FedCM | Error API and Auto-Selected Flag API

Author: yigu@chromium.org *Status:* ***Final*** *Last Updated:* ***2023-08-30***

Motivation

Error API

Today, when a user clicks the “Continue as” button on the FedCM UI to sign in to RP with IdP, if something goes wrong and no token is issued by the IdP, the request will fail silently. The user may be confused in such a case because it’s likely that they won’t see any notification about the error. The FedCM API currently lacks a dedicated API surface that enables an IdP to return information about what went wrong and where to go from there to the RP or the browser. The only option is to overload the IdentityCredential.token parameter and rely on the API caller (IdP SDK, RP, or 3P library) to decode and display error messages. Unfortunately, this creates two problems:

1. IdPs have no guarantees that their errors will be handled as they expect by their callers.
2. RPs have to special-case each IdP to parse and display the IdP-specific error.

A dedicated API surface would address these problems by providing a standard way for IdPs to return error information to RPs and/or users. This would allow IdPs to be confident that their errors would be handled correctly, and it would eliminate the need for RPs to special-case each IdP.

This is a proposal to introduce an API to inform the user agent about errors and handle them consistently across IdPs.

AutoSelectedFlag API

Auto re-authentication is the default user mediation behavior in Credential Management API. However, auto re-authentication may be unavailable due to reasons that only the browser knows; when it’s unavailable the user may be prompted to sign in with explicit user mediation which is a flow with different properties.

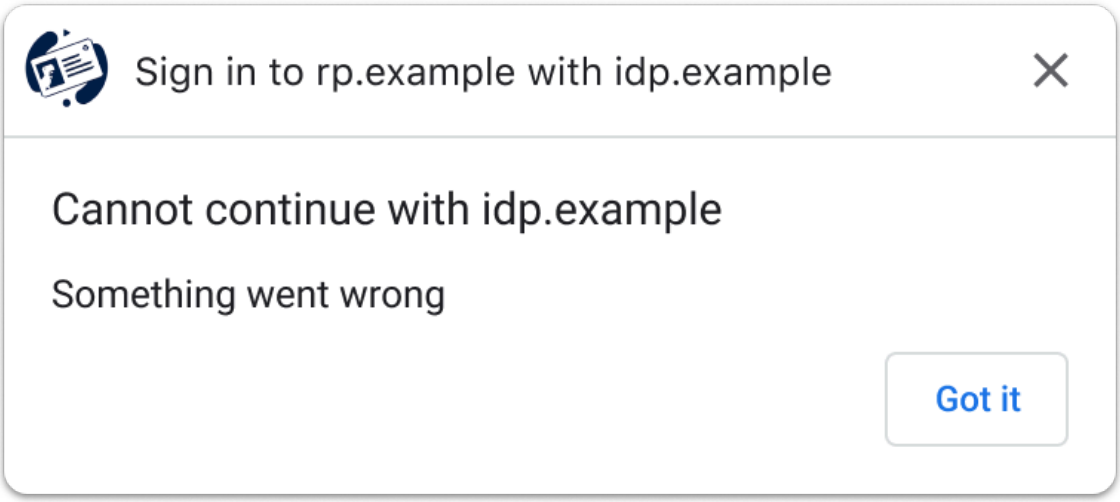
- From an API caller’s perspective, when they receive an id token, they don’t have visibility over whether it was an outcome of an auto re-authn flow. That makes it hard for them to evaluate the API performance and improve UX accordingly.
- From the IdP’s perspective, they are equally unable to tell whether an auto re-authn occurred or not. Whether an explicit user mediation was involved could help them support more security related features. e.g. some users and/or RPs may prefer a higher security tier which requires explicit user mediation in authentication. If an IdP receives a token request, they could reject it if the user didn’t grant permission in the flow.

Therefore, providing visibility of the auto re-authentication flow would be beneficial to developers.

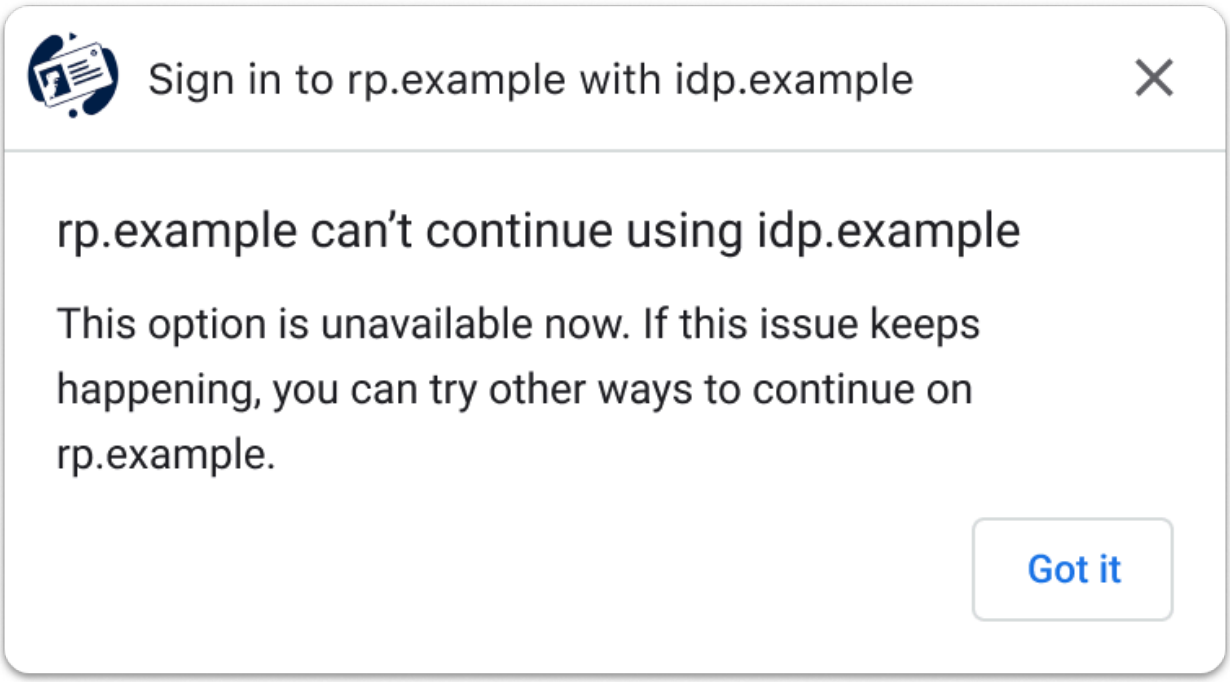
Proposal

Error API

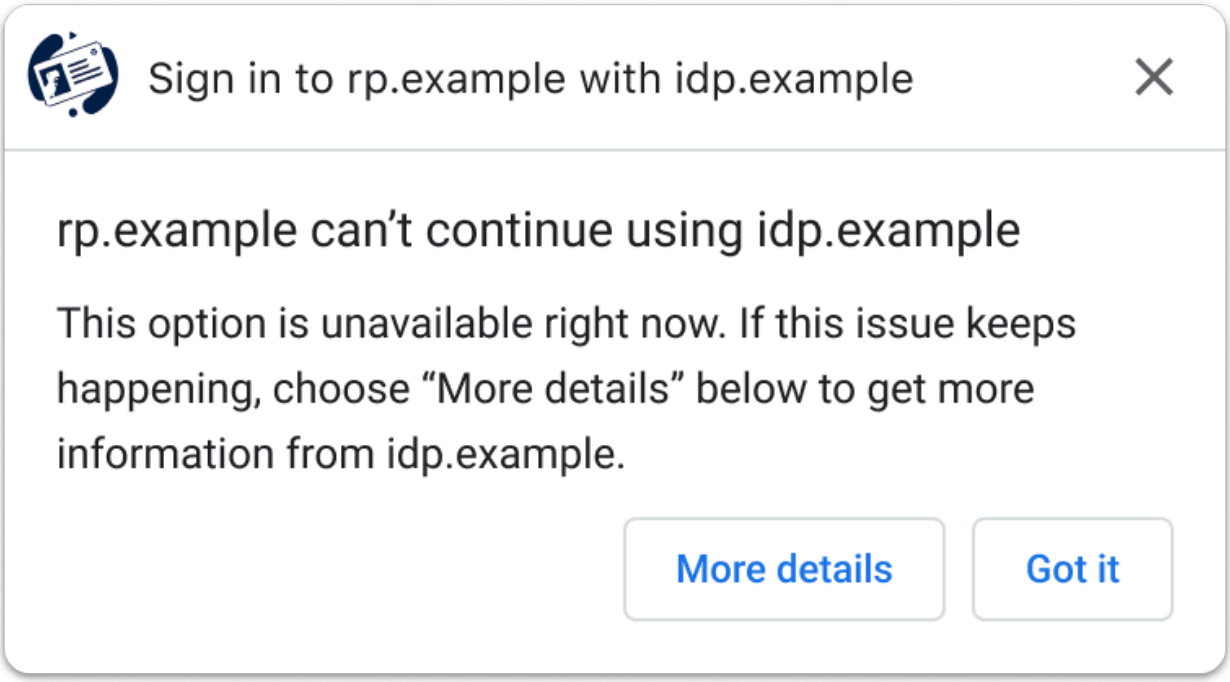
In this proposal, to guarantee to IdPs that errors are shown to inform users that their sign-in attempt has failed, the browser can support displaying error dialogs. To start with, a general purpose error dialog can cover a lot of cases and guarantee that users are notified in a consistent manner across RPs and IdPs. For example (all mocks are browser specific and subject to change):



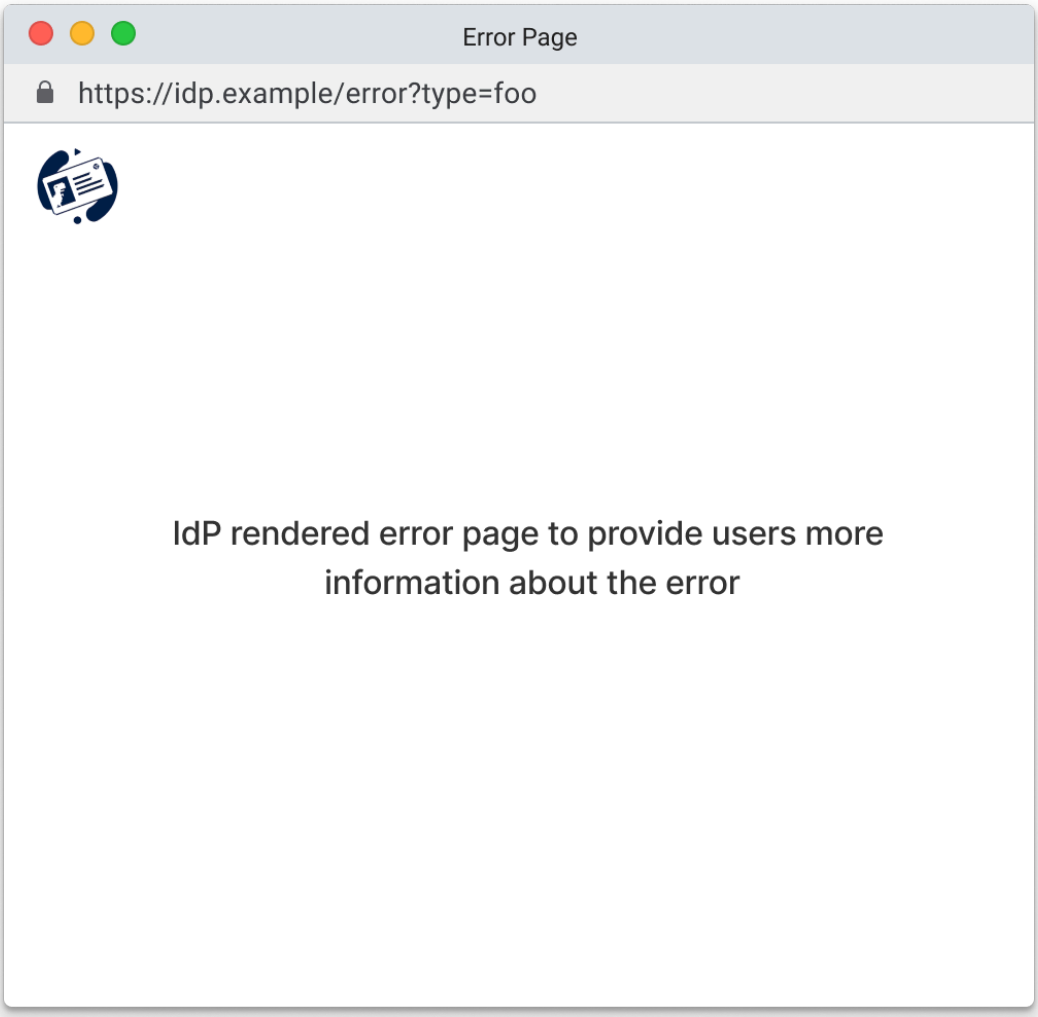
While the general purpose error dialog is better than the status quo, there are known error cases where a more purpose-specific dialog could give users more specific information. Therefore, it would be better if the browser can show some specific error dialogs to give the user more context. e.g.



The enumeration of specific errors could be insufficient and it’s possible that users would want to learn more about the error and potentially some next steps to fix the error. Therefore the browser can provide some affordance to achieve that, e.g. via a new “More details” button.



When the user clicks the “More details” button, the browser can open a pop-up window to show a IdP controlled page with detailed information about the error.

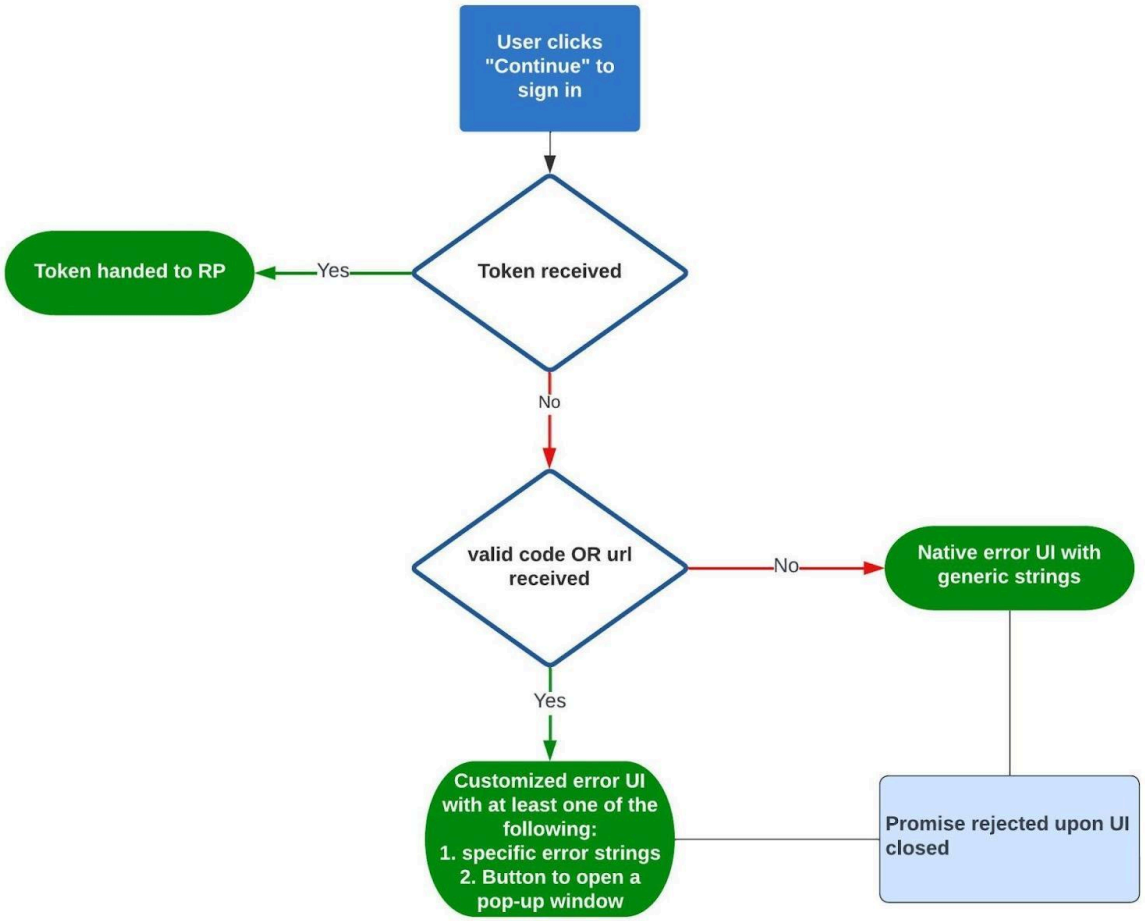


Mocks

To see the mocks for all enumerated errors, please see this doc.

Proposed API

In this proposal, to support the error dialogs described above, the browser introduces error codes and error urls that can be used whenever the IdP cannot produce a token.



The IdP HTTP API

In the `id_assertion_endpoint`, currently the IdP can return a `token` to the browser if it can be issued upon request. In this proposal, in case a token cannot be issued, the IdP can return an “error” response, which has two new fields:

1. code
2. url

code

OPTIONAL. The IdP can use the “code” field to specify one of the known errors from [`invalid_request`, `unauthorized_client`, `access_denied`, `server_error` and `temporarily_unavailable`]. e.g.

```
None
// id_assertion_endpoint response
{
  "error" : {
     "code": "unauthorized_client"
  }
}
```

The list is based on the OAuth 2.0 error response table to cover common errors across IdPs.

If a valid `code` is included in the response of the token request, the browser can trigger a native UI with proper strings to notify users that their sign-in attempt was failed due to the corresponding error. See considered alternatives for other options to show errors.

If an `error` is provided but the code is not on the pre-defined list, we propose to pass the string to the API caller and show the uncustomized generic error UI above.

url

OPTIONAL. A URL identifying a human-readable web page with information about the error, used to provide additional information about the error to users. The uri must be of the same-origin as the IdP `configURL` .

```
None
// id_assertion_endpoint response
{
  "error": {
    "url" : "https://idp.example/error?type=foo"
  }
}
```

This field is useful to users because browsers cannot provide rich error messages on a native UI. e.g. links for next steps, customer service contact information etc.. If a user wants to learn more about the error details and how to fix it, they could visit the provided page from the browser UI for more details.

Note

1. Both `code` and `url` are optional in case of token request failures. The browser should provide a fallback UI to keep users aware if both are missing.
    - The new browser error UI may conflict with existing error UI rendered by the RP if any. That said, the risk is extremely low based on how we see current IdPs implementing FedCM.

    - We believe that the browser should be opinionated to render a fallback error UI to make sure that users are informed when their sign-in attempt has failed.
2. It's possible that there's no response returned from the `id_assertion_endpoint` in which case the browser cannot be sure that an error occurred. Without a time-out mechanism, the browser won't show any UI in this case.

The Client JS API

To give more context to the API caller such that they could provide more sign-in options to users, the browser can pass over the error (`code` and `url)` by failing the promise:

```
None
try {
  await avigator.credentials.get({
    identity: {
      // ...
    }
  });
} catch ({code, url}) {
  // Oops, something went wrong
}
```

AutoSelectedFlag API

In this proposal, the browser shares whether an explicit user permission was acquired in the transaction with both the IdP and RP whenever auto re-authentication occurred or an explicit mediation occurred**. Note that the sharing only happens post user permission for IdP/RP communication.**

IdP

To share the information to the IdP **post user permission**, we can include it in the POST request sent to the `id_assertion_endpoint`:

```
POST /fedcm_assertion_endpoint HTTP/1.1
Host: idp.example
Origin: https://rp.example/
Content-Type: application/x-www-form-urlencoded
Cookie: 0x23223
Sec-Fetch-Dest: webidentity
```

```
account_id=123&client_id=client1234&nonce=Ct60bD&disclosure_text_shown=true&is_auto_selected=true
```

API caller (RP, IdP SDK etc.)

The browser can share the information to the RP via `IdentityCredential`:

```
None
// IdentityCredential object
{
  "token": "eyJC...J9.eyJzdWTE2...MjM5MDIyfQ.SflV_adQssw....5c",
  "isAutoSelected": true
}
```

Privacy Consideration

Error API

The new Error Response API is only invoked **post user permission** to allow RP/IdP communication. e.g. the user is aware that they are “signing in to RP with IdP”. In addition, the IdP has already possessed both the RP information and the user cookie from the `id_assertion_endpoint`. Therefore we believe that there’s no change in the privacy threat model with the new API.

AutoSelectedFlag API

Sharing with IdP

The `is_auto_selected / isAutoSelected` bit is `true` if all of the following are true:

1. `preventSilentAccess` was not in effect (it was not used in the past or it was used but the user has signed in again after that)
2. `mediation: required` is not used in FedCM API
3. the user has only one active session with the IdP
4. the user has granted permission for the {RP, IdP} pair in the browser in the past and the permission is not cleared
5. auto re-authn was not triggered in the last 10 mins

The information in #1 and #2 is already known to the IdP by virtue of the RP having used the SDK. If the SDK is not used, it is still acceptable to expose this information to the IdP, as the RP has already trusted the IdP for federated sign-in and the additional information is closely related to the API.

#3 is IdP populated information.

The information in #4 and #5 is browser-specific information. As the user has already granted permission for the RP/IdP to communicate, revealing this information to the IdP does not introduce any privacy risks.

On the other hand, if the `is_auto_selected / isAutoSelected` bit is `false`, IdP would learn less because they couldn't tell the exact reason. e.g. a user may have started a new browser client or they have cleared site data etc..

Sharing with RP (API caller)

We believe the same analysis above applies to RP as well. In addition, once an IdP has obtained the information, it can already share it with the RP via the opaque `token` string so the browser doesn't introduce any new risk.

Security Consideration

Error API

When the user clicks the "More details" button, we open a popup (same UI and web platform properties as what one would get with `window.open(url,"","popup,noopener,noreferrer")`) that loads the "error.url". Note that no communication between the website and this pop-up is allowed (e.g. no postMessage, no window.opener).

Phishing

The primary threat is a phishing attack, where the attacker (who controls - or colludes with - both the RP as well as the IdP) can provide a fake "error.url" (that impersonates a real IdP) for the browser to display via the pop-up window, and trick the user to enter their (real IdP) password there. e.g. upon user clicking the "More details" button, the browser will open a page that looks like a genuine "Sign in to IdP" website. Then the user "may" be tricked into entering their IdP credentials on that website.

Because of that, the pop-up window has the following properties:

- the preceding UI showing the eTLD+1 of the attacker in a prominent way
- the URL bar shows the full URL (same-origin with `configURL`) that is being loaded
- users are familiar with how pop-up window looks and can move it around
- safe browsing works as usual

As such, the attack has to rely on the fact that the user misses the displayed origin/site in both steps and on safe browsing not knowing about the site.

In addition to that, the attacker may already be able to do this by opening a phishing pop-up window and there's no browser UI involved compared to this proposal.

AutoSelectedFlag API

Similar to the privacy considerations above, the boolean we share with IdP doesn't introduce any security risk. Post user permission, the IdP can already share the boolean with RP directly via the `token` field so the boolean we share with RP should not have any regression.

Considered alternative

Error API

API caller handles all errors

The browser could delegate handling the errors to the API callers (RP or IdP SDK or FedCM library owned neither by RP nor IdP).

When the browser receives the errors from the token request, it rejects the promise with the errors. Once the API caller receives the error, the caller can inform users accordingly. e.g. the caller can render an iframe to show proper information to users.

Pros

- It gives the API caller more control over the error UI. e.g. they can customize the iframe and navigate users to new journeys from there.
- IdPs can expose very specific errors that represent the specific problem that occurred

Cons

- RP DX Problem: API callers need to handle errors for each IdP independently.
- UX Problem: different IdPs will produce inconsistent error UI for users.
- IdP Problem: the IdP has no guarantees that API callers will handle errors (properly), e.g. causing users not to see their error UI.

AutoSelectedFlag API

Instead of the browser sending the bit to both IdP and API callers directly, it can choose to only share it with the IdP, and then let the IdP share it with the RP. For example, IdP can integrate the information to the `token` string. While it's suboptimal to overload the well-specified OIDC id token, they can also do so with some extra FedCM support:

1. IdP first include a new field in the `id_assertion_endpoint` response

```
None
// id_assertion_endpoint response
{
 "token": "eyJC...J9.eyJzdWTE2...MjM5MDIyfQ.SflV_adQssw....5c",
 "is_auto_selected": true
}
```

2. Browser shares the information with RP the same way as this proposal does

```
None
// IdentityCredential object
{
 "token": "eyJC...J9.eyJzdWTE2...MjM5MDIyfQ.SflV_adQssw....5c",
 "isAutoSelected": true
}
```

While allowing IdP to add information in the response is **better from extensibility's perspective**, e.g. they could add more fields to share more information with the API caller, we don't find it very desirable at the moment. More importantly, we don't think it's mutually exclusive with our current proposal so we can add them later if needed.

Appendix

OAuth 2.0 Error Response: link		
Error	Description	Included in FedCM API
invalid_request	The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed.	Yes.

unauthorized_client	The client is not authorized to request an authorization code using this method.	Yes.
access_denied	The resource owner or authorization server denied the request.	Yes.
unsupported_response_type	The authorization server does not support obtaining an authorization code using this method.	No. FedCM is authentication focused at the moment and we can add this type in the future.
invalid_scope	The requested scope is invalid, unknown, or malformed.	No. FedCM is authentication focused at the moment and we can add this type in the future.
server_error	The authorization server encountered an unexpected condition that prevented it from fulfilling the request. (This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect.)	Yes.
temporarily_unavailable	The authorization server is currently unable to handle the request due to a temporary overloading or maintenance of the server. (This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect.)	Yes.