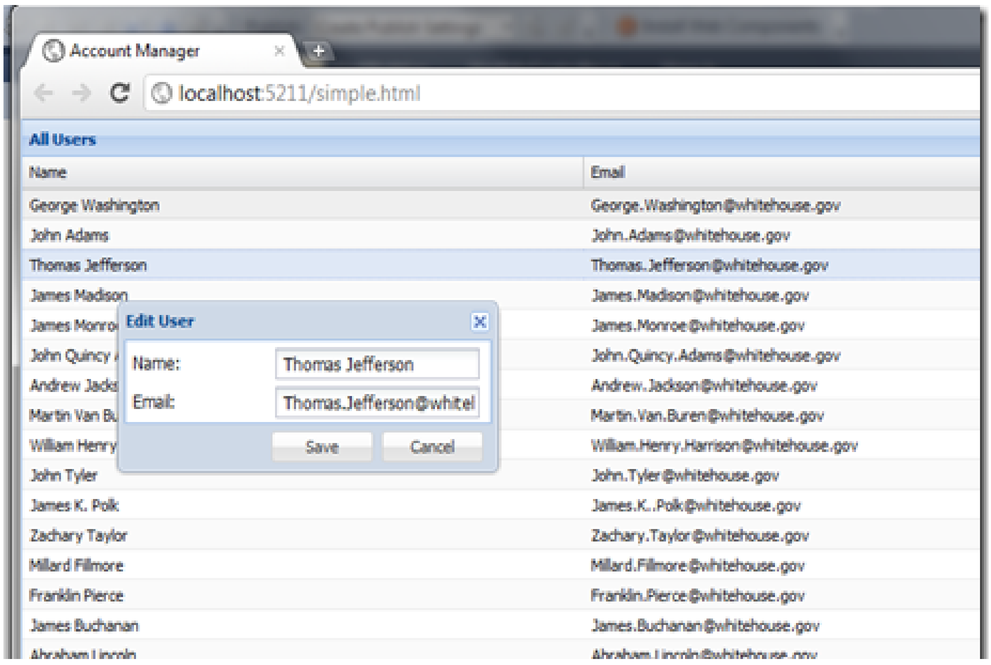
November 4th, 2011Building an Sencha's ExtJS 4.0 MVC Application With Microsoft's ASP.NET MVC3 Series / Basics

Part 1	Basics
Part 2	Enhanced (coming)

Introduction

In this series of articles, we will take the reference application build by the Sencha product team for using Sencha's MVC pattern running with an ASP.NET 4.0 project (IIS in production). The first article takes the reference Sencha MVC app and with almost no changes, makes it work with the ASP.NET Visual Studio 2010 project. By default, the application works with JSON static files. We change that to work with an ASP.NET MVC3 project. The second article in the series embellishes the application to include a more real user experience by adding additional functionality and data. The improved functionality includes both date and number columns as well as paging. It also adds functionality for inserts and deletes which are left out of the base app from Sencha.

The Final Project when running:



Name	Email
George Washington	George.Washington@whitehouse.gov
John Adams	John.Adams@whitehouse.gov
Thomas Jefferson	Thomas.Jefferson@whitehouse.gov
James Madison	James.Madison@whitehouse.gov
James Monro	James.Monroe@whitehouse.gov
John Quincy	John.Quincy.Adams@whitehouse.gov
Andrew Jacks	Andrew.Jackson@whitehouse.gov
Martin Van Bu	Martin.Van.Buren@whitehouse.gov
William Henry	William.Henry.Harrison@whitehouse.gov
John Tyler	John.Tyler@whitehouse.gov
James K. Polk	James.K..Polk@whitehouse.gov
Zachary Taylor	Zachary.Taylor@whitehouse.gov
Millard Fillmore	Millard.Fillmore@whitehouse.gov
Franklin Pierce	Franklin.Pierce@whitehouse.gov
James Buchanan	James.Buchanan@whitehouse.gov
Abraham Lincoln	Abraham.Lincoln@whitehouse.gov

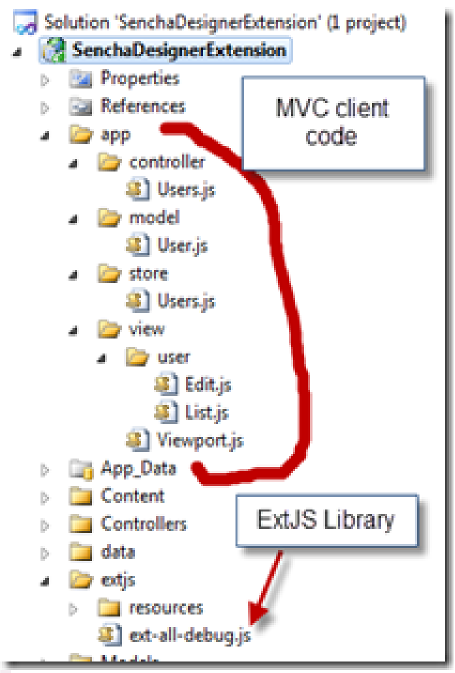
Name: Thomas Jefferson
Email: Thomas.Jefferson@whitel

What Is Sencha's MVC Application

I don't want to spend a lot of time here explaining what Sencha does really well in there article but I feel a short explanation is helpful (http://www.sencha.com/learn/the-mvc-application-architecture/).

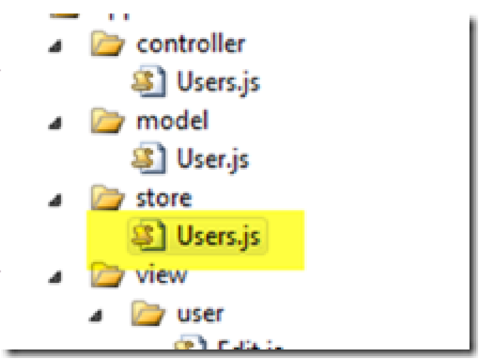
So, here we go. Basically ExtJS is a client side JavaScript framework which allows you to build the full user experience in JavaScript. That is, all the grids, forms, tabs, etc. are all generated by a client side JavaScript library. All that goes up and down from the server is JSON (after the JavaScript framework and code and assets are downloaded). This means that on the client side, all the code for displaying a grid for example, reaching out for data, pushing data changes, etc. all need to be done on the client (browser). Sencha's ExtJS MVC pattern helps organize all these functions into a set of directories and functions that make building apps scalable.

Below is what Sencha's app in Visual Studio looks like:



ASP.NET Implementation

The major change to the stock MVC app from Sencha is to change there Store class. Here is the new Store Class.

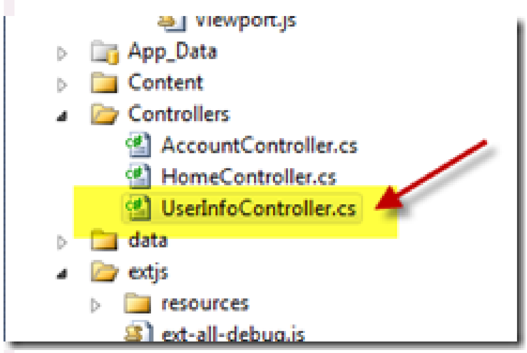


And the JavaScript itself:

```
var writer = new Ext.data.JsonWriter({
    type: 'json',
    encode: false,
    listful: true
    listful: true,
    writeAllFields: true,
    returnJson: true
});

var reader = new Ext.data.JsonReader({
    totalProperty: 'total',
    successProperty: 'success',
    idProperty: 'Id',
    root: 'Data',
    messageProperty: 'message'
});

var proxy = new Ext.data.HttpProxy({
    reader: reader,
    writer: writer,
    type: 'ajax',
    api: {
        read: '/UserInfo/Get',
        create: '/UserInfo/Create',
        update: '/UserInfo/Update',
        destroy: '/UserInfo/Delete'
    },
    headers: {
        'Content-Type': 'application/json; charset=UTF-8'
    }
});

Ext.define('AM.store.Users', {
    extend: 'Ext.data.Store',
    model: 'AM.model.User',
    autoLoad: true,
    paramsAsHash: true,
    proxy: proxy
});
```

The primary change is to the reader. Notice that the API's are defined now to point the ASP.NET controller UserInfo. Also, the column names have been changed to start with uppercase to be consistent with public properties in ASP.NET. That is, id,name and email are now Id,Name and Email.

Now, we need to implement the controller on the ASP.NET. Basically, we implement GET on the read and POST on the write. The controller class in c# is as follows: (Visual Studio Solution Explorer first, then the controller code itself)

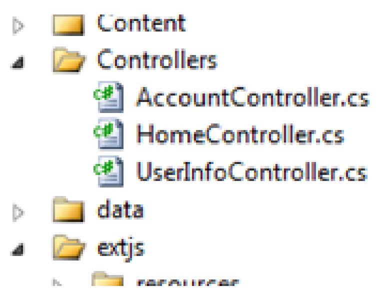


```
public class UserInfoController : Control
ler
{
    public JsonResult Get(int? start, int? limit)
    {
        using (var db = new Db())
        {
            start = start.HasValue ? start.Value : 0;
            limit = limit.HasValue ? limit.Value : Int32.MaxValue
;
            int cnt = db.Users.Count();
            var recs = db.Users.OrderBy(a => a.Id).
                Skip(start.Value).Take(limit.Value).ToList();
            return Json(new
                                {
                                    Data = recs,
                                    total = cnt
                                }, JsonRequestBehavior.AllowGet);
        }
    }


    [HttpPost]
    public JsonResult Update(UserInfo data)
    {
        bool success = false;
        string message = "no record found";
        if (data != null && data.Id > 0)
        {
            using (var db = new Db())
            {
                var rec = db.Users.Where(a => a.Id == data.Id).
                    FirstOrDefault();
                rec.Name = data.Name;
                rec.Email = data.Email;
                db.SaveChanges();
                success = true;
                message = "Update method called successfully";
            }
        }

        return Json(new
                        {
                            data,
                            success,
                            message
                        });
    }
}
```

Notice that the Get method returns a JsonResult. This conveniently returns the data in JSON format. Part of the secret sauce here is in the Db(). This app used Microsoft's new EntityFamework CodeFirst. Following the standard CodeFirst model, we implement the simple c# code and it automagically creates the data tables (and data) in the database (my example uses SqlServer CE because it can be run without installing any database on your windows computer, the code is all in-memory).

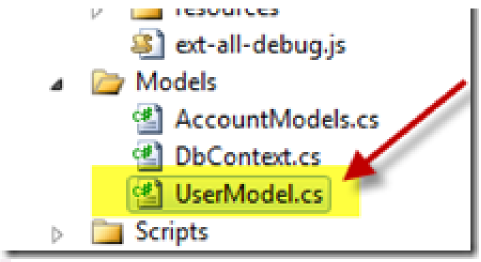
If we look at the Model on the ASP.NET MVC3 side, you'll notice the c# attributes and the DbContext that do the magic for you. There are some excellent getting started guides on the web including one from Julie Lerman (walk through video: http://msdn.microsoft.com/en-us/data/gg715119 and http://msdn.microsoft.com/en-us/data/gg685467).

Here is the Visual Studio Solution Explorer for the model:



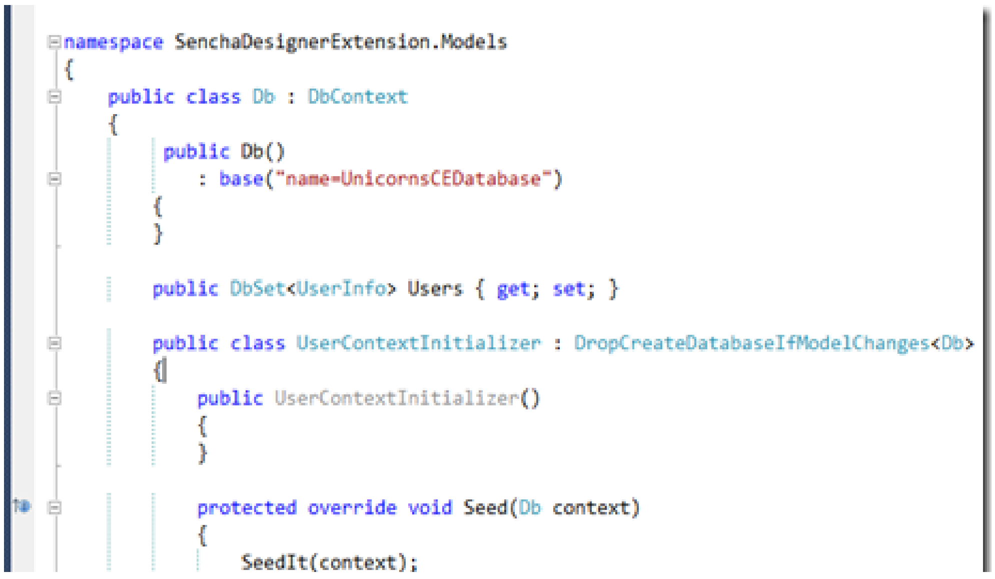
and here is the model code in c#:

```
namespace SenchaDesignerExtension.Models
```



```
{
    public class UserInfo
    {
        [Key]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Ident
ity)]
        public int Id { get; set; }

        [MaxLengthAttribute(256)]
        public string Email { get; set; }

        [MaxLengthAttribute(256)]
        public string Name { get; set; }

    }
}
```

and the actual repository which is in the file DbContext.cs (screen shot for brevity below, the code is attached in the example.



```
namespace SenchaDesignerExtension.Models
{
    public class Db : DbContext
    {
        public Db()
            : base("name=UnicornsCEDatabase")
        {
        }

        public DbSet<UserInfo> Users { get; set; }

        public class UserContextInitializer : DropCreateDatabaseIfModelChanges<Db>
        {
            public UserContextInitializer()
            {
            }

            protected override void Seed(Db context)
            {
                SeedIt(context);
```

peterkellner.net

http://peterkellner.net/2011/11/04/building-an-senchas-extjs-4-0-mvc-application-with-microsofts-asp-net-mvc3-series-basics/

http://goo.gl/ffgS