Javascript library to connect to swagger-enabled APIs via browser or nodejs http://swagger.io

Swagger JS library

build passing npm package 2.1.32

This is the Swagger javascript client for use with swagger enabled APIs. It's written in javascript and tested with mocha, and is the fastest way to enable a javascript client to communicate with a swagger-enabled server.

Check out Swagger-Spec for additional information about the Swagger project, including additional libraries with support for other languages and more.

Calling an API with swagger + node.js!

Install swagger-client:

```
npm install swagger-client
```

or:

```
bower install swagger-js
```

Then let swagger do the work!

```
var Swagger = require('swagger-client');
```

```
var client = new Swagger({
  url: 'http://petstore.swagger.io/v2/swagger.json',
  success: function() {
    client.pet.getPetById({petId:7},{responseContentType: 'application/json'},function(pet){
      console.log('pet', pet);
    });
  }
});
```

NOTE: we're explicitly setting the responseContentType, because we don't want you getting stuck when there is more than one content type available.

That's it! You'll get a JSON response with the default callback handler:

```
{
  "id": 1,
  "category": {
    "id": 2,
    "name": "Cats"
  },
  "name": "Cat 1",
  "photoUrls": [
    "url1",
    "url2"
  ],
  "tags": [
    {
      "id": 1,
      "name": "tag1"
    },
    {
      "id": 2,
      "name": "tag2"
    }
  ],
  "status": "available"
}
```

Handling success and failures

You need to pass success and error functions to do anything reasonable with the responses:

```
var Swagger = require('swagger-client');

var client = new Swagger({
  url: 'http://petstore.swagger.io/v2/swagger.json',
  success: function() {
    client.pet.getPetById({petId:7}, function(success){
      console.log('succeeded and returned this object: ' + success.obj);
    },
    function(error) {
      console.log('failed with the following: ' + error.statusText);
    });
  }
});
```

You can use promises, too, by passing the `usePromise: true` option:

```
var Swagger = require('swagger-client');

new Swagger({
  url: 'http://petstore.swagger.io/v2/swagger.json',
  usePromise: true
})
.then(function(client) {
  client.pet.getPetById({petId:7})
    .then(function(pet) {
      console.log(pet.obj);
    })
    .catch(function(error) {
      console.log('Oops!  failed with message: ' + error.statusText);
```

```
    });
});
```

Authorization

Need to pass an API key? Ok, lets do it for this sample `swagger.yml`:

```
# ...

securityDefinitions:

  api_scheme_name:           # swagger scheme name
    type: apiKey            # swagger type (one of "basic", "apiKey" or "oauth2")
    name: queryParamName    # The name of the header or query parameter to be used
    in: query               # location of the API key

  api_scheme_name_2:
    type: apiKey
    name: X-KEY-PARAM
    in: header

# ...
```

Configure auth for that definition in your client instance as a *query string*:

```
client.clientAuthorizations.add("api_scheme_name",
  new Swagger.ApiKeyAuthorization(
    "queryParamName",
    "<YOUR-SECRET-KEY>",
    "query"
  )
);
```

...or with a *header*:

```
client.clientAuthorizations.add("api_scheme_name_2",
  new Swagger.ApiKeyAuthorization(
    "X-KEY-PARAM",
    "<YOUR-SECRET-KEY>",
    "header"
  )
);
```

...or with the swagger-client constructor:

```
var client = new Swagger({
  url: 'http://example.com/spec.json',
  success: function() {},
  authorizations : {
    easyapi_basic: new Swagger.PasswordAuthorization('<username>', '<password>'),
    someHeaderAuth: new Swagger.ApiKeyAuthorization('<nameOfHeader>', '<value>', 'header'),
    someQueryAuth: new Swagger.ApiKeyAuthorization('<nameOfQueryKey>', '<value>', 'query'),
    someCookieAuth: new Swagger.CookieAuthorization('<cookie>'),
  }
});
```

Note the authorization nickname, such as `easyapi_basic` in the above example, must match the `security` requirement in the specification (see the OAI Specification for details).

You can also pass authorzations on a *per-request* basis, in the event that you're reusing a `swagger-client` object across multiple connections:

```
client.pet.addPet({pet: {
    name: 'doggie'
  }}, {
    clientAuthorizations: {
      api_key: new Swagger.ApiKeyAuthorization('foo', 'bar', 'header')
    }
```

```
  })
  .then(function(pet) {
    console.log(pet.obj);
  });
```

Calling an API with swagger + the browser!

Download `browser/swagger-client.min.js` and place it into your webapp:

```
<script src='browser/swagger-client.js' type='text/javascript'></script>
<script type="text/javascript">
  // initialize swagger client, point to a resource listing
  window.client = new SwaggerClient({
    url: "http://petstore.swagger.io/v2/swagger.json",
    success: function() {
      // upon connect, fetch a pet and set contents to element "mydata"
      client.pet.getPetById({petId:1},{responseContentType: 'application/json'}, function(data) {
        document.getElementById("mydata").innerHTML = JSON.stringify(data.obj);
      });
    }
  });
</script>

<body>
  <div id="mydata"></div>
</body>
```

Need to send an object to your API via POST or PUT?

```
var pet = {
  id: 100,
  name: "dog"};

// note: the parameter for `addPet` is named `body` in the example below
client.pet.addPet({body: pet});
```

Sending XML in as a payload to your API?

```
var pet = "<Pet><id>2</id><name>monster</name></Pet>";

client.pet.addPet({body: pet}, {requestContentType:"application/xml"});
```

Need XML response? (assuming your server can produce it)

```
client.pet.getPetById({petId:1}, {responseContentType:"application/xml"});
```

Custom request signing

You can easily write your own request signing code for Swagger. For example:

```
var CustomRequestSigner = function(name) {
  this.name = name;
};

CustomRequestSigner.prototype.apply = function(obj, authorizations) {
  var hashFunction = this._btoa;
  var hash = hashFunction(obj.url);

  obj.headers["signature"] = hash;
  return true;
};
```

In the above simple example, we're creating a new request signer that simply Base64 encodes the URL. Of course you'd do something more sophisticated, but after encoding it, a header called `signature` is set before sending the request.

You can add it to the swagger-client like such:

```
client.clientAuthorizations.add('my-auth', new CustomRequestSigner());
```

Setting headers

Headers are a type of `parameter`, and can be passed with the other parameters. For example, if you supported translated pet details via the `Accept-Language` header:

```
"parameters": [
  {
    "name": "petId",
    "description": "ID of pet that needs to be fetched",
    "required": true,
    "type": "integer",
    "format": "int64",
    "paramType": "path",
    "minimum": "1.0",
    "defaultValue": 3,
    "maximum": "100000.0"
  },
  "LanguageHeader": {
    "name": "Accept-Language",
    "in": "header",
    "description": "Specify the user's language",
    "required": false,
    "type": "string"
  }
...
```

Then you would pass the header value via the parameters (header parameters are case-insenstive):

```
client.pet.getPetById({
  petId: 7,
  'accept-language': 'fr'
}, function(pet){
  console.log('pet', pet);
});
```

Using your own HTTP client

Don't like superagent? Despise JQuery? Well, you're in luck. You can plug your own HTTP library easily:

```
var myHttpClient = {
  // implment an execute function
  execute: function(obj) {
    var httpMethod = obj.method;
    var requestHeaders = obj.headers;
    var body = obj.body;
    var url = obj.url;
    // do your thing, and call `obj.on.response`
    if(itWorked) {
      obj.on.response('horray');
    }
    else {
      obj.on.error('boo');
    }
  }
};

var client = new SwaggerClient({
  spec: petstoreRaw,
  client: myHttpClient,
  success: function () {
    client.pet.getPetById({petId: 3}, function(data){
      expect(data).toBe('ok');
      done();
    });
  }
});
```

You can also pass in your own version superagent (if, for example, you have other superagent plugins etc that you want to use)

```
var agent = require('some-other-special-superagent');

var client = new SwaggerClient({
  spec: petstoreRaw,
  requestAgent: agent,
  success: function () {
    client.pet.getPetById({petId: 3}, function(data){
      expect(data).toBe('ok');
      done();
    });
  }
});
```

Using custom http(s) agent

In case if you need to sign all requests to petstore with custom certificate

```
var connectionAgent = {
    rejectUnauthorized: false,
    key: "/certs/example.key",
    cert: "/certs/example.pem",
    ca: ["/certs/example.ca.pem"]
}

var client = new SwaggerClient({
  url: "http://petstore.swagger.io/v2/swagger.json",
  connectionAgent: connectionAgent,
  success: function() {
    // upon connect, fetch a pet and set contents to element "mydata"
    client.pet.getPetById({petId:1},{responseContentType: 'application/json'}, function(data) {
      document.getElementById("mydata").innerHTML = JSON.stringify(data.obj);
    });
  }
});
```

How does it work?

The swagger javascript client reads the swagger api definition directly from the server. As it does, it constructs a client based on the api definition, which means it is completely dynamic. It even reads the api text descriptions (which are intended for humans!) and provides help if you need it:

```
s.apis.pet.getPetById.help()
'* petId (required) - ID of pet that needs to be fetched'
```

The HTTP requests themselves are handled by the excellent superagent library, which has a ton of features itself. But it runs on both node and the browser.

Development

Please fork the code and help us improve swagger-js. Send us a pull request to the `master` branch! Tests make merges get accepted more quickly.

Note! We *will not* merge pull requests for features not supported in the OAI Specification! Add an issue there instead!

swagger-js use gulp for Node.js.

```
# Install the gulp client on the path
npm install -g gulp

# Install all project dependencies
npm install
```

```
# List all tasks.
gulp -T

# Run lint (will not fail if there are errors/warnings), tests (without coverage) and builds the browser binaries
gulp

# Run the test suite (without coverage)
gulp test

# Build the browser binaries (One for development with source maps and one that is minified and without source maps)
gulp build

# Continuously run the test suite:
gulp watch

# Run jshint report
gulp lint

# Run a coverage report based on running the unit tests
gulp coverage
```

License