# Basic JavaScript Part 9: Reusing Methods of Other Objects

January 31st, 2011

Here are the links to the previous installments:

1. Functions
2. Objects
3. Prototypes
4. Enforcing New on Constructor Functions
5. Hoisting
6. Automatic Semicolon Insertion
7. Static Properties and Methods
8. Namespaces

In one of my previous posts, I mentioned that functions in JavaScript are plain objects that can have their own properties and methods. One of these methods that are available for every function object is a method named *call(),* which is defined on the prototype of *Function.* This method allows you to *'reuse'* a method from another object. Let's talk code.Suppose we have an object called *podcast* which has a *download()* function:

```
var podcast = {
    name : 'Astronomy podcast',
    download : function(episode) {
        console.log('Downloading ' + episode + ' of ' + this.name);
    }
};

podcast.download('the first episode');
```

The output we get by executing this function is exactly what we expect it to be:

> *"Downloading the first episode of Astronomy podcast"*

Now suppose we bring on another object called *screencast* :

```
var screencast = {
    name: 'Node tuts'
};
```

Wouldn't it be nice if we could somehow reuse the *download()* method of the *podcast* object so that we can download a *screencast* as well. We can do this by invoking *call()* on the *download()*method like so:

```
podcast.download.call(screencast, 'the last episode');
```

This gives us the following output:

> *"Downloading the last episode of Node tuts"*

What exactly happened here? Well, we just invoked the *call()* method on the *download* function object, specifying a reference to the *screencast* object and a string argument that describes the particular episode. The *download()* method got invoked with the *screencast* object bound to*this*. Therefore, the *download()* method uses the *name* property of the *screencast* object instead of the *name* of the *podcast* object.

There's also another method with similar functionality defined on the Function prototype named *apply()*. This method behaves exactly the same as the *call()* method, except that the arguments passed to the underlying method in question need to be passed as an array.

```
podcast.download.apply(screencast, ['the last episode']);
```

Utilizing the *call()/apply()* methods is a very powerful way to reuse features of objects that are built into JavaScript (like Array, String, etc. …) or from third-party libraries. But I don't recommend applying this

technique when you own the code of the method you want to reuse. A simple refactoring like extracting the particular method into another object is a much better approach.

Until next time.