# API

## The oboe object

Start a new AJAX request by calling one of these methods:

```
oboe( String url ) // makes a GET request

oboe({
   method: String,  // defaults to GET
   url: String,
   headers:{ key: value, ... },
   body: Object|String,
   cached: Boolean
})

// The oboe.doFoo() methods are deprecated
// and will be removed for oboe v2.0.0:
oboe.doGet(    String url )
oboe.doDelete( String url )
oboe.doPost(   String url, Object|String body )
oboe.doPut(    String url, Object|String body )
oboe.doPatch(  String url, Object|String body )

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

If the body is given as an object it will be serialised using JSON.stringify. Method, body and headers arguments are all optional.

If the cached option is set to false caching will be avoided by appending `_={timestamp}` to the URL's query string.

Under Node you can also give Oboe a ReadableStream:

```
oboe( ReadableStream source ) // Node.js only
```

When reading from a stream http headers and status code will not be available via the `start` event or the `.header()` method.

## Detecting nodes and paths

When you make a request the returned Oboe instance exposes a few chainable methods:

```
.node( String pattern,
       Function callback(node, String[] path, Object[] ancestors)
)

.on( 'node',
     String pattern,
     Function callback(node, String[] path, Object[] ancestors)
)

// 2-argument style .on() ala Node.js EventEmitter#on
.on( 'node:{pattern}',
     Function callback(node, String[] path, Object[] ancestors)
)
```

Listening for nodes registers an interest in JSON nodes which match the given pattern so that when the pattern is matched the callback is given the matching node. Inside the callback `this` will be the Oboe instance (unless you bound the callback)

The parameters to callback are:

| | |
|---|---|
| `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 location where the noc |
| `ancestors` | An array of node's ancestors. `ancestors[ancestors.length-1]` is the parent object, `ancestor` |

```
.path( String pattern,
       Function callback( thingFound, String[] path, Object[] ancestors)
)

.on( 'path',
     String pattern,
     Function callback(thingFound, String[] path, Object[] ancestors)
)

// 2-argument style .on() ala Node.js EventEmitter#on
.on( 'path:{pattern}',
     Function callback(thingFound, String[] path, Object[] ancestors)
)
```

`.path()` is the same as `.node()` except the callback is fired when we know about the matching *path*, before we know about the thing at the path.

Alternatively, several patterns may be registered at once using either `.path` or `.node`:

```
.node({
   pattern1 : Function callback,
   pattern2 : Function callback
});

.path({
   pattern3 : Function callback,
   pattern4 : Function callback
});
```

## .done()

```
.done(Function callback(Object wholeJson))

.on('done', Function callback(Object wholeJson))
```

Register a callback for when the response is complete. Gets passed the entire JSON. Usually it is better to read the json in small parts than waiting for it to completely download but this is there when you need the whole picture.

## .start()

```
.start(Function callback(Object json))

.on('start', Function callback(Number statusCode, Object headers))
```

Registers a listener for when the http response starts. When the callback is called we have the status code and the headers but no content yet.

## .header([name])

```
.header()

.header(name)
```

Get http response headers if we have received them yet. When a parameter name is given the named header will be returned, or undefined if it does not exist. When no name is given all headers will be returned as an Object. The headers will be available from inside `node`, `path`, `start` or `done` callbacks, or after any of those callbacks have been called. If have not recieved the headers yet `.header()` returns `undefined`.

The code below logs the ETag from the response before any other content is received.

```
oboe('/content')
   .on('start', function(item){
      console.log( 'content has tag', this.header('ETag') );
   })
```

## .root()

```
.root()
```

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

## .forget()

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

Calling .forget() on the Oboe instance from inside a node or path callback de-registers the currently executing callback.

The code below reads from an array at the root of the response but stops using the objects found after the tenth item:

```
oboe('/content')
   .on('node:!.*', function(item, path){

      if( path[0] == 9 ) {
         this.forget();
      }

      useItem(item);
   })
```

## .removeListener()

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

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

Remove a `node`, `path`, `start`, `done`, or `fail` listener. From inside the listener itself `.forget()` is usually more convenient but this works from anywhere.

## .abort()

`.abort()` Stops the http call at any time. This is useful if you want to read a json response only as far as is necessary. You are guaranteed not to get any further .path() or .node() callbacks, even if the underlying xhr already has additional content buffered and the .done() callback will not fire.

## .fail()

Fetching a resource could fail for several reasons:

- non-2xx status code
- connection lost
- invalid JSON from the server
- error thrown by a callback

```
.fail(Function callback(Object errorReport))

.on('fail', Function callback(Object errorReport))
```

An object is given to the callback with 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( errorReport.statusCode == 404 ){
         console.error('no such content');
      }
   });
```

## 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.