Adding and Updating Snowflake Events in Resource Management

Smartsheet utilizes Snowflake, a cloud-based data warehouse, to store user and action-driven events. The following sections detail Resource Management's process for instrumenting these events.

> ⚠ Events should **NOT** be instrumented until they have been approved by the Data Classification team.
> Note: In addition to `Approved`, events with the BI Status `Approved - Pending Taxonomy` in the intake request sheets are ready for engineering to proceed. This means that the BI Ops team needs to merge in changes on their backend to accept a new property.

Notes before getting started

- Request Snowflake access through through Freshservice. Confirm with your manager that you have the necessary permissions.
- Events can be found in the Product Data 2.0 Taxonomy WorkApp. Note there are two sheets, one for frontend events and one for backend events. These sheets serve as a glossary to define events and help the Product team formalize queries in Amplitude.
- The following instructions are for adding events to the RM `user-events` Snowflake table. If you are adding events to the Smartsheet events table (which you shouldn't need to unless you are adding events outside of RM-owned code), their process is a bit different. See their doc for more information.

How to log events

Required Fields

- any missing fields will prevent the event from being logged in Snowflake
- unless explicitly mentioned, the field is required in both frontend and backend events

<details>
<summary>Required fields for the user-events table</summary>

- **productArea**
- **featureArea**
- **eventArea**
- **actionType** (backend only)
- **eventName**
- **elementLocation** (frontend only)
- **elementName** (frontend only)
- **eventFull** - a concatenation of a number of fields
  - Frontend: `{eventArea}.{elementLocation}.{eventName}.{elementName}`
  - Backend: `{eventArea}.{featureArea}.{eventName}.{actionType}`
  - this is constructed on the backend, so you just need to make sure the fields are passed correctly
- **properties** - additional data to be associated with the event (ensure there is no PII being logged here)
</details>

Sample frontend event in frontend team repo

```
BiEventLogger.getInstance()?.push({
  eventTimestamp: Date.now(),
  eventData: {
    featureArea: 'sampleFeatureArea',
    elementLocation: 'sampleElementLocation',
    elementName: 'sampleElementName',
    eventName: 'sampleEventName',
```

```
    properties: {
      someProperty: 'samplePropertyValue',
    },
  },
})
```

- note that productArea, eventArea, and eventFull are not explicitly defined in this event. This is because they are appended to the event in our logger class, defined here.

Sample backend event in backend team repo

```
eventData = {
  product_area: 'sampleProductAreaBackend',
  feature_area: 'sampleFeatureAreaBackend',
  event_area: 'sampleEventAreaBackend',
  action_type:  'sampleActionTypeBackend',
  event_name: 'sampleEventNameBackend',
  properties: {
    "someProperty": 'sampleBackendPropertyValue',
  }
}

with_event_logging(current_user: user, current_organization: organization, **eventData)
```

- note that eventFull is not explicitly defined in this event. This is because it is concatenated and added in the EventLogging Ruby module defined here.

How to view event logs

Before deployment:

- backend: ssh into your stepbox environment and run the command `docker logs -f team-repo-web` to connect to the backend logs. You can then trigger the event via the UI or api call and view the logs to see if the correct data was sent. Stepbox details can be found here.
- frontend: open the Chrome dev tools and then open the network tab. User events will be sent via the `user-events` calls

After deployment:

- verify the events are being populated in Snowflake correctly. View the user-events table here.