



AppAuth

for JS

AppAuth for JavaScript is a client SDK for public clients for communicating with OAuth 2.0 and OpenID Connect providers following the best practice RFC 8252 - OAuth 2.0 for Native Apps. The library is designed for use in [Web Apps](#), [Node.js](#) CLI applications, [Chrome Apps](#) and applications that use [Electron](#) or similar frameworks.

It strives to directly map the requests and responses of those specifications, while following the idiomatic style of the implementation language.

The library also supports the PKCE extension to OAuth which was created to secure authorization codes in public clients when custom URI scheme redirects are used. The library is friendly to other extensions (standard or otherwise) with the ability to handle additional parameters in all protocol requests and responses.

Examples

An example application using the library is included in the [src/node_app](#) folder and at <https://github.com/googlesamples/appauth-js-electron-sample>.

Auth Flow AppAuth supports manual interaction with the Authorization Server where you need to perform your own token exchanges. This example performs a manual exchange.

Fetch Service Configuration

```
1 AuthorizationServiceConfiguration.fetchFromIssuer(openIdConnectUrl)
2   .then(response => {
3     log('Fetched service configuration', response);
4     this.configuration = response;
5     this.showMessage('Completed fetching configuration');
6   })
7   .catch(error => {
8     log('Something bad happened', error);
9     this.showMessage(`Something bad happened ${error}`)
10  });
```

Make Authorization Requests

```
1 this.notifier = new AuthorizationNotifier();
2 // uses a redirect flow
3 this.authorizationHandler = new RedirectRequestHandler();
4 // set notifier to deliver responses
5 this.authorizationHandler.setAuthorizationNotifier(this.notifier);
6 // set a listener to listen for authorization responses
7 this.notifier.setAuthorizationListener((request, response, error) => {
8     log('Authorization request complete ', request, response, error);
9     if (response) {
10         this.code = response.code;
11         this.showMessage(`Authorization Code ${response.code}`);
12     }
13 });
14
15 // create a request
16 let request = new AuthorizationRequest({
17     client_id: clientId,
18     redirect_uri: redirectUri,
19     scope: scope,
20     response_type: AuthorizationRequest.RESPONSE_TYPE_CODE,
21     state: undefined,
22     extras: {'prompt': 'consent', 'access_type': 'offline'}
23 });
24
25 // make the authorization request
26 this.authorizationHandler.performAuthorizationRequest(this.configuration, request);
```

Making Token Requests

```
1 this.tokenHandler = new BaseTokenRequestHandler();
2
3 let request: TokenRequest|null = null;
4
5 if (this.code) {
6     let extras: StringMap|undefined = undefined;
7     if (this.request && this.request.internal) {
8         extras = {};
9         extras['code_verifier'] = this.request.internal['code_verifier'];
10    }
11    // use the code to make the token request.
12    request = new TokenRequest({
13        client_id: clientId,
14        redirect_uri: redirectUri,
15        grant_type: GRANT_TYPE_AUTHORIZATION_CODE,
16        code: this.code,
17        refresh_token: undefined,
18        extras: extras
19    });
20 } else if (this.tokenResponse) {
```

```
21 // use the token response to make a request for an access token
22 request = new TokenRequest({
23     client_id: clientId,
24     redirect_uri: redirectUri,
25     grant_type: GRANT_TYPE_REFRESH_TOKEN,
26     code: undefined,
27     refresh_token: this.tokenResponse.refreshToken,
28     extras: undefined
29 });
30 }
31
32 this.tokenHandler.performTokenRequest(this.configuration, request)
33   .then(response => {
34     // ... do something with token response
35   });
```

Development

Preamble This client has been written with TypeScript.

Setup

- Install the latest version of Node. NVM (Node Version Manager is highly recommended).
- Use `nvm install` to install the recommended Node.js version.
- Download the latest version of Visual Studio Code from [here](#).

Provision Dependencies This app uses `npm` to provision its dependencies.

- `git clone` the `AppAuthJS` library and go to the root folder of the project containing `package.json` file.
- `npm install` to install all the dev and project dependencies.

That's it! You are now ready to start working on `AppAuthJS`.

Development Workflow The project uses `npm` scripts to automate development workflows. These scripts are made available via the `package.json` file.

The following scripts are included:

- `npm run-script compile` or `tsc` will compile all your TypeScript files. All compiled files go into the `built/` folder.

-
- `npm run-script watch` or `tsc --watch` will compile your TypeScript files in `watch` mode. Recommended if you want to get continuous feedback.
 - `npm run-script build-app` generates the output `bundle.js` file in the `built/` directory. This includes the full `AppAuthJS` library including all its dependencies.
 - `npm test` provisions the `Karma` test runner to run all unit tests. All tests are written using `Jasmine`. To *DEBUG* your tests, click on the `Debug` button in the Karma test runner to look at the actual source of the tests. You can attach break points here.
 - `npm run-script app` builds the test app on a local web server. This is an end-to-end app which uses `AppAuthJS` and is a demonstration on how to use the library.
 - `npm run-script node-app` builds a Node.js CLI sample app. This is an end-to-end app which uses `AppAuthJS` in a Node.js context.