smartsheet
ENGAGE

Advanced Use of the Smartsheet API

Moving beyond basic API calls

The Smartsheet API is...

- **Secure** – share data, not sensitive info

- **Automated** – trigger and respond to events

- **Scalable** – reliably meet your needs at high volume

Overview

What are we actually covering?

We'll discuss **what a topic is** and **how Smartsheet does it**.

- **Secure –** share data, not sensitive info
  - Smartsheet **OAuth Flow**
- **Automated –** trigger and respond to events
  - Smartsheet **webhooks**
- **Scalable –** reliably meet your needs at high volume
  - Strategies for **building at a larger scale**

OAuth 2.0

Connecting apps without sharing sensitive data

What is it?

- Open authorization protocol
- Allows apps to share information without exchanging sensitive data
- Widely supported

When do I need it?
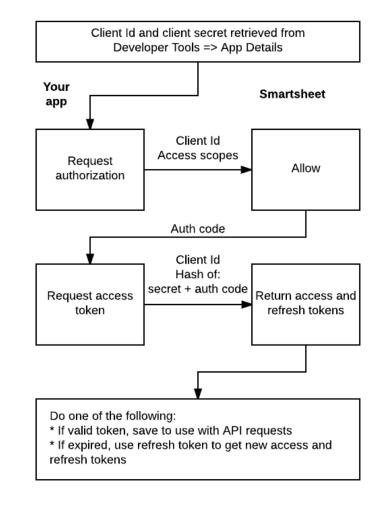
- When building "third-party" integrations
- You want other users to share *their* Smartsheet data
  - Attribution of changes
  - Security (roles and permissions)
  - Not passing around a token

OAuth Flow

Smartsheet's
3-legged OAuth Flow

1. Request an **Authorization Code**.

2. User gives consent and Smartsheet sends **Authorization Code**.

3. Authorization code (and app secret) is used to request an **Access Token**.

Setup: Register an Application

1.) Sign up for a Smartsheet Developer Account

Option A – Create new developer sandbox account

Visit page→ register → build stuff

https://developers.smartsheet.com/register

Option B – Activate 'Developer Tools' on existing account

Request access from Smartsheet support.

smartsheet
ENGAGE

Setup: Register an Application

2.) Create a new application

Click Account → Developer Tools → Create New App.

Setup: Register an Application

3.) Enter details about application
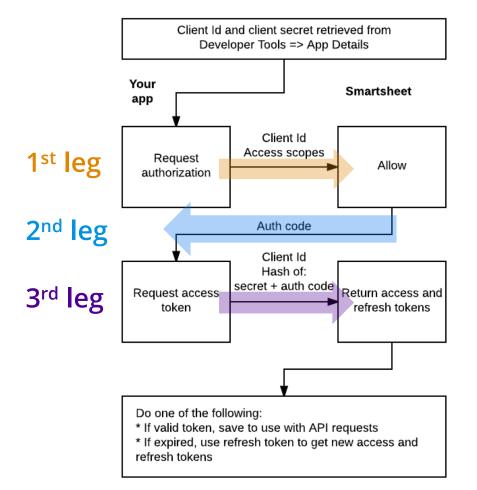
Inform users about your application when they consent to share Smartsheet data.

Smartsheet sends important data to your **App redirect URL** .

The App redirect URL *must* be a domain you control.

smartsheet
ENGAGE

Setup: Register an Application

4.) Record sensitive info for use in OAuth Flow

The **App client id** and **App secret** are used to create the OAuth Flow.

3-legged OAuth Flow
Step by step walkthrough

1st leg

2nd leg

3rd leg

Client Id and client secret retrieved from Developer Tools => App Details

Your app

Smartsheet

Request authorization — Client Id Access scopes → Allow

Auth code

Request access token — Client Id Hash of: secret + auth code → Return access and refresh tokens

Do one of the following:
* If valid token, save to use with API requests
* If expired, use refresh token to get new access and refresh tokens

This graphic and more [detailed instructions on OAuth Flow](link) are available in the API documentation.

1st leg

Requesting an Authorization Code

Client Id and client secret retrieved from
Developer Tools => App Details

Your app   **Smartsheet**

Request authorization → Client Id / Access scopes → Allow

Auth code

Request access token → Client Id / Hash of: secret + auth code → Return access and refresh tokens

Do one of the following:
* If valid token, save to use with API requests
* If expired, use refresh token to get new access and refresh tokens

Request Authorization Code

Leg #1: Smartsheet OAuth Flow

## The initial request requires three parameters:
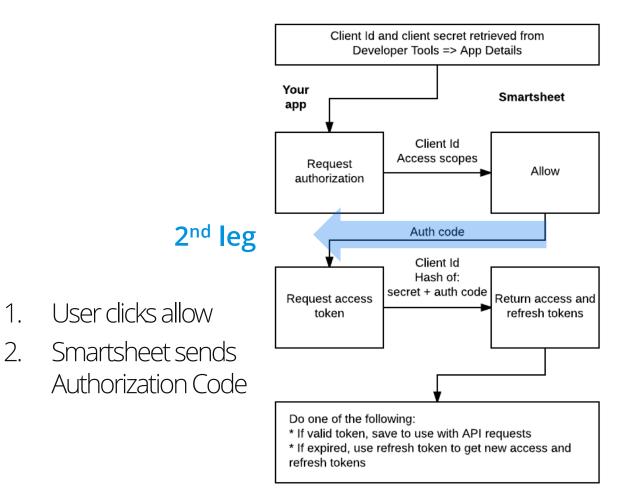
1. **App client id**
   – From the 'App Profile' page of Smartsheet
2. **Access scopes**
   – Define the level of access granted to a user
   – I.E. WRITE_SHEETS, ADMIN_WEBHOOKS, READ_CONTACTS
3. **Response type set to code**
   – Automatically added by SDKs
- **State (optional)**
   – Arbitrary string that is returned to the app. Helps add custom logic to OAuth Flow

A properly formatted request looks like this:

https://app.smartsheet.com/b/authorize?
**response_type=code**&**client_id=<your_client_id>**&**scope=<your_access_scopes>**&**state=<some_string>**

2ⁿᵈ leg

1. User clicks allow
2. Smartsheet sends Authorization Code
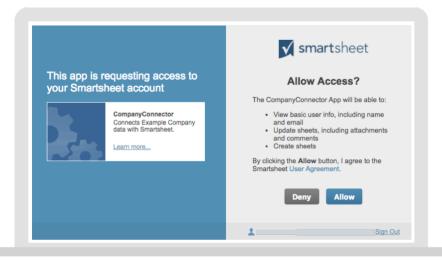
Grant Access → Receive Authorization Code

Legacy Smartsheet OAuth Flow
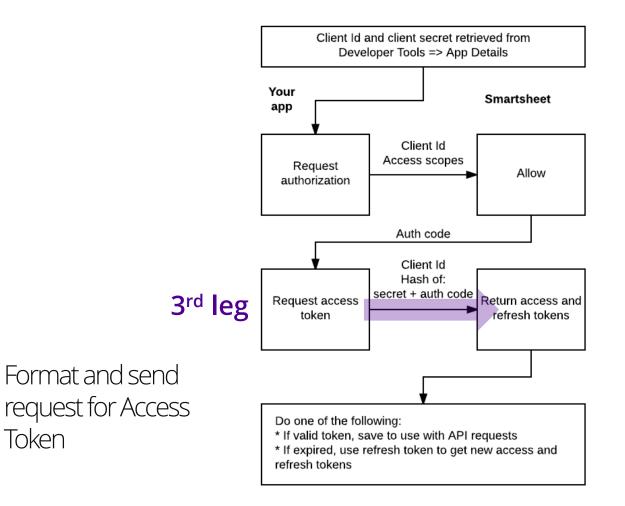
User clicks Allow—consenting to share Smartsheet data.

Smartsheet sends Authorization Code

- Smartsheet sends **Authorization Code** to application's **App redirect URL** as a query parameter.

- The **Authorization Code** must be captured for use in the final request.

Client Id and client secret retrieved from
Developer Tools => App Details

Your app

Smartsheet

Request authorization

Client Id
Access scopes

Allow

Auth code

3rd leg

Request access token

Client Id
Hash of:
secret + auth code

Return access and refresh tokens

Format and send request for Access Token

Do one of the following:
* If valid token, save to use with API requests
* If expired, use refresh token to get new access and refresh tokens

smartsheet
ENGAGE

Format Access Token Request

Leg #3: Smartsheet OAuth Flow

The final POST request to the /token endpoint must contain a user-generated Hash.

— The **Hash** is an SHA-256 hash of the **App secret** concatenated with a pipe and **authorization code**.

An example of a user-generated Hash in Node.js

```
var hash =
  crypto.createHash('sha256')
    .update('9samp7le3for71you' + '|' + 'sample6p9qisx6a')
        // ('Your App Secret' + '|' + 'Received Authorization Code')
    .digest('hex');
```

Request Access Token

Leg #3: Smartsheet OAuth Flow

```javascript
const options = {
    queryParameters: {
        client_id: config.APP_CLIENT_ID,
        code: authCode,
        hash: generated_hash
    }
};
smartsheet.tokens.getAccessToken(options, processToken)
    .then((token) => {
        return res
            .status(200)
            .json(token);
    });
```
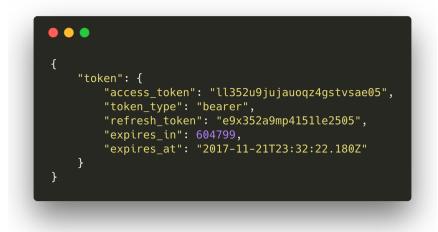
The request contains three parameters:

- Client id
- Authorization code
- Hash from previous step

Success! Access Token Received

Smartsheet sent back an Access Token

```
{
    "token": {
        "access_token": "ll352u9jujauoqz4gstvsae05",
        "token_type": "bearer",
        "refresh_token": "e9x352a9mp4151le2505",
        "expires_in": 604799,
        "expires_at": "2017-11-21T23:32:22.180Z"
    }
}
```

- **Save the entire token object**

- **The access token is valid for ~ 7 days**

- **The refresh token can renew the access token without completing another full OAuth Flow**

Automating with the API

Creating and Using Smartsheet Webhooks

Webhooks

Connecting the web with event notifications

What and why?

- Remove complexity and overhead of polling for changes

- Receive valuable information when it happens

- Event callbacks sent via HTTP POST to your callback URL

- No standard, no protocol... just a useful design pattern

How can I use them?

- Link together separate tools or services

- Respond to events with custom code

- Automate tasks

- "Glue" everything together

smartsheet
ENGAGE

Smartsheet for JIRA

Webhooks in action

- Webhooks are used on *both sides* to trigger a workflow when something changes
- Application takes action after receiving the webhook event
- Data is kept in sync automatically across two systems in real time as people work

How does Smartsheet Create Webhooks?

API Client	**1** Send Request: Create webhook `POST /webhooks`	**3** Send Request: Update webhook `PUT /webhooks/{id}` `{ "enabled": true }`	
	A ↕ B `webhook.enabled: false` `webhook.status: NEW_NOT_VERIFIED`	`webhook.enabled: true` `webhook.status: ENABLED`	
Smartsheet	**2** Respond (webhook created but not enabled)	**4** Send 'Verification Request' w/ Header: `Smartsheet-Hook-Challenge` → C	**6** Respond (webhook enabled)
		A ↕ B	
Subscriber (callback URL)		**5** Respond (HTTP Status 200) w/ Header: `Smartsheet-Hook-Response`	

Webhook in Action

Consuming webhook data from Smartsheet

Implementation differs based on programming language.

Event data is 'thin'... tells you where rather than what.

You need to query the sheet to determine changes to data.

Demo uses our [sample node.js webhook](#) (available on Github).

Smartsheet Webhooks

Automatically respond to Smartsheet events

Overview

1. Subscribe to event-notifications by creating and registering a webhook.
2. Smartsheet sends data via POST when changes occur in monitored sheet.
3. Your application receives and processes the event data.

Constraints

- Can only listen for changes on a sheet
- Webhooks fire on all changes to a sheet

Plain English

- Cell changed in sheet_b, send webhook to https://YourApp.com/abc

- Row added to sheet_w, send webhook to https://YourApp.com/def

Working at Scale

Using the Smartsheet API at higher volumes

Using the API at Higher Volumes
Considerations and strategies

- **Important concepts for working at scale**
  - Making calls efficiently
  - Using a system for logging
  - Planning for rate limits
- **Making use of Smartsheet SDKs**
  - Languages: Java, C#, Python, JavaScript (node.js), Ruby
  - Benefits of using an SDK

Making API Calls Efficiently

Using bulk operations

Many endpoints support bulk operations—allowing you to create, update, or delete multiple objects in a single request.

Practically speaking

- Call **UpdateRows** once with an array of 30 row objects instead of individually on each row object.
- Call **ShareWorkspace** once on an array of user objects instead of calling it for each user.

Using bulk operations...

➢ Better performance
➢ Prevents save errors from overwhelming an endpoint (I.E. quickly updating 60 rows individually)
➢ Keeps you from hitting the rate limit

Bulk Operations

Understanding requests with multiple objects

By default, bulk operations are all or nothing; one invalid objects makes the whole request fail

You can modify this behavior with the query parameter allowPartialSuccess=true
- Valid objects are successful
- Returns array of failed items

Don't be fooled!
- A 200 success HTTP status is returned but the **request didn't fully complete**.

```
{
    "message": "PARTIAL_SUCCESS",
    "resultCode": 3,
    "result": [],
    "version": 14,
    "failedItems": []
}
```

```
{
    "message": "SUCCESS",
    "resultCode": 0,
    "result": [{
        "type": "sheet",
        "objectId": 8400677765441412
    }]
}
```

Key Concepts to Address

Gain insight into your app and handle errors gracefully

Logging

- Capture stack traces for future reference.
- Gain visibility into errors.
- Identify what happened *before* point of failure.
- Better understand the flow of information in your app.

Rate limits

- Throttling occurs at 300 requests per minute. Smartsheet may change this.
- Some requests cost extra…
  - Cell history & adding a file attachment count as 10 API requests
  - Check the API docs!
- Plan for the worst.

Think about these at the beginning. A little planning will save a lot of pain.

About That Rate Limit...

Help! I hit the rate limit

What you get back

- Rejected request with 429 HTTP status code

- Body:
```
{
    "errorCode": 4003,
    "message": "Rate limit exceeded."
}
```

How you should handle it

- Wait and retry

This is a good reason to use the Smartsheet SDK for your respective language.

What's an SDK Again?

Smartsheet has Software Development Kits (SDK)

Languages

- Java
- C#
- Python
- JavaScript (Node.js)
- Ruby

Benefits

- Native language object models
- Logging
- Rate limit retry / backoff
- Sample code
- Full reference documentation

SDKs are Awesome

And come with nice things

SDK	Logging	Retry (rate limit)	Details
JavaScript (Node.js)	Winston	Yes	JavaScript SDK Docs
Python	Python library	Yes	Python SDK Docs*
Ruby	Standard logger	Yes	Ruby SDK Docs
C#	NLog project	Yes	C# SDK Docs — Advanced Topics
Java	Console and SLF4J	Yes	Java SDK Docs — Advanced Topics

Java SDK and C# SDK

Retry logic

The Java SDK advanced topics docs and C# SDK advanced topics docs have examples of how to override the default retry / timeout logic.

```
switch(error.getErrorCode()) {
    // The default shouldRetry, retries 4001, 4002, 4003, 4004 codes
    case 4001:
    case 4002:
    case 4003:
    case 4004:
    case 9999: // adding my fictional error code
        break;
    default:
        return false;
}
```

Error code	Meaning
4001	Smartsheet offline
4002	Server timeout
4003	Rate limit exceeded
4004	Unexpected error

...Implement your own custom retries
- E.g. an authorization error triggering a token refresh.

Advanced Use of the Smartsheet API

Learn the hard parts!

- The API can **securely** share Smartsheet data using an **OAuth Flow**.

- Learn about changes in Smartsheet **automatically** by using **webhooks**.

- For your application to **scale** well, it must **use calls efficiently**,

  implement **logging**, and contain **retry / backoff logic**.

  - Pro tip: logging and retry are built into the SDKs

Questions about the API?

Smartsheet API docs

Stack Overflow ('smartsheet-api' tag)

https://www.smartsheet.com/gethelp

Smartsheet Developer Relations

https://developers.smartsheet.com

devrel@smartsheet.com

@SmartsheetDev

Don't forget to:

- Share your feedback in our survey in the ENGAGE app
- Stop by the Administration & Custom Solutions booth on the first floor
- Visit the Innovation Center for hands-on learning, support, services, swag, and more