleBar will be blank. With the title configuration, when you are on the root level, the TitleBar will display this value so your user can tell what they are looking at. Check out figure 2 to see what you get you run listing 2.

![Figure 2 Results of listing 3 showing a simple NestedList](image)

Looking at figure 2, the image on the left is the root level. You can see the title that we specified in listing 2 at the top. Without it, you would just get the blue toolbar docked there, which doesn’t look the best. Plus, with the title configuration set, we can now see that this list is the Authors. Imagine we tapped on the first list item, Mitchell Simoens. It would then animate to the image on the right to show the children of the Mitchell Simoens list item. Do note that the top TitleBar now displays the same text for the title as the list item we tapped on. If we tapped on Anthony De Moss, it would say Anthony De Moss instead of Mitchell Simoens in the TitleBar. If the text were longer than could fit in the TitleBar, it would be cut off and have ellipsis to give a better presentation than just letting the text run off screen. Also note in the TitleBar that we now have a Button that says we can go back to the Authors list. This Button uses the back ui and uses the text from what was in the TitleBar as the title. In this case, the last title was Authors. If we didn’t specify the title configuration on the NestedList, the Button would display the text Back, so the back Button should never be blank.

**Showing details**

So what happens when we get to the level that is a leaf? You can tap on it and it will get selected but nothing happens. Because it is a leaf, there are no more children, but usually you would want to show details about that leaf. A configuration of the NestedList that we haven’t covered so far is the detailCard configuration. What this
detailCard configuration does when you tap on a leaf is that it will create and add a detailCard to the NestedList and animate to this component. The detailCard can be anything you want it to be: another List, a Container, or a form, anything you need it to be.

Let’s take a look at how to work with it (listing 3), and then we can talk about what we see.

### Listing 3 NestedList details

```javascript
var nestedlist = Ext.create('Ext.dataview.NestedList', {
    fullscreen: true,
    store: store,
    title: 'Authors',
    detailCard: {
        xtype: 'container',
        tpl: [
            '<h1>Titter Page: ',
            '{link}',
            '</a>',
            '</h1>',
            '<br />
            <br />
            ',
            'Tapping on this link will take you outside of this app'
        ]
    },
    listeners: {
        leafitemtap: function(nestedlist, list, index, t, record) {
            var detailCard = nestedlist.getDetailCard();
            detailCard.setData(record.getData());
        }
    }
});
```

- **#1** Specifies the config for detailCard
- **#2** Uses xtype container
- **#3** Includes a template to display data
- **#4** Adds a listener to the leafitemtap
- **#5** Gets the detailCard instance
- **#6** Applies data to the detailCard

We took the NestedList from listing 3 and added the detailCard configuration **#1**. This is the configuration object that will be used to create the component to be used to display details about the leaf that was tapped. This detail card will use the XType 'container' **#2** and a template using the tpl configuration **#3** to accept data and display it. If we left it here, you would get a blank screen, so we need to add a leafitemtap event listener **#4** that will only be fired when you tap on a list item that is a leaf. Within this leafitemtap event listener, we need to get the reference to the detail card using the getDetailCard method **#5** on the NestedList, which is passed as an argument on the function. Finally, to show data, we execute the setData method on the detail card instance, passing in the data from the record that was tapped on **#6**. Now you should see what is shown in figure 3.
Instead of showing another List, we will see the component we specified in the detailCard config. With the logic in the `leafItemTap` event listener, we applied the data from the record we tapped onto the detail card. Now you have the ability to create a view that can handle hierarchical data and even show a detail view on it. Can you imagine what you can use `NestedList` for? It’s powerful but very simple to configure.

**Summary**

You saw what hierarchical data looks like and how to consume it with the specialized `TreeStore`. We used the `NestedList` component to display this data one level at a time and used the `detailCard` configuration in the `NestedList` to display details about a leaf item.
Here are some other Manning titles you might be interested in:

**Hello! iOS Development**  
Lou Franco and Eitan Mendelowitz

**50 Android Hacks**  
Carlos Sessa

**Objective-C Fundamentals**  
Christopher K. Fairbairn, Johannes Fahrenkrug, and Collin Ruffenach

Last updated: November 20, 2012