Introducing Adobe Firefly Service

At Adobe Summit this year we announced the launch of Firefly Services. This suite of APIs encompasses the Photoshop and Lightroom APIs I've discussed before, as well as a whole new suite of APIs for Firefly itself. Best of all, the APIs are easy to use. I've been building demos and samples over the past few weeks, and while I'm obviously biased, they're truly a pleasure to use. Let's look at an example of how developers can integrate these APIs today.

Basics

First, some quick basics:

1. Authentication is required and consists of a client ID and secret value, much like the Photoshop API and Acrobat Services. You exchange this for an access token that can be used for subsequent calls.

2. The Firefly APIs, when working with media, require you to upload the resource to an API endpoint first. This is different from the Photoshop API which requires cloud storage. This will be made more consistent in the future.

3. Results are provided via a cloud storage URL that you can download, or use in further calls.

4. This is all done via REST calls in whatever language, or low-code platform, you wish.

Features

Currently, the following endpoints are supported (and again, Firefly "Services" refers to the gen AI stuff, Photoshop, Lightroom, and more, I'm focusing on the generative stuff for this post):

- Upload - used to upload images that are referenced by other methods
- Text to Image - what you see in the website - you take a prompt and get images. Like the website, there's a crap ton of tuning options, including using a reference image which would make use of the upload method described above.
- Generative Expand - take an image and use AI to expand it. Basically, it expands an image with what it thinks makes sense around the existing image. Can use a prompt to help control what's added.
- Generative Fill - same idea, but fills in an area instead. Can also take a prompt.

Check out the reference for full docs.

Building a Demo

First, grab your credentials, in this case from the environment:

```
/* Set our creds based on environment variables.
*/
const CLIENT_ID = process.env.CLIENT_ID;
const CLIENT_SECRET = process.env.CLIENT_SECRET;
```

Second, exchange this for an access token. This is very similar to all the other Adobe IDs with the main exception being the `scope`:

```
async function getAccessToken(id, secret) {

    const params = new URLSearchParams();

    params.append('grant_type', 'client_credentials');
    params.append('client_id', id);
    params.append('client_secret', secret);
    params.append('scope', 'openid,AdobeID,session,additional_

    let resp = await fetch('https://ims-na1.adobelogin.com/ims
        {
            method: 'POST',
            body: params
```

```
        }
    );

    let data = await resp.json();
    return data.access_token;
}

let token = await getAccessToken(CLIENT_ID, CLIENT_SECRET);
```

Now let's make an image. The text to image API has a *lot* of options, but you can get by with the minimum of just a prompt. We also want to get a bunch of options so we'll change the default limit of 1 to 4:

```
{
    "prompt":"a stack of papers on top of a desk",
    "n":4
}
```

A basic wrapper function could look like so:

```
async function textToImage(prompt, id, token) {

    let body = {
        "n":4,
        prompt
    }


    let req = await fetch('https://firefly-api.adobe.io/v2/ima
        method:'POST',
        headers: {
            'X-Api-Key':id,
            'Authorization':`Bearer ${token}`,
            'Content-Type':'application/json'
        },
        body: JSON.stringify(body)
    });

    return await req.json();
}
```

This returns, if everything went well, a JSON packet containing links to the results. Here's an example of a result when one image is requested:


```
{
    "version": "2.10.2",
    "size": {
            "width": 2048,
            "height": 2048
    },
    "predictedContentClass": "art",
    "outputs": [
            {
                    "seed": 1613067352,
                    "image": {
                            "id": "03a221df-98a2-4597-ac2d-3dc
                            "presignedUrl": "https://pre-signe
                    }
            }
    ]
}
```


And that's it. You then just need to write some code to download the bits:

```
async function downloadFile(url, filePath) {
    let res = await fetch(url);
    const body = Readable.fromWeb(res.body);
    const download_write_stream = fs.createWriteStream(filePat
    return await finished(body.pipe(download_write_stream));
}
```

Here's one example from the prompt used above, and remember, there are numerous options that could have been tweaked, and the prompt could have been more descriptive.

Getting Started

Most of the code above was taken from the *excellent* intro guide from the Firefly docs that includes both a Node and Python version. To learn more, check out the developer homepage for Firefly Services as well!