oauth2-server Documentation

Release 3.0.2

Max Truxa <dev@maxtruxa.com>

May 24, 2020

Contents

oauth2-server is a complete, compliant and well tested module for implementing an OAuth2 server in Node.js. The project is hosted on GitHub and the included test suite is automatically run on Travis CI.

Installation

CHAPTER 1

Example Usage

```
const OAuth2Server = require('oauth2-server');
const Request = OAuth2Server.Request;
const Response = OAuth2Server.Response;

const oauth = new OAuth2Server({
  model: require('./model')
});

let request = new Request({
  method: 'GET',
  query: {},
  headers: {Authorization: 'Bearer foobar'}
});

let response = new Response({
  headers: {}
});

oauth.authenticate(request, response)
  .then((token) => {
    // The request was successfully authenticated.
  })
  .catch((err) => {
    // The request failed authentication.
  });
```

See the *Model Specification* of what is required from the model passed to *OAuth2Server*.

1.1 Getting Started

1.1.1 Installation

oauth2-server is available via npm.

```
$ npm install oauth2-server
```

Note: The *oauth2-server* module is framework-agnostic but there are several officially supported adapters available for popular HTTP server frameworks such as Express and Koa. If you're using one of those frameworks it is strongly recommended to use the respective adapter module instead of rolling your own.

1.1.2 Features

- Supports *authorization code*, *client credentials*, *refresh token* and *password* grant, as well as *extension grants*, with scopes.
- Can be used with *promises*, *Node-style callbacks*, *ES6 generators* and *async/await* (using Babel).
- Fully **RFC 6749** and **RFC 6750** compliant.
- Implicitly supports any form of storage, e.g. *PostgreSQL*, *MySQL*, *MongoDB*, *Redis*, etc.
- Complete test suite.

1.1.3 Quick Start

OAuth2Server

```
const OAuth2Server = require('oauth2-server');

const oauth = new OAuth2Server({
  model: require('./model')
});
```

Request and *Response*

```
const Request = OAuth2Server.Request;
const Response = OAuth2Server.Response;

let request = new Request({/*...*/});
let response = new Response({/*...*/});
```

OAuth2Server#authenticate()

```
oauth.authenticate(request, response)
  .then((token) => {
    // The request was successfully authenticated.
  })
  .catch((err) => {
    // The request failed authentication.
  });
```

OAuth2Server#authorize()

```
const AccessDeniedError = require('oauth2-server/lib/errors/access-denied-error');

oauth.authorize(request, response)
  .then((code) => {
    // The resource owner granted the access request.
  })
  .catch((err) => {
    if (err instanceof AccessDeniedError) {
      // The resource owner denied the access request.
    } else {
      // Access was not granted due to some other error condition.
    }
  });
```

OAuth2Server#token()

```
oauth.token(request, response)
  .then((token) => {
    // The resource owner granted the access request.
  })
  .catch((err) => {
    // The request was invalid or not authorized.
  });
```

1.2 Adapters

The *oauth2-server* module is typically not used directly but through one of the available adapters, converting the interface to a suitable one for the HTTP server framework in use.

- express-oauth-server for Express
- koa-oauth-server for Koa

1.2.1 Writing Adapters

Adapters typically do the following:

- Inherit from *OAuth2Server*.
- Override *authenticate()*, *authorize()* and *token()*.

  Each of these functions should:

  - Create *Request* and *Response* objects from their framework-specific counterparts.
  - Call the original function.
  - Copy all fields from the *Response* back to the framework-specific request object and send it.

Adapters should preserve functionality provided by *oauth2-server* but are free to add additional features that make sense for the respective HTTP server framework.

1.3 OAuth2Server

Represents an OAuth2 server instance.

```
const OAuth2Server = require('oauth2-server');
```

1.3.1 new OAuth2Server(options)

Instantiates `OAuth2Server` using the supplied model.

Arguments:

Name	Type	Description
options	Object	Server options.
options.model	Object	The *Model*.

Return value:

A new `OAuth2Server` instance.

Remarks:

Any valid option for *OAuth2Server#authenticate()*, *OAuth2Server#authorize()* and *OAuth2Server#token()* can be passed to the constructor as well. The supplied options will be used as default for the other methods.

Basic usage:

```
const oauth = new OAuth2Server({
  model: require('./model')
});
```

Advanced example with additional options:

```
const oauth = new OAuth2Server({
  model: require('./model'),
  allowBearerTokensInQueryString: true,
  accessTokenLifetime: 4 * 60 * 60
});
```

1.3.2 authenticate(request, response, [options], [callback])

Authenticates a request.

Arguments:

Name	Type	Description
request	*Request*	Request object.
response	*Response*	Response object.
[options={}]	Object	Handler options.
[options.scope=undefined]	String	The scope(s) to authenticate.
[options.addAcceptedScopesHeader=true]	Boolean	Set the `X-Accepted-OAuth-Scopes` HTTP header on response objects.
[options.addAuthorizedScopesHeader=true]	Boolean	Set the `X-OAuth-Scopes` HTTP header on response objects.
[options.allowBearerTokensInQueryString=false]	Boolean	Allow clients to pass bearer tokens in the query string of a request.
[callback=undefined]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

A `Promise` that resolves to the access token object returned from *Model#getAccessToken()*. In case of an error, the promise rejects with one of the error types derived from *OAuthError*.

Possible errors include but are not limited to:

***UnauthorizedRequestError*:** The protected resource request failed authentication.

The returned `Promise` **must** be ignored if `callback` is used.

Remarks:

```
const oauth = new OAuth2Server({model: ...});

function authenticateHandler(options) {
  return function(req, res, next) {
    let request = new Request(req);
    let response = new Response(res);
    return oauth.authenticate(request, response, options)
      .then(function(token) {
        res.locals.oauth = {token: token};
        next();
      })
      .catch(function(err) {
        // handle error condition
      });
  }
}
```

1.3.3 `authorize(request, response, [options], [callback])`

Authorizes a token request.

Arguments:

Name	Type	Description
request	*Request*	Request object.
[request.query.allowed=undefined]	String	`'false'` to deny the authorization request (see remarks section).
response	*Response*	Response object.
[options={}]	Object	Handler options.
[options.authenticateHandler=undefined]	Object	The authenticate handler (see remarks section).
[options.allowEmptyState=false]	Boolean	Allow clients to specify an empty `state`.
[options.authorizationCodeLifetime=300]	Number	Lifetime of generated authorization codes in seconds (default = 5 minutes).
[callback=undefined]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

A `Promise` that resolves to the authorization code object returned from *Model#saveAuthorizationCode()*. In case of an error, the promise rejects with one of the error types derived from *OAuthError*.

Possible errors include but are not limited to:

AccessDeniedError The resource owner denied the access request (i.e. `request.query.allow` was `'false'`).

The returned `Promise` **must** be ignored if `callback` is used.

Remarks:

If `request.query.allowed` equals the string `'false'` the access request is denied and the returned promise is rejected with an *AccessDeniedError*.

In order to retrieve the user associated with the request, `options.authenticateHandler` should be supplied. The `authenticateHandler` has to be an object implementing a `handle(request, response)` function that returns a user object. If there is no associated user (i.e. the user is not logged in) a falsy value should be returned.

```
let authenticateHandler = {
  handle: function(request, response) {
    return /* get authenticated user */;
  }
};
```

When working with a session-based login mechanism, the handler can simply look like this:

```
let authenticateHandler = {
  handle: function(request, response) {
    return request.session.user;
  }
};
```

Todo: Move `authenticateHandler` to it's own section.

```
const oauth = new OAuth2Server({model: ...});

function authorizeHandler(options) {
  return function(req, res, next) {
```

(continues on next page)

(continued from previous page)

```
    let request = new Request(req);
    let response = new Response(res);
    return oauth.authorize(request, response, options)
      .then(function(code) {
        res.locals.oauth = {code: code};
        next();
      })
      .catch(function(err) {
        // handle error condition
      });
  }
}
```

1.3.4 token(request, response, [options], [callback])

Retrieves a new token for an authorized token request.

Arguments:

Name	Type	Description
request	*Request*	Request object.
response	*Response*	Response object.
[options={}]	Object	Handler options.
[options.accessTokenLifetime=3600]	Number	Lifetime of generated access tokens in seconds (default = 1 hour).
[options.refreshTokenLifetime=1209600]	Number	Lifetime of generated refresh tokens in seconds (default = 2 weeks).
[options.allowExtendedTokenAttributes=false]	Boolean	Allow extended attributes to be set on the returned token (see remarks section).
[options.requireClientAuthentication={}]	Object	Require a client secret (see remarks section). Defaults to `true` for all grant types.
[options.alwaysIssueNewRefreshToken=true]	Boolean	Always revoke the used refresh token and issue a new one for the `refresh_token` grant.
[options.extendedGrantTypes={}]	Object	Additional supported grant types.
[callback=undefined]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

A `Promise` that resolves to the token object returned from *Model#saveToken()*. In case of an error, the promise rejects with one of the error types derived from *OAuthError*.

Possible errors include but are not limited to:

***InvalidGrantError*:** The access token request was invalid or not authorized.

The returned `Promise` **must** be ignored if `callback` is used.

Remarks:

If `options.allowExtendedTokenAttributes` is `true` any additional properties set on the object returned from *Model#saveToken()* are copied to the token response sent to the client.

By default all grant types require the client to send it's `client_secret` with the token request. `options.requireClientAuthentication` can be used to disable this check for selected grants. If used, this server option must be an object containing properties set to `true` or `false`. Possible keys for the object include all supported values for the token request's `grant_type` field (`authorization_code`, `client_credentials`, `password` and `refresh_token`). Grants that are not specified default to `true` which enables verification of the `client_secret`.

```
let options = {
  // ...
  // Allow token requests using the password grant to not include a client_secret.
  requireClientAuthentication: {password: false}
};
```

`options.extendedGrantTypes` is an object mapping extension grant URIs to handler types, for example:

```
let options = {
  // ...
  extendedGrantTypes: {
    'urn:foo:bar:baz': MyGrantType
  }
};
```

For information on how to implement a handler for a custom grant type see *Extension Grants*.

```
const oauth = new OAuth2Server({model: ...});

function tokenHandler(options) {
  return function(req, res, next) {
    let request = new Request(req);
    let response = new Response(res);
    return oauth.token(request, response, options)
      .then(function(code) {
        res.locals.oauth = {token: token};
        next();
      })
      .catch(function(err) {
        // handle error condition
      });
  }
}
```

1.4 Request

Represents an incoming HTTP request.

```
const Request = require('oauth2-server').Request;
```

1.4.1 `new Request(options)`

Instantiates `Request` using the supplied options.

Arguments:

Name	Type	Description
options	Object	Request options.
options.method	String	The HTTP method of the request.
options.query	Object	The request's query string parameters.
options.headers	Object	The request's HTTP header fields.
[options.body={}]	Object	Key-value pairs of data submitted in the request body.

All additional own properties are copied to the new `Request` object as well.

Return value:

A new `Request` instance.

Remarks:

The names of HTTP header fields passed in as `options.headers` are converted to lower case.

To convert Express' request to a `Request` simply pass `req` as `options`:

```
function(req, res, next) {
  var request = new Request(req);
  // ...
}
```

1.4.2 `get(field)`

Returns the specified HTTP header field. The match is case-insensitive.

Arguments:

Name	Type	Description
field	String	The header field name.

Return value:

The value of the header field or `undefined` if the field does not exist.

1.4.3 `is(types)`

Checks if the request's `Content-Type` HTTP header matches any of the given MIME types.

Arguments:

Name	Type	Description
types	Array<String>|String	The MIME type(s) to test against.

Return value:

Returns the matching MIME type or `false` if there was no match.

1.4.4 method

The HTTP method of the request (`'GET'`, `'POST'`, `'PUT'`, ...).

1.4.5 query

The request's query string parameters.

1.4.6 headers

The request's HTTP header fields. Prefer *Request#get()* over accessing this object directly.

1.4.7 body

Key-value pairs of data submitted in the request body.

1.5 Response

Represents an outgoing HTTP response.

```
const Response = require('oauth2-server').Response;
```

1.5.1 new Response(options)

Instantiates `Response` using the supplied options.

Arguments:

Name	Type	Description
options	Object	Response options.
options.headers	Object	The response's HTTP header fields.
[options.body={}]	Object	Key-value pairs of data to be submitted in the response body.

All additional own properties are copied to the new `Response` object as well.

Return value:

A new `Response` instance.

Remarks:

The names of HTTP header fields passed in as `options.headers` are converted to lower case.

To convert Express' response to a `Response` simply pass `res` as `options`:

```
function(req, res, next) {
  var response = new Response(res);
  // ...
}
```

1.5.2 get(field)

Returns the specified HTTP header field. The match is case-insensitive.

Arguments:

Name	Type	Description
field	String	The header field name.

Return value:

The value of the header field or `undefined` if the field does not exist.

1.5.3 set(field, value)

Sets the specified HTTP header field. The match is case-insensitive.

Arguments:

Name	Type	Description
field	String	The header field name.
value	String	The header field value.

Return value:

None.

1.5.4 redirect(url)

Redirects to the specified URL using `302 Found`.

Arguments:

Name	Type	Description
url	String	The URL to redirect to.

Return value:

None.

Remarks:

This is essentially a convenience function that sets `status` to `302` and the `Location` header to the provided URL.

1.5.5 status

The HTTP status of the response (default = 200).

1.5.6 headers

The response's HTTP header fields. Prefer *Response#get()*/*Response#set()* over accessing this object directly.

1.5.7 body

Key-value pairs of data to be submitted in the response body.

1.6 Errors

Noteable error types:

OAuthError *OAuthError* is the base class for all exceptions thrown/returned by *oauth2-server*.

ServerError *ServerError* is used to wrap unknown exceptions encountered during request processing.

InvalidArgumentError *InvalidArgumentError* is thrown when an invalid argument is encountered. This error indicates that the module is used incorrectly and should never be seen because of external errors.

1.6.1 OAuthError

Base class for all errors returned by this module.

```
const OAuthError = require('oauth2-server/lib/errors/oauth-error');
```

---

new OAuthError(message, properties)

Instantiates OAuthError.

Note: Do not use OAuthError directly; it's intended to be used as a base class. Instead, use one of the other error types derived from it.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	Error message or nested exception.
[properties={}]	Object	Additional properties to be set on the error object.
[properties.code=500]	Object	An HTTP status code associated with the error.
[properties.name=undefined]	String	The name of the error. If left undefined, the name is copied from the constructor.

Return value:

A new instance of `OAuthError`.

Remarks:

By default `code` is set to `500` and `message` is set to the respective HTTP phrase.

```
const err = new OAuthError();
// err.message === 'Internal Server Error'
// err.code === 500
// err.name === 'OAuthError'
```

```
const err = new OAuthError('test', {name: 'test_error'});
// err.message === 'test'
// err.code === 500
// err.name === 'test_error'
```

```
const err = new OAuthError(undefined, {code: 404});
// err.message === 'Not Found'
// err.code === 404
// err.name === 'OAuthError'
```

All additional `properties` are copied to the error object.

```
const err = new OAuthError('test', {foo: 'bar', baz: 1234});
// err.message === 'test'
// err.code === 500
// err.name === 'OAuthError'
// err.foo === 'bar'
// err.baz === 1234
```

When wrapping an exception, the `message` property is automatically copied from the existing exception.

```
const anotherError = new Error('test');
const err = new OAuthError(e);
// err.message === 'test'
// err.code === 500
// err.name === 'OAuthError'
// err.inner === anotherError
```

---

message

A message describing the error.

---

code

An HTTP status code associated with the error.

For compatibility reasons, two more properties exist that have the same value as `code`: `status` and `statusCode`. Note that changes to one of these are not reflected by the other properties.

---

inner

Another exception that was wrapped by this `OAuthError` instance. This property is set only if the error is constructed from an existing exception.

---

name

The name of the error, intended to be used as the `error` parameter as described by **RFC 6749** in **Section 4.1.2.1**, **Section 4.2.2.1** and **Section 5.2**, as well as **Section 3.1 of RFC 6750**.

1.6.2 ServerError

The authorization server encountered an unexpected condition that prevented it from fulfilling the request. See **Section 4.1.2.1 of RFC 6749**.

```
const ServerError = require('oauth2-server/lib/errors/server-error');
```

`ServerError` is used to wrap unknown exceptions encountered during request processing.

---

new ServerError(message, properties)

Instantiates an `ServerError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=503]	Object	See *new OAuthError(message, properties)*.
[properties.name='server_error']	String	The error name used in responses generated from this error.

Return value:

A new instance of `ServerError`.

Remarks:

```
const err = new ServerError();
// err.message === 'Service Unavailable Error'
// err.code === 503
// err.name === 'server_error'
```

---

message

See *OAuthError#message*.

---

code

Typically `503`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'server_error'`. See *OAuthError#name*.

1.6.3 InvalidArgumentError

An invalid argument was encountered.

```
const InvalidArgumentError = require('oauth2-server/lib/errors/invalid-argument-error
↪');
```

Note: This error indicates that the module is used incorrectly (i.e., there is a programming error) and should never be seen because of external errors (like invalid data sent by a client).

---

new InvalidArgumentError(message, properties)

Instantiates an `InvalidArgumentError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=500]	Object	See *new OAuthError(message, properties)*.
[properties.name='invalid_argument']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InvalidArgumentError`.

Remarks:

```
const err = new InvalidArgumentError();
// err.message === 'Internal Server Error'
// err.code === 500
// err.name === 'invalid_argument'
```

---

message

See *OAuthError#message*.

---

code

Typically `500`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'invalid_argument'`. See *OAuthError#name*.

1.6.4 AccessDeniedError

The resource owner or authorization server denied the request. See **Section 4.1.2.1 of RFC 6749**.

```
const AccessDeniedError = require('oauth2-server/lib/errors/access-denied-error');
```

---

new AccessDeniedError(message, properties)

Instantiates an `AccessDeniedError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='access_denied']	String	The error name used in responses generated from this error.

Return value:

A new instance of `AccessDeniedError`.

Remarks:

```
const err = new AccessDeniedError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'access_denied'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'access_denied'`. See *OAuthError#name*.

1.6.5 InsufficientScopeError

The request requires higher privileges than provided by the access token. See **Section 3.1 of RFC 6750**.

```
const InsufficientScopeError = require('oauth2-server/lib/errors/insufficient-scope-
↪error');
```

---

new InsufficientScopeError(message, properties)

Instantiates an `InsufficientScopeError`.

Arguments:

Name	Type	Description
[message=undefined]	StringlError	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=403]	Object	See *new OAuthError(message, properties)*.
[properties.name='insufficient_scope']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InsufficientScopeError`.

Remarks:

```
const err = new InsufficientScopeError();
// err.message === 'Forbidden'
// err.code === 403
// err.name === 'insufficient_scope'
```

---

message

See *OAuthError#message*.

---

code

Typically `403`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'insufficient_scope'`. See *OAuthError#name*.

1.6.6 InvalidClientError

Client authentication failed (e.g., unknown client, no client authentication included, or unsupported authentication method). See **Section 5.2 of RFC 6749**.

```
const InvalidClientError = require('oauth2-server/lib/errors/invalid-client-error');
```

---

new InvalidClientError(message, properties)

Instantiates an `InvalidClientError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='invalid_client']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InvalidClientError`.

Remarks:

```
const err = new InvalidClientError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'invalid_client'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'invalid_client'`. See *OAuthError#name*.

1.6.7 InvalidGrantError

The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client. See **Section 5.2 of RFC 6749**.

```
const InvalidGrantError = require('oauth2-server/lib/errors/invalid-grant-error');
```

---

new InvalidGrantError(message, properties)

Instantiates an `InvalidGrantError`.

Arguments:

Name	Type	Description
[message=undefined]	StringlError	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='invalid_grant']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InvalidGrantError`.

Remarks:

```
const err = new InvalidGrantError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'invalid_grant'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'invalid_grant'`. See *OAuthError#name*.

1.6.8 InvalidRequestError

The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed. See **Section 4.2.2.1 of RFC 6749**.

```
const InvalidRequestError = require('oauth2-server/lib/errors/invalid-request-error');
```

---

new InvalidRequestError(message, properties)

Instantiates an `InvalidRequestError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='invalid_request']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InvalidRequestError`.

Remarks:

```
const err = new InvalidRequestError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'invalid_request'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'invalid_request'`. See *OAuthError#name*.

1.6.9 InvalidScopeError

The requested scope is invalid, unknown, or malformed. See **Section 4.1.2.1 of RFC 6749**.

```
const InvalidScopeError = require('oauth2-server/lib/errors/invalid-scope-error');
```

---

new InvalidScopeError(message, properties)

Instantiates an `InvalidScopeError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='invalid_scope']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InvalidScopeError`.

Remarks:

```
const err = new InvalidScopeError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'invalid_scope'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'invalid_scope'`. See *OAuthError#name*.

1.6.10 InvalidTokenError

The access token provided is expired, revoked, malformed, or invalid for other reasons. See **Section 3.1 of RFC 6750**.

```
const InvalidTokenError = require('oauth2-server/lib/errors/invalid-token-error');
```

---

new InvalidTokenError(message, properties)

Instantiates an `InvalidTokenError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=401]	Object	See *new OAuthError(message, properties)*.
[properties.name='invalid_token']	String	The error name used in responses generated from this error.

Return value:

A new instance of `InvalidTokenError`.

Remarks:

```
const err = new InvalidTokenError();
// err.message === 'Unauthorized'
// err.code === 401
// err.name === 'invalid_token'
```

---

message

See *OAuthError#message*.

---

code

Typically `401`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'invalid_token'`. See *OAuthError#name*.

1.6.11 UnauthorizedClientError

The authenticated client is not authorized to use this authorization grant type. See **Section 4.1.2.1 of RFC 6749**.

```
const UnauthorizedClientError = require('oauth2-server/lib/errors/unauthorized-client-
↪error');
```

---

new UnauthorizedClientError(message, properties)

Instantiates an `UnauthorizedClientError`.

Arguments:

Name	Type	Description
[message=undefined]	StringlError	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='unauthorized_client']	String	The error name used in responses generated from this error.

Return value:

A new instance of `UnauthorizedClientError`.

Remarks:

```
const err = new UnauthorizedClientError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'unauthorized_client'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'unauthorized_client'`. See *OAuthError#name*.

1.6.12 UnauthorizedRequestError

The request lacked any authentication information or the client attempted to use an unsupported authentication method.

```
const UnauthorizedRequestError = require('oauth2-server/lib/errors/unauthorized-
↪request-error');
```

According to **Section 3.1 of RFC 6750** you should just fail the request with `401 Unauthorized` and not send any error information in the body if this error occurs:

> If the request lacks any authentication information (e.g., the client was unaware that authentication is necessary or attempted using an unsupported authentication method), the resource server SHOULD NOT include an error code or other error information.

---

new UnauthorizedRequestError(message, properties)

Instantiates an `UnauthorizedRequestError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=401]	Object	See *new OAuthError(message, properties)*.
[properties.name='unauthorized_request']	String	The error name used in responses generated from this error.

Return value:

A new instance of `UnauthorizedRequestError`.

Remarks:

```
const err = new UnauthorizedRequestError();
// err.message === 'Unauthorized'
// err.code === 401
// err.name === 'unauthorized_request'
```

---

message

See *OAuthError#message*.

---

code

Typically `401`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'unauthorized_request'`. See *OAuthError#name*.

1.6.13 UnsupportedGrantTypeError

The authorization grant type is not supported by the authorization server. See **Section 4.1.2.1 of RFC 6749**.

```
const UnsupportedGrantTypeError = require('oauth2-server/lib/errors/unsupported-grant-
↪type-error');
```

---

new UnsupportedGrantTypeError(message, properties)

Instantiates an `UnsupportedGrantTypeError`.

Arguments:

Name	Type	Description
[message=undefined]	StringlError	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='unsupported_grant_type']	String	The error name used in responses generated from this error.

Return value:

A new instance of `UnsupportedGrantTypeError`.

Remarks:

```
const err = new UnsupportedGrantTypeError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'unsupported_grant_type'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'unsupported_grant_type'`. See *OAuthError#name*.

1.6.14 UnsupportedResponseTypeError

The authorization server does not supported obtaining an authorization code using this method. See **Section 4.1.2.1 of RFC 6749**.

```
const UnsupportedResponseTypeError = require('oauth2-server/lib/errors/unsupported-
↪response-type-error');
```

---

new UnsupportedResponseTypeError(message, properties)

Instantiates an `UnsupportedResponseTypeError`.

Arguments:

Name	Type	Description
[message=undefined]	String|Error	See *new OAuthError(message, properties)*.
[properties={}]	Object	See *new OAuthError(message, properties)*.
[properties.code=400]	Object	See *new OAuthError(message, properties)*.
[properties.name='unsupported_response_type']	String	The error name used in responses generated from this error.

Return value:

A new instance of `UnsupportedResponseTypeError`.

Remarks:

```
const err = new UnsupportedResponseTypeError();
// err.message === 'Bad Request'
// err.code === 400
// err.name === 'unsupported_response_type'
```

---

message

See *OAuthError#message*.

---

code

Typically `400`. See *OAuthError#code*.

---

inner

See *OAuthError#inner*.

---

name

Typically `'unsupported_response_type'`. See *OAuthError#name*.

1.7 Model Overview

OAuth2Server requires a model object through which some aspects of storage, retrieval and custom validation are abstracted.

1.7.1 Grant Types

RFC 6749 describes a number of grants for a client application to acquire an access token.

The following grant types are supported by *oauth2-server*:

Authorization Code Grant

See **Section 4.1 of RFC 6749**.

An authorization code is a credential representing the resource owner's authorization (to access its protected resources) which is used by the client to obtain an access token.

Model functions used by the authorization code grant:

- *generateAccessToken(client, user, scope, [callback])*
- *generateRefreshToken(client, user, scope, [callback])*
- *generateAuthorizationCode(client, user, scope, [callback])*
- *getAuthorizationCode(authorizationCode, [callback])*
- *getClient(clientId, clientSecret, [callback])*
- *saveToken(token, client, user, [callback])*
- *saveAuthorizationCode(code, client, user, [callback])*
- *revokeAuthorizationCode(code, [callback])*
- *validateScope(user, client, scope, [callback])*

Client Credentials Grant

See **Section 4.4 of RFC 6749**.

The client can request an access token using only its client credentials (or other supported means of authentication) when requesting access to the protected resources under its control.

Note: The client credentials grant type **must** only be used by confidential clients.

Model functions used by the client credentials grant:

- *generateAccessToken(client, user, scope, [callback])*
- *getClient(clientId, clientSecret, [callback])*
- *getUserFromClient(client, [callback])*
- *saveToken(token, client, user, [callback])*
- *validateScope(user, client, scope, [callback])*

Refresh Token Grant

See **Section 6 of RFC 6749**.

If the authorization server issued a refresh token to the client, the client can request a refresh of their authorization token.

Model functions used by the refresh token grant:

- *generateRefreshToken(client, user, scope, [callback])*

- *getRefreshToken(refreshToken, [callback])*
- *getClient(clientId, clientSecret, [callback])*
- *saveToken(token, client, user, [callback])*
- *revokeToken(token, [callback])*

Password Grant

See **Section 4.3 of RFC 6749**.

The password grant is suitable for clients capable of obtaining the resource owner's credentials (username and password, typically using an interactive form).

Model functions used by the password grant:

- *generateAccessToken(client, user, scope, [callback])*
- *generateRefreshToken(client, user, scope, [callback])*
- *getClient(clientId, clientSecret, [callback])*
- *getUser(username, password, [callback])*
- *saveToken(token, client, user, [callback])*
- *validateScope(user, client, scope, [callback])*

Extension Grants

See **Section 4.5 of RFC 6749**.

The authorization server may also implement custom grant types to issue access (and optionally refresh) tokens.

See *Extension Grants*.

1.7.2 Request Authentication

See **Section 2 of RFC 6750**.

The authorization server authenticates requests sent to the resource server by verifying the included bearer token.

Model functions used during request authentication:

- *getAccessToken(accessToken, [callback])*
- *verifyScope(accessToken, scope, [callback])*

1.8 Model Specification

Each model function supports *promises*, *Node-style callbacks*, *ES6 generators* and *async/await* (using Babel). Note that promise support implies support for returning plain values where asynchronism is not required.

```
const model = {
  // We support returning promises.
  getAccessToken: function() {
    return new Promise('works!');
  },

  // Or, calling a Node-style callback.
  getAuthorizationCode: function(done) {
    done(null, 'works!');
  },

  // Or, using generators.
  getClient: function*() {
    yield somethingAsync();
    return 'works!';
  },

  // Or, async/wait (using Babel).
  getUser: async function() {
    await somethingAsync();
    return 'works!';
  }
};

const OAuth2Server = require('oauth2-server');
let oauth = new OAuth2Server({model: model});
```

Code examples on this page use *promises*.

1.8.1 `generateAccessToken(client, user, scope, [callback])`

Invoked to generate a new access token.

This model function is **optional**. If not implemented, a default handler is used that generates access tokens consisting of 40 characters in the range of `a..z0..9`.

Invoked during:

- `authorization_code` grant
- `client_credentials` grant
- `refresh_token` grant
- `password` grant

Arguments:

Name	Type	Description
client	Object	The client the access token is generated for.
user	Object	The user the access token is generated for.
scope	String	The scopes associated with the access token. Can be `null`.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

A `String` to be used as access token.

RFC 6749 Appendix A.12 specifies that access tokens must consist of characters inside the range `0x20..0x7E` (i.e. only printable US-ASCII characters).

Remarks:

`client` is the object previously obtained through *Model#getClient()*.

`user` is the user object previously obtained through *Model#getAuthorizationCode()* (`code.user`; authorization code grant), *Model#getUserFromClient()* (client credentials grant), *Model#getRefreshToken()* (`token.user`; refresh token grant) or *Model#getUser()* (password grant).

1.8.2 `generateRefreshToken(client, user, scope, [callback])`

Invoked to generate a new refresh token.

This model function is **optional**. If not implemented, a default handler is used that generates refresh tokens consisting of 40 characters in the range of `a..z0..9`.

Invoked during:

- `authorization_code` grant
- `refresh_token` grant
- `password` grant

Arguments:

Name	Type	Description
client	Object	The client the refresh token is generated for.
user	Object	The user the refresh token is generated for.
scope	String	The scopes associated with the refresh token. Can be `null`.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

A `String` to be used as refresh token.

RFC 6749 Appendix A.17 specifies that refresh tokens must consist of characters inside the range `0x20..0x7E` (i.e. only printable US-ASCII characters).

Remarks:

`client` is the object previously obtained through *Model#getClient()*.

`user` is the user object previously obtained through *Model#getAuthorizationCode()* (`code.user`; authorization code grant), *Model#getRefreshToken()* (`token.user`; refresh token grant) or *Model#getUser()* (password grant).

1.8.3 generateAuthorizationCode(client, user, scope, [callback])

Invoked to generate a new authorization code.

This model function is **optional**. If not implemented, a default handler is used that generates authorization codes consisting of 40 characters in the range of `a..z0..9`.

Invoked during:

- `authorization_code` grant

Arguments:

Name	Type	Description
client	Object	The client the authorization code is generated for.
user	Object	The user the authorization code is generated for.
scope	String	The scopes associated with the authorization code. Can be `null`.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

A `String` to be used as authorization code.

RFC 6749 Appendix A.11 specifies that authorization codes must consist of characters inside the range `0x20..0x7E` (i.e. only printable US-ASCII characters).

1.8.4 getAccessToken(accessToken, [callback])

Invoked to retrieve an existing access token previously saved through *Model#saveToken()*.

This model function is **required** if *OAuth2Server#authenticate()* is used.

Invoked during:

- request authentication

Arguments:

Name	Type	Description
accessToken	String	The access token to retrieve.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the access token and associated data.

Name	Type	Description
token	Object	The return value.
token.accessToken	String	The access token passed to `getAccessToken()`.
[token.accessTokenExpiresAt]	Date	The expiry time of the access token.
[token.scope]	String	The authorized scope of the access token.
token.client	Object	The client associated with the access token.
token.client.id	String	A unique string identifying the client.
token.user	Object	The user associated with the access token.

`token.client` and `token.user` can carry additional properties that will be ignored by *oauth2-server*.

Remarks:

```
function getAccessToken(accessToken) {
  // imaginary DB queries
  db.queryAccessToken({access_token: accessToken})
    .then(function(token) {
      return Promise.all([
        token,
        db.queryClient({id: token.client_id}),
        db.queryUser({id: token.user_id})
      ]);
    })
    .spread(function(token, client, user) {
      return {
        accessToken: token.access_token,
        accessTokenExpiresAt: token.expires_at,
        scope: token.scope,
        client: client, // with 'id' property
        user: user
      };
    });
}
```

1.8.5 `getRefreshToken(refreshToken, [callback])`

Invoked to retrieve an existing refresh token previously saved through *Model#saveToken()*.

This model function is **required** if the `refresh_token` grant is used.

Invoked during:

- `refresh_token` grant

Arguments:

Name	Type	Description
refreshToken	String	The access token to retrieve.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the refresh token and associated data.

Name	Type	Description
token	Object	The return value.
token.refreshToken	String	The refresh token passed to `getRefreshToken()`.
[token.refreshTokenExpiresAt]	Date	The expiry time of the refresh token.
[token.scope]	String	The authorized scope of the refresh token.
token.client	Object	The client associated with the refresh token.
token.client.id	String	A unique string identifying the client.
token.user	Object	The user associated with the refresh token.

`token.client` and `token.user` can carry additional properties that will be ignored by *oauth2-server*.

Remarks:

```
function getRefreshToken(refreshToken) {
  // imaginary DB queries
  db.queryRefreshToken({refresh_token: refreshToken})
    .then(function(token) {
      return Promise.all([
        token,
        db.queryClient({id: token.client_id}),
        db.queryUser({id: token.user_id})
      ]);
    })
    .spread(function(token, client, user) {
      return {
        refreshToken: token.refresh_token,
        refreshTokenExpiresAt: token.expires_at,
        scope: token.scope,
        client: client, // with 'id' property
        user: user
      };
    });
}
```

1.8.6 `getAuthorizationCode(authorizationCode, [callback])`

Invoked to retrieve an existing authorization code previously saved through *Model#saveAuthorizationCode()*.

This model function is **required** if the `authorization_code` grant is used.

Invoked during:

- `authorization_code` grant

Arguments:

Name	Type	Description
authorizationCode	String	The authorization code to retrieve.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the authorization code and associated data.

Name	Type	Description
code	Object	The return value.
code.code	String	The authorization code passed to `getAuthorizationCode()`.
code.expiresAt	Date	The expiry time of the authorization code.
[code.redirectUri]	String	The redirect URI of the authorization code.
[code.scope]	String	The authorized scope of the authorization code.
code.client	Object	The client associated with the authorization code.
code.client.id	String	A unique string identifying the client.
code.user	Object	The user associated with the authorization code.

`code.client` and `code.user` can carry additional properties that will be ignored by *oauth2-server*.

Remarks:

```
function getAuthorizationCode(authorizationCode) {
  // imaginary DB queries
  db.queryAuthorizationCode({authorization_code: authorizationCode})
    .then(function(code) {
      return Promise.all([
        code,
        db.queryClient({id: code.client_id}),
        db.queryUser({id: code.user_id})
      ]);
    })
    .spread(function(code, client, user) {
      return {
        code: code.authorization_code,
        expiresAt: code.expires_at,
        redirectUri: code.redirect_uri,
        scope: code.scope,
        client: client, // with 'id' property
        user: user
      };
    });
}
```

1.8.7 `getClient(clientId, clientSecret, [callback])`

Invoked to retrieve a client using a client id or a client id/client secret combination, depending on the grant type.

This model function is **required** for all grant types.

Invoked during:

- `authorization_code` grant
- `client_credentials` grant
- `refresh_token` grant
- `password` grant

Arguments:

Name	Type	Description
clientId	String	The client id of the client to retrieve.
clientSecret	String	The client secret of the client to retrieve. Can be `null`.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the client and associated data, or a falsy value if no such client could be found.

Name	Type	Description
client	Object	The return value.
client.id	String	A unique string identifying the client.
[client.redirectUris]	Array<String>	Redirect URIs allowed for the client. Required for the `authorization_code` grant.
client.grants	Array<String>	Grant types allowed for the client.
[client.accessTokenLifetime]	Number	Client-specific lifetime of generated access tokens in seconds.
[client.refreshTokenLifetime]	Number	Client-specific lifetime of generated refresh tokens in seconds.

The return value (`client`) can carry additional properties that will be ignored by *oauth2-server*.

Remarks:

```
function getClient(clientId, clientSecret) {
  // imaginary DB query
  let params = {client_id: clientId};
  if (clientSecret) {
    params.client_secret = clientSecret;
  }
  db.queryClient(params)
    .then(function(client) {
      return {
        id: client.id,
        redirectUris: client.redirect_uris,
        grants: client.grants
      };
    });
}
```

1.8.8 `getUser(username, password, [callback])`

Invoked to retrieve a user using a username/password combination.

This model function is **required** if the `password` grant is used.

Invoked during:

- `password` grant

Arguments:

Name	Type	Description
username	String	The username of the user to retrieve.
password	String	The user's password.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the user, or a falsy value if no such user could be found. The user object is completely transparent to *oauth2-server* and is simply used as input to other model functions.

Remarks:

```
function getUser(username, password) {
  // imaginary DB query
  return db.queryUser({username: username, password: password});
}
```

1.8.9 getUserFromClient(client, [callback])

Invoked to retrieve the user associated with the specified client.

This model function is **required** if the `client_credentials` grant is used.

Invoked during:

- `client_credentials` grant

Arguments:

Name	Type	Description
client	Object	The client to retrieve the associated user for.
client.id	String	A unique string identifying the client.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the user, or a falsy value if the client does not have an associated user. The user object is completely transparent to *oauth2-server* and is simply used as input to other model functions.

Remarks:

`client` is the object previously obtained through *Model#getClient()*.

```
function getUserFromClient(client) {
  // imaginary DB query
  return db.queryUser({id: client.user_id});
}
```

1.8.10 saveToken(token, client, user, [callback])

Invoked to save an access token and optionally a refresh token, depending on the grant type.

This model function is **required** for all grant types.

Invoked during:

- `authorization_code` grant
- `client_credentials` grant
- `refresh_token` grant
- `password` grant

Arguments:

Name	Type	Description
token	Object	The token(s) to be saved.
token.accessToken	String	The access token to be saved.
token.accessTokenExpiresAt	Date	The expiry time of the access token.
[token.refreshToken]	String	The refresh token to be saved.
[token.refreshTokenExpiresAt]	Date	The expiry time of the refresh token.
[token.scope]	String	The authorized scope of the token(s).
client	Object	The client associated with the token(s).
user	Object	The user associated with the token(s).
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

An `Object` representing the token(s) and associated data.

Name	Type	Description
token	Object	The return value.
token.accessToken	String	The access token passed to `saveToken()`.
token.accessTokenExpiresAt	Date	The expiry time of the access token.
token.refreshToken	String	The refresh token passed to `saveToken()`.
token.refreshTokenExpiresAt	Date	The expiry time of the refresh token.
[token.scope]	String	The authorized scope of the access token.
token.client	Object	The client associated with the access token.
token.client.id	String	A unique string identifying the client.
token.user	Object	The user associated with the access token.

`token.client` and `token.user` can carry additional properties that will be ignored by *oauth2-server*.

If the `allowExtendedTokenAttributes` server option is enabled (see *OAuth2Server#token()*) any additional attributes set on the result are copied to the token response sent to the client.

Remarks:

```
function saveToken(token, client, user) {
  // imaginary DB queries
  let fns = [
    db.saveAccessToken({
      access_token: token.accessToken,
      expires_at: token.accessTokenExpiresAt,
      scope: token.scope,
      client_id: client.id,
      user_id: user.id
    }),
    db.saveRefreshToken({
      refresh_token: token.refreshToken,
      expires_at: token.refreshTokenExpiresAt,
      scope: token.scope,
      client_id: client.id,
      user_id: user.id
    })
  ];
  return Promise.all(fns);
    .spread(function(accessToken, refreshToken) {
```

(continues on next page)

(continued from previous page)

```
      return {
        accessToken: accessToken.access_token,
        accessTokenExpiresAt: accessToken.expires_at,
        refreshToken: refreshToken.refresh_token,
        refreshTokenExpiresAt: refreshToken.expires_at,
        scope: accessToken.scope,
        client: {id: accessToken.client_id},
        user: {id: accessToken.user_id}
      };
    });
}
```

1.8.11 `saveAuthorizationCode(code, client, user, [callback])`

Invoked to save an authorization code.

This model function is **required** if the `authorization_code` grant is used.

Invoked during:

- `authorization_code` grant

Arguments:

Name	Type	Description
code	Object	The code to be saved.
code.authorizationCode	String	The authorization code to be saved.
code.expiresAt	Date	The expiry time of the authorization code.
code.redirectUri	String	The redirect URI associated with the authorization code.
[code.scope]	String	The authorized scope of the authorization code.
client	Object	The client associated with the authorization code.
user	Object	The user associated with the authorization code.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Todo: Is `code.scope` really optional?

Return value:

An `Object` representing the authorization code and associated data.

Name	Type	Description
code	Object	The return value.
code.authorizationCode	String	The authorization code passed to `saveAuthorizationCode()`.
code.expiresAt	Date	The expiry time of the authorization code.
code.redirectUri	String	The redirect URI associated with the authorization code.
[code.scope]	String	The authorized scope of the authorization code.
code.client	Object	The client associated with the authorization code.
code.client.id	String	A unique string identifying the client.
code.user	Object	The user associated with the authorization code.

`code.client` and `code.user` can carry additional properties that will be ignored by *oauth2-server*.

Remarks:

```
function saveAuthorizationCode(code, client, user) {
  // imaginary DB queries
  let authCode = {
    authorization_code: code.authorizationCode,
    expires_at: code.expiresAt,
    redirect_uri: code.redirectUri,
    scope: code.scope,
    client_id: client.id,
    user_id: user.id
  };
  return db.saveAuthorizationCode(authCode)
    .then(function(authorizationCode) {
      return {
        authorizationCode: authorizationCode.authorization_code,
        expiresAt: authorizationCode.expires_at,
        redirectUri: authorizationCode.redirect_uri,
        scope: authorizationCode.scope,
        client: {id: authorizationCode.client_id},
        user: {id: authorizationCode.user_id}
      };
    });
}
```

1.8.12 `revokeToken(token, [callback])`

Invoked to revoke a refresh token.

This model function is **required** if the `refresh_token` grant is used.

Invoked during:

- `refresh_token` grant

Arguments:

Name	Type	Description
token	Object	The token to be revoked.
token.refreshToken	String	The refresh token.
[token.refreshTokenExpiresAt]	Date	The expiry time of the refresh token.
[token.scope]	String	The authorized scope of the refresh token.
token.client	Object	The client associated with the refresh token.
token.client.id	String	A unique string identifying the client.
token.user	Object	The user associated with the refresh token.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

Return `true` if the revocation was successful or `false` if the refresh token could not be found.

Remarks:

`token` is the refresh token object previously obtained through *Model#getRefreshToken()*.

```
function revokeToken(token) {
  // imaginary DB queries
  return db.deleteRefreshToken({refresh_token: token.refreshToken})
    .then(function(refreshToken) {
      return !!refreshToken;
    });
}
```

1.8.13 revokeAuthorizationCode(code, [callback])

Invoked to revoke an authorization code.

This model function is **required** if the `authorization_code` grant is used.

Invoked during:

- `authorization_code` grant

Arguments:

Name	Type	Description
code	Object	The return value.
code.code	String	The authorization code.
code.expiresAt	Date	The expiry time of the authorization code.
[code.redirectUri]	String	The redirect URI of the authorization code.
[code.scope]	String	The authorized scope of the authorization code.
code.client	Object	The client associated with the authorization code.
code.client.id	String	A unique string identifying the client.
code.user	Object	The user associated with the authorization code.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

Return `true` if the revocation was successful or `false` if the authorization code could not be found.

Remarks:

`code` is the authorization code object previously obtained through *Model#getAuthorizationCode()*.

```
function revokeAuthorizationCode(code) {
  // imaginary DB queries
  return db.deleteAuthorizationCode({authorization_code: code.authorizationCode})
    .then(function(authorizationCode) {
      return !!authorizationCode;
    });
}
```

1.8.14 validateScope(user, client, scope, [callback])

Invoked to check if the requested `scope` is valid for a particular `client`/`user` combination.

This model function is **optional**. If not implemented, any scope is accepted.

Invoked during:

- `authorization_code` grant
- `client_credentials` grant
- `password` grant

Arguments:

Name	Type	Description
user	Object	The associated user.
client	Object	The associated client.
client.id	Object	A unique string identifying the client.
scope	String	The scopes to validate.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

Validated scopes to be used or a falsy value to reject the requested scopes.

Remarks:

`user` is the user object previously obtained through *Model#getAuthorizationCode()* (`code.user`; authorization code grant), *Model#getUserFromClient()* (client credentials grant) or *Model#getUser()* (password grant).

`client` is the object previously obtained through *Model#getClient* (all grants).

You can decide yourself whether you want to reject or accept partially valid scopes by simply filtering out invalid scopes and returning only the valid ones.

To reject invalid or only partially valid scopes:

```
// list of valid scopes
const VALID_SCOPES = ['read', 'write'];

function validateScope(user, client, scope) {
  if (!scope.split(' ').every(s => VALID_SCOPES.indexOf(s) >= 0)) {
    return false;
  }
  return scope;
}
```

To accept partially valid scopes:

```
// list of valid scopes
const VALID_SCOPES = ['read', 'write'];

function validateScope(user, client, scope) {
  return scope
    .split(' ')
    .filter(s => VALID_SCOPES.indexOf(s) >= 0)
    .join(' ');
}
```

Note that the example above will still reject completely invalid scopes, since `validateScope` returns an empty string if all scopes are filtered out.

1.8.15 verifyScope(accessToken, scope, [callback])

Invoked during request authentication to check if the provided access token was authorized the requested scopes.

This model function is **required** if scopes are used with *OAuth2Server#authenticate()*.

Invoked during:

- request authentication

Arguments:

Name	Type	Description
token	Object	The access token to test against
token.accessToken	String	The access token.
[token.accessTokenExpiresAt]	Date	The expiry time of the access token.
[token.scope]	String	The authorized scope of the access token.
token.client	Object	The client associated with the access token.
token.client.id	String	A unique string identifying the client.
token.user	Object	The user associated with the access token.
scope	String	The required scopes.
[callback]	Function	Node-style callback to be used instead of the returned `Promise`.

Return value:

Returns `true` if the access token passes, `false` otherwise.

Remarks:

`token` is the access token object previously obtained through *Model#getAccessToken()*.

`scope` is the required scope as given to *OAuth2Server#authenticate()* as `options.scope`.

```
function verifyScope(token, scope) {
  if (!token.scope) {
    return false;
  }
  let requestedScopes = scope.split(' ');
  let authorizedScopes = token.scope.split(' ');
  return requestedScopes.every(s => authorizedScopes.indexOf(s) >= 0);
}
```

1.9 Extension Grants

Todo: Describe how to implement extension grants.

Extension grants are registered through *OAuth2Server#token()* (`options.extendedGrantTypes`).

1.10 Migrating from 2.x to 3.x

This module is now promise-based but allows for **ES6 generators**, **async/await** (using *[babel](https://babeljs.io)* or node v7.6+), **node-style** callbacks and **promises** in your model.

1.10.1 Middlewares

The naming of the exposed middlewares has changed to match the OAuth2 _RFC_ more closely. Please refer to the table below:

oauth2-server 2.x	oauth2-server 3.x
authorise	authenticate
authCodeGrant	authorize
grant	token
errorHandler	**removed** (now handled by external wrappers)
lockdown	**removed** (specific to *Express* middleware)

1.10.2 Server options

The following server options can be set when instantiating the OAuth service:

- *addAcceptedScopesHeader*: **default true** Add the *X-Accepted-OAuth-Scopes* header with a list of scopes that will be accepted
- *addAuthorizedScopesHeader*: **default true** Add the *X-OAuth-Scopes* header with a list of scopes that the user is authorized for
- *allowBearerTokensInQueryString*: **default false** Determine if the bearer token can be included in the query string (i.e. *?access_token=*) for validation calls
- *allowEmptyState*: **default false** If true, *state* can be empty or not passed. If false, *state* is required.
- *authorizationCodeLifetime*: **default 300** Default number of milliseconds that the authorization code is active for
- *accessTokenLifetime*: **default 3600** Default number of milliseconds that an access token is valid for
- *refreshTokenLifetime*: **default 1209600** Default number of milliseconds that a refresh token is valid for
- *allowExtendedTokenAttributes*: **default false** Allows additional attributes (such as *id_token*) to be included in token responses.
- *requireClientAuthentication*: **default true for all grant types** Allow ability to set client/secret authentication to *false* for a specific grant type.

The following server options have changed behavior in v3.0.0:

- *accessTokenLifetime* can no longer be set to *null* to indicate a non-expiring token. The recommend alternative is to set accessTokenLifetime to a high value.

The following server options have been removed in v3.0.0:

- *grants*: **removed** (now returned by the *getClient* method).
- *debug*: **removed** (not the responsibility of this module).
- *clientIdRegex*: **removed** (the *getClient* method can return *undefined* or throw an error).
- *passthroughErrors*: **removed** (not the responsibility of this module).
- *continueAfterResponse*: **removed** (not the responsibility of this module).

1.10.3 Model specification

- *generateAccessToken(client, user, scope)* is **optional** and should return a *String*.

- *generateAuthorizationCode()* is **optional** and should return a *String*.
- *generateRefreshToken(client, user, scope)* is **optional** and should return a *String*.
- *getAccessToken(token)* should return an object with:
  - *accessToken* (*String*)
  - *accessTokenExpiresAt* (*Date*)
  - *client* (*Object*), containing at least an *id* property that matches the supplied client
  - *scope* (optional *String*)
  - *user* (*Object*)
- *getAuthCode()* was renamed to *getAuthorizationCode(code)* and should return:
  - *client* (*Object*), containing at least an *id* property that matches the supplied client
  - *expiresAt* (*Date*)
  - *redirectUri* (optional *String*)
  - *user* (*Object*)
- *getClient(clientId, clientSecret)* should return an object with, at minimum:
  - *redirectUris* (*Array*)
  - *grants* (*Array*)
- *getRefreshToken(token)* should return an object with:
  - *refreshToken* (*String*)
  - *client* (*Object*), containing at least an *id* property that matches the supplied client
  - *refreshTokenExpiresAt* (optional *Date*)
  - *scope* (optional *String*)
  - *user* (*Object*)
- *getUser(username, password)* should return an object:
  - No longer requires that *id* be returned.
- *getUserFromClient(client)* should return an object:
  - No longer requires that *id* be returned.
- *grantTypeAllowed()* was **removed**. You can instead:
  - Return *falsy* in your *getClient()*
  - Throw an error in your *getClient()*
- *revokeAuthorizationCode(code)* is **required** and should return true
- *revokeToken(token)* is **required** and should return true
- *saveAccessToken()* was renamed to *saveToken(token, client, user)* and should return:
  - *accessToken* (*String*)
  - *accessTokenExpiresAt* (*Date*)
  - *client* (*Object*)
  - *refreshToken* (optional *String*)

  - *refreshTokenExpiresAt* (optional *Date*)
  - *user* (*Object*)
- *saveAuthCode()* was renamed to *saveAuthorizationCode(code, client, user)* and should return:
  - *authorizationCode* (*String*)
- *validateScope(user, client, scope)* should return a *Boolean*.

The full model specification is [also available](https://oauth2-server.readthedocs.io/en/latest/model/spec.html).

Index

R