OAuth Token Exchange Extensions

IETF 122 (Mar 21, 2025)

Nick Watson (Google LLC)

ToC

- Error expressiveness
- Refresh token expiration

Problem: Errors insufficiently expressive

error code lumps most runtime errors into invalid_grant, with the rest of the errors indicating a coding or configuration error by the client

error_description is insufficient and cannot cover errors that:

- Require/allow user action (e.g. reauth required, request access from Enterprise admin)
- Need to display richer detail (e.g. explain to user why account is blocked)
- Contain sensitive data (e.g. account is flagged by AS as underage)

Spec also indicates that error_description is for the client developer, rather than the user.

Drop-off

- Google rejects hundreds of thousands of refresh token exchanges daily due to policy errors.
- Almost none of these errors (~1%) result in re-attempting an authorization flow (where an error or recovery flow could be shown).

Possible Solution 1

Add error boolean field – `reattempt_authorization` – indicating that the client should handle the error by sending the user through the authorization flow again.

Pros:

- Simple
- Allows returning semantically correct errors while decoupling expected client behavior.

Cons:

- Potentially suboptimal experience to restart authorization flow just to show an error.
- Doesn’t allow out-of-band resolution

Possible Solution 2

Add `error_display_uri` which the client can open (in system browser) if possible, or communicate to the user through some other channel (e.g. email).

Optional: Client can attach `redirect_uri` and `state` to this URI to allow recovery in the case where the existing RT can still be used (e.g. some forms of step-up auth).

Pros:

- Most flexible, can address all problems presented in first slide

Cons:

- Most complex
- Provides limited benefit if the client has no way to open or present a URL

Possible Solution 3 (aka do nothing)

Return `invalid_grant` for all runtime errors (i.e. not coding or config errors that result in `invalid_request` etc.)

Pros:

- Clients (should) already handle this error
- No protocol update required (but spec would need broader definition of `invalid_grant`)

Cons:

- Philosophically wrong.
- Aliasing all errors prevents clients from handling them differently.
- Potentially suboptimal experience to restart authorization flow just to show an error.

Refresh token expiration

Two concepts at play:

- Consent expiration: how long is the user's consent good for (e.g. user opts for 30-day consent)
- Credential expiration: how long is the returned credential good for (e.g. when implementing RT rotation)

Refresh token will expire in `min(rotation_cadence, user_selected_expiry)`.

Refresh token expiration

Clients want to know refresh token expiration. Is it useful for clients to know both of these values individually?

- For consent expiration, clients wish to remind the user of upcoming consent expiration while the user is active to avoid interruption of service if the token expires while the user is inactive.
- For RT rotation, AS documentation *could* be good enough for clients to know how often they need to call.
    - The AS can always rate-limit clients rotating too fast by simply returning the existing RT.

Refresh token expiration

A single field set to `min(rotation_cadence, user_selected_expiry)` will not help clients who wish to remind the user if the AS also implements RT rotation. (Users who gave, say, 60d consent should not be bothered daily for RT rotation.)

But if RT rotation cadence can be addressed adequately by AS documentation, then a single field would be sufficient to address the consent expiry case.