Ed Spencer

A JavaScript Architect

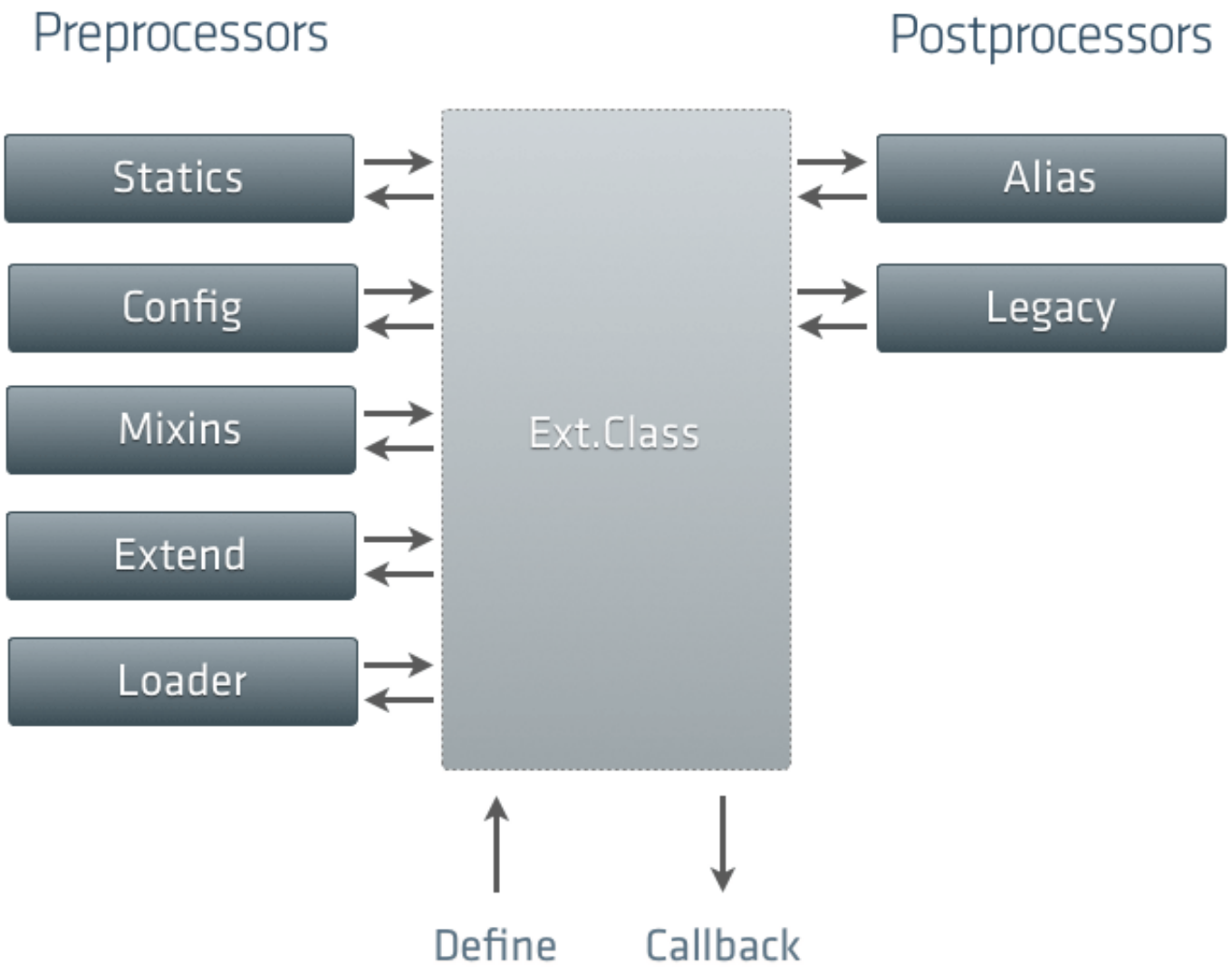
Ext JS 4: The Class Definition Pipeline

JANUARY 25, 2011 27 COMMENTS (HTTP://EDSPENCER.NET/2011/01/25/EXT-JS-4-THE-CLASS-DEFINITION-PIPELINE/#COMMENTS)

Last time, we looked at some of the features of the new class system in Ext JS 4 (http://edspencer.net/2011/01/classes-in-ext-js-4-under-the-hood.html), and explored some of the code that makes it work. Today we're going to dig a little deeper and look at the class definition pipeline – the framework responsible for creating every class in Ext JS 4.

As I mentioned last time (http://edspencer.net/2011/01/classes-in-ext-js-4-under-the-hood.html), every class in Ext JS 4 is an instance of Ext.Class. When an Ext.Class is constructed, it hands itself off to a pipeline populated by small, focused processors, each of which handles one part of the class definition process. We ship a number of these processors out of the box – there are processors for handling mixins, setting up configuration functions and handling class extension.

The pipeline is probably best explained with a picture. Think of your class starting its definition journey at the bottom left, working its way up the preprocessors on the left hand side and then down the postprocessors on the right, until finally it reaches the end, where it signals its readiness to a callback function:



(http://edspencerdotnet.files.wordpress.com/2011/01/processors.png)

The distinction between preprocessors and postprocessors is that a class is considered 'ready' (e.g. can be instantiated) after the preprocessors have all been executed. Postprocessors typically perform functions like aliasing the class name to an xtype or back to a legacy class name – things that don't affect the class' behavior.

Each processor runs asynchronously, calling back to the Ext.Class constructor when it is ready – this is what enables us to extend classes that don't exist on the page yet. The first preprocessor is the Loader, which checks to see if all of the new Class' dependencies are available. If they are not, the Loader can dynamically load those dependencies before calling back to Ext.Class and allowing the next preprocessor to run. We'll take another look at the Loader in another post.

After running the Loader, the new Class is set up to inherit from the declared superclass by the Extend preprocessor. The Mixins preprocessor takes care of copying all of the functions from each of our mixins, and the Config preprocessor handles the creation of the 4 config functions we saw

last time (e.g. getTitle, setTitle, resetTitle, applyTitle – check out yesterday's post (http://edspencer.net/2011/01/classes-in-ext-js-4-under-the-hood.html) to see how the Configs processor helps out).

Finally, the Statics preprocessor looks for any static functions that we set up on our new class and makes them available statically on the class. The processors that are run are completely customizable, and it's easy to add custom processors at any point. Let's take a look at that Statics preprocessor as an example:

```
1  //Each processor is passed three arguments - the class under constructio
2  //the configuration for that class and a callback function to call when
3  Ext.Class.registerPreprocessor('statics', function(cls, data, callback)
4      if (Ext.isObject(data.statics)) {
5          var statics = data.statics,
6              name;
7
8          //here we just copy each static function onto the new Class
9          for (name in statics) {
10             if (statics.hasOwnProperty(name)) {
11                 cls[name] = statics[name];
12             }
13         }
14     }
15
16     delete data.statics;
17
18     //Once the processor's work is done, we just call the callback funct
19     if (callback) {
20         callback.call(this, cls, data);
21     }
22 });
23
24 //Changing the order that the preprocessors are called in is easy too -
25 Ext.Class.setDefaultPreprocessors(['extend', 'mixins', 'config', 'static
```

What happens above is pretty straightforward. We're registering a preprocessor called 'statics' with Ext.Class. The function we provide is called whenever the 'statics' preprocessor is invoked, and is passed the new Ext.Class instance, the configuration for that class, and a callback to call when the preprocessor has finished its work.

The actual work that this preprocessor does is trivial – it just looks to see if we declared a 'statics' property in our class configuration and if so copies it onto the new class. For example, let's say we want to create a static getNextId function on a class:

```
1  Ext.define('MyClass', {
2      statics: {
3          idSeed: 1000,
4          getNextId: function() {
5              return this.idSeed++;
6          }
```

```
    }
});
```

Because of the Statics preprocessor, we can now call the function statically on the Class (e.g. without creating an instance of MyClass):

```
MyClass.getNextId(); //1000
MyClass.getNextId(); //1001
MyClass.getNextId(); //1002
... etc
```

Finally, let's come back to that callback at the bottom of the picture above. If we supply one, a callback function is run after all of the processors have run. At this point the new class is completely ready for use in your application. Here we create an instance of MyClass using the callback function, guaranteeing that the dependency on Ext.Window has been honored:

```
Ext.define('MyClass', {
    extend: 'Ext.Window'
}, function() {
   //this callback is called when MyClass is ready for use
   var cls = new MyClass();
   cls.setTitle('Everything is ready');
   cls.show();
});
```

That's it for today. Next time we'll look at some of the new features in the part of Ext JS 4 that is closest to my heart – the data package.

FILED UNDER EXAMPLES, EXTJS, JAVASCRIPT, MISC TAGGED WITH CLASSES, EXTJS, EXTJS4

27 Responses to *Ext JS 4: The Class Definition Pipeline*

Scott says:
January 25, 2011 at 10:19 am
Sorry for being picky: shouldn't the first call to MyClass.getNextId() return 1001 instead of 1000?

Reply
Ed Spencer says:
January 25, 2011 at 10:24 am
@Scott – try it! You'll get 1000. Now replace idSeed++ with ++idSeed and it'll start with 1001. These are known as pre and post increment – http://www.hunlock.com/blogs/The_Complete_Javascript_Number_Reference is a good reference