API

The oboe function

Oboe.js exposes only one function, `oboe`, which is used to instantiate a new Oboe instance. Calling this function starts a new HTTP request unless the caller is managing the stream themselves.

```
oboe( String url )

oboe({
   method: String,
   url: String,
   headers: Object,
   body: String|Object,
   cached: Boolean
})

// the doMethod style of calling is deprecated
// and will be removed in v2.0.0:
oboe.doGet(    url )
oboe.doDelete( url )
oboe.doPost(   url, body )
oboe.doPut(    url, body )
oboe.doPatch(  url, body )

oboe.doGet(    {url:String, headers:Object, cached:Boolean} )
oboe.doDelete( {url:String, headers:Object, cached:Boolean} )
oboe.doPost(   {url:String, headers:Object, cached:Boolean, body:String|Object} )
oboe.doPut(    {url:String, headers:Object, cached:Boolean, body:String|Object} )
oboe.doPatch(  {url:String, headers:Object, cached:Boolean, body:String|Object} )
```

The `method`, `headers`, `body`, and `cached` arguments are optional.

- If `method` is not given Oboe defaults to `GET`.
- If `body` is given as an object it will be stringified using `JSON.stringify` prior to sending. The Content-Type request header will automatically be set to `text/json` unless a different value is explicitly given.
- If the cached option is given as `false` cachebusting will be applied by appending `_={timestamp}` to the URL's query string. Any other value will be ignored.

BYO stream

Under Node.js you may also pass `oboe` an arbitrary ReadableStream for it to read JSON from. It is your responsibility to initiate the stream and Oboe will not start a new HTTP request on your behalf.

```
oboe( stream )
```

node event

The methods `.node()` and `.on()` are used to register interest in particular nodes by providing JSONPath patterns. As the JSON stream is parsed the Oboe instance checks for matches against these patterns and when a matching node is found it emits a `node` event.

```
.on('node', pattern, callback)

// 2-argument style .on() ala Node.js EventEmitter#on
.on('node:{pattern}', callback)

.node(pattern, callback)

// register several listeners at once
.node({
   pattern1 : callback1,
   pattern2 : callback2
});
```

When the callback is notified, the context, `this`, is the Oboe instance, unless it is bound otherwise. The callback receives three parameters:

`node`	The node that was found in the JSON stream. This can be any valid JSON type - `Array`, `Objec`
`path`	An array of strings describing the path from the root of the JSON to the matching item. For ex
`ancestors`	An array of the found item's ancestors such that `ancestors[0]` is the JSON root, `ancestors[a`

```
oboe('friends.json')
   .node('name', function(name){
      console.log('You have a friend called', name);
   });
```

path event

Path events are identical to node events except that they are emitted as soon as matching paths are found, without waiting for the thing at the path to be revealed.

```
.on('path', pattern, callback)

// 2-argument style .on() ala Node.js EventEmitter#on
.on('path:{pattern}', callback)

.path(pattern, callback)

// register several listeners at once
.path({
   pattern1 : callback1,
   pattern2 : callback2
});


oboe('friends.json')
   .path('friend', function(name){
      friendCount++;
   });
```

One use of path events is to start adding elements to an interface before they are complete.

done event

```
.done(callback)

.on('done', callback)
```

Done events are fired when the response is complete. The callback is passed the entire parsed JSON.

In most cases it is faster to read the JSON in small parts by listening to `node` events (see above) than waiting for it to be completely download.

```
oboe('resource.json')
   .on('done', function(parsedJson){
      console.log('Request complete', parsedJson);
   });
```

start event

```
.start(callback)

.on('start', callback)
```

Start events are fired when Oboe has parsed the status code and the response headers but has not yet received any content from the response body.

The callback receives two parameters:

name	type	
`status`	`Number`	HTTP status code
`headers`	`Object`	Object of response headers

```
oboe('resource.json')
   .on('start', function(status, headers){
      console.log('Resource cached for', headers.Age, 'secs');
   });
```

Under Node.js this event is never fired for BYO streaming.

fail event

```
.fail(callback)

.on('fail', callback)
```

Fetching a resource could fail for several reasons:

- Non-2xx status code
- Connection lost
- Invalid JSON from the server
- Error thrown by an event listener

The fail callback receives an object with four fields:

Field	Meaning

`thrown`	The error, if one was thrown
`statusCode`	The status code, if the request got that far
`body`	The response body for the error, if any
`jsonBody`	If the server's error response was JSON, the parsed body

```
oboe('/content')
   .fail( function( errorReport ){
      if( 404 == errorReport.statusCode ){
         console.error('no such content');
      }
   });
```

.header([name])

```
.header()

.header(name)
```

`.header()` returns one or more HTTP response headers. If the name parameter is given that named header will be returned as a String, otherwise all headers are returned as an Object.

`undefined` wil be returned if the headers have not yet been received. The headers are available anytime after the `start` event has been emitted. They will always be available from inside a `node`, `path`, `start` or `done` callback.

`.header()` always returns undefined for non-HTTP streams.

```
oboe('data.json')
   .node('id', function(id){
      console.log(   'Server has id', id,
                     'as of', this.headers('Date'));
   });
```

.root()

At any time, call `.root()` on the oboe instance to get the JSON parsed so far. If nothing has yet been received this will return `undefined`.

```
var interval;
```

```
oboe('resourceUrl')
   .start(function(){
      interval = window.setInterval(function(){
         console.log('downloaded so far:', this.root());
      }.bind(this), 10);
   })
   .done(function(completeJson){
      console.log('download finished:', completeJson);
      window.clearInterval(interval);
   });
```

.forget()

```
.node('*', function(){
   this.forget();
})
```

`.forget()` is a shortcut for .removeListener() in the case where the listener to be removed is currently executing. Calling `.forget()` on the Oboe instance from inside a `node` or `path` callback de-registers that callback.

```
// Display only the first ten downloaded items
// but place all in the model

oboe('/content')
   .node('!.*', function(item, path){
      if( path[0] == 9 )
         this.forget();

      displayItem(item);
   })
   .node('!.*', function(item){
      addToModel(item);
   });
```

.removeListener()

```
.removeListener('node', pattern, callback)
.removeListener('node:{pattern}', pattern, callback)

.removeListener('path', pattern, callback)
.removeListener('path:{pattern}', pattern, callback)

.removeListener('start', callback)
```

```
.removeListener('done', callback)
.removeListener('fail', callback)
```

Remove any listener on the Oboe instance.

From inside node and path callbacks .forget() is usually more convenient because it does not require that the programmer stores a reference to the callback function. However, `.removeListener()` has the advantage that it may be called from anywhere.

.abort()

Calling `.abort()` stops an ongoing HTTP call at any time. You are guaranteed not to get any further `path` or `node` callbacks, even if the underlying transport has unparsed buffered content. After calling `.abort()` the `done` event will not fire.

Under Node.js, if the Oboe instance is reading from a stream that it did not create this method deregisters all listeners but it is the caller's responsibility to actually terminate the streaming.

```
// Display the first nine nodes, then hang up
oboe('/content')
   .node('!.*', function(item){
      display(item);
   })
   .node('![9]', function(){
      this.abort();
   });
```

Pattern matching

Oboe's pattern matching is a variation on JSONPath. It supports these clauses:

Clause	Meaning
`!`	Root object
`.`	Path separator
`person`	An element under the key 'person'
`{name email}`	An element with attributes name and email
`*`	Any element at any name
`[2]`	The second element (of an array)

`['foo']`	Equivalent to .foo
`[*]`	Equivalent to .*
`$`	Explicitly specify an intermediate clause in the jsonpath spec the callback should be applied

The pattern engine supports CSS-4 style node selection using the dollar, `$`, symbol. See also the example patterns.

Browser support

These browsers have full support:

- Recent Chrome
- Recent Firefox
- Internet Explorer 10
- Recent Safaris

These browsers will run Oboe but not stream:

- Internet explorer 8 and 9, given appropriate shims for ECMAScript 5

Unfortunately, IE before version 10 doesn't provide any convenient way to read an http request while it is in progress.

The good news is that in older versions of IE Oboe gracefully degrades, it'll just fall back to waiting for the whole response to return, then fire all the events together. You don't get streaming but it isn't any worse than if you'd have designed your code to non-streaming AJAX.