One of the reasons for Node.js’ high speed is the fact that it is coded around events. Instead of reading all the files necessary with every request (like PHP), with Node you just start your server, initiate most of the variables, declare your functions and then just wait for an event to occur.

While Node.js has some useful built-in events, like the request event, wouldn’t it be useful to be able to create our own events and be able to trigger them ourselves? That’s what I’m going to talk about in this article.

First I will show you how normal events are emitted. As event I will take the example that when a person enters a shop, a bell may ring to show his presence.  It will act a bit like the [observer pattern](http://en.wikipedia.org/wiki/Observer_pattern) where our event acts like subject and all functions attached to the event are like observers. So the shop example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var events = require('events');  var eventEmitter = new events.EventEmitter();    var ringBell = function ringBell()  {    console.log('ring ring ring');  }  eventEmitter.on('doorOpen', ringBell);    eventEmitter.emit('doorOpen'); |

First we load the [events module](http://nodejs.org/api/events.html), which is part of the Node.js core. Then we create a new instance of theEventEmitter class (which we are going to extend later). After then we place the ringBell function inside a variable so that it can be called that way. It simply prints ‘ring ring ring’ in our console.

Now the interesting things are coming. We add our ringBell function to the function list of the doorOpenevent. This is done by the eventEmitter.on() method and as first argument the event, as second argument the function to be added. This doesn’t really do anything, it just registers our function. The real magic happens the line after that, when we emit our event. Calling the emit() method will execute all the functions that are registered with the on method.

This isn’t that interesting, could just have called that one function if we wanted the bell to ring. But that’s what makes events that interesting: you can register just as many functions as you want.

We could also have done this, for example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | eventEmitter.on('doorOpen', ringBell);  eventEmitter.on(‘doorOpen’, doSomething);  eventEmitter.on(‘doorOpen’, doSomethingElse);    eventEmitter.emit('doorOpen'); |

That would also work and would more use the functionalities that the EventEmitter provides us.

We can also use functions with arguments as listeners:

|  |  |
| --- | --- |
| 1  2  3  4  5 | eventEmitter.on(‘doorOpen’, function(ring)  {      Console.log(ring);  }  eventEmitter.emit(‘doorOpen’, ‘ringeling’); |

We just pass the arguments in the emit() method.

While this is all very powerful, a common practice within the Node community is to inherit from the eventEmitter class. We could do this by having a door class, with an open() method which will emit thedoorOpen event. Take a look at this code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | var events = require('events');    function Door(colour) {    this.colour = colour;    events.EventEmitter.call(this);      this.open = function()    {    this.emit('open');    }  }    Door.prototype.\_\_proto\_\_ = events.EventEmitter.prototype;    var frontDoor = new Door('brown');    frontDoor.on('open', function() {      console.log('ring ring ring');    });  frontDoor.open(); |

In the constructor of our Door object, we set the colour of the door and, we use the call() method of our EventEmitter object, which executes the constructor method of EventEmitter. Then we declare our open method, which emits the ‘open’ event. This line:

|  |  |
| --- | --- |
| 1 | Door.prototype.\_\_proto\_\_ = events.EventEmitter.prototype; |

Copies all of the EventEmitter properties to the Door object.

Then we create our front door, which is an instance of Door and which has a brown-ish colour. We then add an event listener and finally we open the door, and print a message to our console. I hope you all see that this events module is very powerful and useful!

Then finally, the events module provides us with a way to list all of the event listeners, attached to an event and a way to delete event listeners.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var ring = function()  {      console.log('ring');  }  frontDoor.on('open', ring);    console.log(require('util').inspect(frontDoor.listeners('open'))); // Outputs ring |

You can do that by using the listeners property. Of course this only works if you didn’t use an anonymous function as event listener.

If we would ever want to, we could remove the bell from our door:

|  |  |
| --- | --- |
| 1 | frontDoor.removeListener('open', ring); |

Or we could even remove all the listeners:

|  |  |
| --- | --- |
| 1 | frontDoor. .removeAllListeners(‘open’); |