Nice — I’ll walk you through **everything** about Apptimize *Feature Variables* in simple words, show how they differ from feature flags, and give a **complete, copy-pasteable React example** for your file-upload feature. I’ll also explain the dashboard steps (with what each screen does) so you can set this up end-to-end. Let’s go. 🚀

**TL;DR (one-line)**

**Feature variables** are values you manage in the Apptimize dashboard (numbers, strings, arrays, booleans, etc.) that your app reads at runtime to change how a feature behaves — without redeploying. They are different from **feature flags** (ON/OFF) because variables let you change *parameters* (limits, types, text, percentages) not just enable/disable. ([Airship](https://apptimize.com/docs/featurevariables.html))

**1) What are Feature Variables? (plain language)**

* Think of a **feature flag** as a light switch: on/off.
* Think of a **feature variable** as the thermostat for that room: it sets the temperature (e.g., maxFileSize = 20), or the color (primaryColor = "blue"), or which file types are allowed (["pdf","jpg"]).
* You define the variable in Apptimize, give it variants (possible values), and assign those variants to groups of users (segments). The app asks Apptimize what value it should use for the current user and behaves accordingly — live. ([Airship](https://apptimize.com/docs/featurevariables.html))

**2) Platforms & data types (short)**

* **Works on many platforms** — iOS, Android, REST/server, React / JS, Java, Node, Python, Roku, etc. One variable can be used across platforms. ([Airship](https://apptimize.com/docs/featurevariables.html))
* **Supported data types:** string, integer (int), double / float, boolean, array, dictionary (map/object). Arrays/dictionaries may contain any supported types. Use the type that matches your code. ([Airship](https://apptimize.com/docs/featurevariables.html))

**3) How Feature Variables relate to A/B experiments & feature flags**

* You can use feature variables inside A/B experiments (dynamic variable experiments) to try different values and measure results. Example: test maxFileSize = 10 vs 20 and see which group uploads more files. ([Airship](https://apptimize.com/docs/examples/dynvar-experiments.html?utm_source=chatgpt.com))
* You can also create variables from winning segments of experiments — then promote that value as a targeted feature variable. ([Airship](https://apptimize.com/docs/featurevariables.html))

**4) Dashboard flow — step by step (what each step means)**

Apptimize breaks this into four main steps on the **Feature Variables** project type in the dashboard: Set up the feature → Add dynamic variables → Set up audience segments → Preview & Launch. I’ll explain each and what you should enter.

1. **Set up the feature (Details)**
   * Click **Create → Feature Variables** in the dashboard.
   * Give your feature a name (e.g., file-upload) and notes explaining the goal (e.g., “experiment file size & types for free vs. premium users”). You can add tags for filtering. ([Airship](https://apptimize.com/docs/featurevariables.html))
2. **Add dynamic variables (Configure)**
   * Reuse existing variables or **Define a New Variable** (give it a stable name and choose its data type). Example variables for file upload:
     + file\_upload\_enabled — **boolean**
     + file\_upload\_max\_mb — **integer**
     + file\_upload\_allowed\_types — **string array** (or string CSV if you prefer)
   * After selecting/creating variables you set **variants** (each variant is a bundle of values for the variables). The UI also shows the **Integration** code snippet (how to read these variables from code). ([Airship](https://apptimize.com/docs/featurevariables.html))
3. **Set up audience segments (Target / allocate)**
   * For each variant, choose which users see it using filters (device, OS, custom attributes like subscription=premium, country, etc.). You can also set percentage allocations within a segment.
   * There is an **Everyone Else** segment you can use as a catch-all (set its percentage to 0 to exclude everyone outside your segments). This is where you control rollout percentages. ([Airship](https://apptimize.com/docs/featurevariables.html))
4. **Preview & Launch**
   * Preview variants on a device or with pilot groups; the Preview & Launch screen lets you schedule start/end times and repetition. If any setup errors exist, the UI shows warnings and prevents launch. Click **Launch** to start delivery. You can stop a live feature manually. ([Airship](https://apptimize.com/docs/featurevariables.html))

**5) Practical example (your file upload feature) — Dashboard plan**

Create a **Feature**: file-upload

**Variables (name / type / default):**

* file\_upload\_enabled — boolean — default false
* file\_upload\_max\_mb — int — default 10
* file\_upload\_allowed\_types — string array — default ["pdf","jpg"]

**Variants (examples):**

* **Variant A** (Free users): file\_upload\_enabled = true, file\_upload\_max\_mb = 10, file\_upload\_allowed\_types = ["pdf","jpg"]
* **Variant B** (Premium users): file\_upload\_enabled = true, file\_upload\_max\_mb = 50, file\_upload\_allowed\_types = ["pdf","jpg","png","mp4"]
* **Variant C** (Off): file\_upload\_enabled = false (for a control group)

**Audience targeting:**

* Variant A → segment subscription == free (100% of that segment)
* Variant B → segment subscription == premium (100% of that segment)
* Variant C → Everyone Else (0% if you want no one else to see it) ([Airship](https://apptimize.com/docs/featurevariables.html))

**Preview + Launch:** preview in a test device or pilot group before full rollout. ([Airship](https://apptimize.com/docs/featurevariables.html))

**6) React integration — full example (install + init + component)**

Below is a complete example using the **official Apptimize Web SDK** (@apptimize/apptimize-web-sdk) that shows initialization, waiting for the SDK, reading variables (including arrays), reacting to metadata updates, and doing safe server validation reminders.

**Install SDK**

npm i @apptimize/apptimize-web-sdk

(you can also drop the bundle script into a non-node web app). ([Airship](https://apptimize.com/docs/installation/javascript-installation.html))

**apptimizeClient.js — initialization helper**

// apptimizeClient.js

import Apptimize from '@apptimize/apptimize-web-sdk';

export function initApptimize({ appKey, userId = null, customAttributes = {} }) {

// Optional: set a stable customer user id so the user gets the same variant across devices

if (userId && Apptimize.setCustomerUserId) {

Apptimize.setCustomerUserId(userId);

}

// Optional: set custom attributes used for targeting (e.g., subscription: "premium")

if (customAttributes && Apptimize.setCustomAttributes) {

Apptimize.setCustomAttributes(customAttributes);

}

// Optional: set a callback to know when Apptimize is ready

if (Apptimize.setOnApptimizeInitializedCallback) {

Apptimize.setOnApptimizeInitializedCallback(() => {

console.log('Apptimize initialized');

// you can track an event too

if (Apptimize.track) Apptimize.track('Apptimize Initialized');

});

}

// Finally, start Apptimize with your key

Apptimize.setup(appKey);

// Helpful for local testing: force a metadata update (if you need it)

// Apptimize.updateApptimizeMetadataOnce && Apptimize.updateApptimizeMetadataOnce();

}

setCustomerUserId and setCustomAttributes help keep user assignment consistent and enable targeting. Use them before setup if possible or immediately after. ([Apptimize](https://sdk.apptimize.com/js-client/docs/apptimize/Apptimize.html))

**FileUpload.jsx — component that reads feature variables**

// FileUpload.jsx

import React, { useEffect, useState } from 'react';

import Apptimize from '@apptimize/apptimize-web-sdk';

export default function FileUpload() {

const [ready, setReady] = useState(false);

const [isEnabled, setIsEnabled] = useState(false);

const [maxFileSizeMB, setMaxFileSizeMB] = useState(10);

const [allowedTypes, setAllowedTypes] = useState(['pdf']); // default

// Helper: read variables from Apptimize and update state

const refreshVariables = () => {

try {

// Use the exact variable names you created in the dashboard

const enabled = Apptimize.getBool

? Apptimize.getBool('file\_upload\_enabled', false)

: false;

const maxMB = Apptimize.getInt

? Apptimize.getInt('file\_upload\_max\_mb', 10)

: 10;

// getStringArray returns an array (if you defined array type in dashboard)

const types = Apptimize.getStringArray

? Apptimize.getStringArray('file\_upload\_allowed\_types', ['pdf'])

: ['pdf'];

setIsEnabled(Boolean(enabled));

setMaxFileSizeMB(Number(maxMB));

setAllowedTypes(Array.isArray(types) ? types.map(t => t.toLowerCase()) : ['pdf']);

} catch (err) {

console.error('Error reading Apptimize variables', err);

}

};

useEffect(() => {

// When Apptimize is initialized, refresh variables

const onInit = () => {

setReady(true);

refreshVariables();

};

// Some SDKs expose setOnApptimizeInitializedCallback (JS doc example)

if (Apptimize.setOnApptimizeInitializedCallback) {

Apptimize.setOnApptimizeInitializedCallback(onInit);

} else {

// fallback: try to refresh immediately (if setup completed earlier)

refreshVariables();

setReady(true);

}

// If metadata updates happen in the dashboard, refresh variables live

if (Apptimize.setOnMetadataUpdatedCallback) {

Apptimize.setOnMetadataUpdatedCallback(refreshVariables);

}

return () => {

// cleanup - clear callbacks if API allows

if (Apptimize.setOnApptimizeInitializedCallback) Apptimize.setOnApptimizeInitializedCallback(null);

if (Apptimize.setOnMetadataUpdatedCallback) Apptimize.setOnMetadataUpdatedCallback(null);

};

}, []);

const handleFileSelect = (evt) => {

const file = evt.target.files?.[0];

if (!file) return;

const ext = file.name.split('.').pop().toLowerCase();

// client-side validation (good UX)

if (!allowedTypes.includes(ext)) {

alert(`Type not allowed. Allowed: ${allowedTypes.join(', ')}`);

return;

}

if (file.size > maxFileSizeMB \* 1024 \* 1024) {

alert(`File too large. Max ${maxFileSizeMB} MB`);

return;

}

// TODO: upload to server; always validate again server-side!

alert(`"${file.name}" accepted — ready to upload`);

};

if (!ready) return <div>Loading...</div>;

if (!isEnabled) {

return <div>File upload is currently disabled for you.</div>;

}

return (

<div>

<h3>Upload a file</h3>

<input type="file" onChange={handleFileSelect} />

<p>Allowed: {allowedTypes.join(', ')} | Max: {maxFileSizeMB} MB</p>

</div>

);

}

**Notes on the code:**

* Use the **exact variable names** you used in the Apptimize dashboard.
* We use getBool, getInt, getStringArray which are available in the JS/React SDK. These calls accept a default value so your app has sensible fallbacks if the dashboard variable is missing. ([Apptimize](https://sdk.apptimize.com/js-client/docs/apptimize/Apptimize.html))
* We set callbacks to refresh values when metadata updates so changes in the dashboard can show without a full page reload. ([Apptimize](https://sdk.apptimize.com/js-client/docs/apptimize/Apptimize.html))

**7) Testing & rollout tips**

* **Preview on device / pilot groups:** Use the dashboard Preview & Launch to test on real devices and pilot cohorts before full rollout. ([Airship](https://apptimize.com/docs/featurevariables.html))
* **Use setCustomerUserId + setCustomAttributes** to make segmentation deterministic (e.g., subscription = premium) so users consistently get the same variant. ([Apptimize](https://sdk.apptimize.com/js-client/docs/apptimize/Apptimize.html))
* **Use Everyone Else carefully** — by default it catches users not matched by other segments; set its percentage to zero if you want to exclude uncovered users. ([Airship](https://apptimize.com/docs/featurevariables.html))
* **Preview/force update for dev:** Apptimize.updateApptimizeMetadataOnce() (if available on SDK) can fetch latest metadata for quick testing. Also the dashboard has device preview and pilot features. ([Apptimize](https://sdk.apptimize.com/js-client/docs/apptimize/Apptimize.html))

**8) Important best practices & cautions**

* **Server-side validation is essential.** Client feature variables control UI/UX — but *never* trust client checks for security or billing logic. Always validate file size, file type, auth, and quotas on the server. (Best practice)
* **Name variables clearly and consistently** (e.g., file\_upload\_max\_mb) so their purpose is obvious and reuse is safer.
* **Assign sensible defaults in code** so users not covered by experiments still have expected behavior.
* **Monitor metrics** (uploads completed, failures, errors) when testing different variants so you don’t accidentally degrade experience. Use Apptimize experiments to gather that data. ([Airship](https://apptimize.com/docs/examples/dynvar-experiments.html?utm_source=chatgpt.com))

**9) Quick checklist to implement (summary)**

1. Create Feature Variables project → name file-upload. ([Airship](https://apptimize.com/docs/featurevariables.html))
2. Define variables: file\_upload\_enabled (bool), file\_upload\_max\_mb (int), file\_upload\_allowed\_types (string array). ([Airship](https://apptimize.com/docs/featurevariables.html))
3. Create variants and target segments (free vs premium) with allocation percentages. ([Airship](https://apptimize.com/docs/featurevariables.html))
4. Install SDK in your React app (npm i @apptimize/apptimize-web-sdk) and initialize using Apptimize.setup(appKey). Wait for setOnApptimizeInitializedCallback before showing variant UI to avoid flicker. ([Airship](https://apptimize.com/docs/installation/javascript-installation.html))
5. Read variables with getBool/getInt/getStringArray in your component and apply them. ([Apptimize](https://sdk.apptimize.com/js-client/docs/apptimize/Apptimize.html))
6. Preview, pilot, then Launch. Monitor & iterate. ([Airship](https://apptimize.com/docs/featurevariables.html))

If you want, I can:

* Turn the FileUpload example above into a tiny working repo scaffold (React + example) you can run locally, or
* Give the exact steps & screenshots for the Apptimize dashboard flows (Create → Configure → Audience → Preview) tailored to your feature names, or
* Provide a mock wrapper to develop locally without the Apptimize SDK (so you can build UI/UX before dashboard setup).

Which of those would help most next?

Absolutely — this part is **super important** because it decides **who sees which variant** of your feature.  
Let me break it down **step by step, in very simple words**, and then we’ll apply it to your “1000 employees/users” case.

**What is Audience Segmentation (Targeting / Allocation)?**

When you create a **feature with variants**, you need to decide:

* **Who should get Variant 1?**
* **Who should get Variant 2?**
* **Who should not get the feature at all?**

This is what **audience segments** do.

**How It Works (Simplified)**

1. **Define Filters**  
   Choose which users should be in a segment using conditions like:
   * Device type (iOS, Android, Web)
   * OS version
   * Country / Region
   * Custom attributes you pass to Apptimize (e.g. subscription=premium, role=manager)
2. **Assign Variants to Segments**  
   For each segment, decide which variant they see.
3. **Allocate Percentages**  
   If you don’t want *all* users in that segment to get the feature, you can roll it out gradually.  
   Example:
   * Segment: "Premium Users"
   * Variant 1 (new feature): 20% of users
   * Variant 2 (control): 80% of users
4. **Everyone Else Segment**  
   If a user does not match any segment you defined, they fall into **Everyone Else**.
   * You can give them a default variant (usually Control)
   * Or set their percentage to **0%** so they get nothing (excluded from experiment)

**Example: 1000 Employees**

Imagine you have **1000 employees** and you are rolling out a **new file upload feature** with two variants:

* **Variant 1**: 10 MB limit
* **Variant 2**: 50 MB limit

Here’s how you can do it:

| **Segment Name** | **Filter** | **Variant** | **Allocation** |
| --- | --- | --- | --- |
| Beta Testers | role = "IT" (50 employees) | Variant 1 | 100% |
| Managers | role = "Manager" (100 employees) | Variant 2 | 50% (only half managers see it) |
| Everyone Else | (All remaining 850 employees) | Control (feature off) | 100% |

So:

* First 50 employees (IT) → Always see Variant 1
* Half of Managers → See Variant 2, half → see Control
* Rest of employees → See Control (feature OFF)

**Why Percentages Are Useful**

* You can **gradually roll out a risky feature** (e.g., start with 5%, then increase to 50%, then 100%)
* You can do **A/B testing** (e.g., compare 2 different upload limits)
* You can **exclude** everyone except a small test group

**How You Assign to 1000 Users**

1. **Pass User Attributes to Apptimize in Code**  
   Example in React:
2. import { Apptimize } from "apptimize-sdk";
3. Apptimize.updateUserAttributes({
4. employeeId: "123",
5. role: "Manager",
6. location: "India",
7. });

This lets Apptimize know which segment this user belongs to.

1. **Create Segments in Apptimize Dashboard**
   * Segment 1: role = "IT"
   * Segment 2: role = "Manager"
   * Segment 3: Everyone Else
2. **Assign Variants & Percentages**
   * Segment 1 → Variant 1 (100%)
   * Segment 2 → Variant 2 (50%)
   * Everyone Else → Control (100%)
3. **Launch**  
   Now, each user who opens the app will automatically get the variant based on:
   * Their attributes
   * The percentage allocation

**Simple Analogy**

Think of **audience segments** like **buckets of users**.  
You decide:

* Which bucket each user falls into (filters)
* What flavor of the feature each bucket gets (variant)
* What share of that bucket should get it (percentage)

Would you like me to show you a **visual rollout plan** (like a small diagram/table) for 1000 employees — showing exactly how they are split and which variant each group gets?  
(It will make it crystal clear how targeting and percentages work.)

Got it — this is a great question because **some targeting conditions you configure entirely in the Apptimize dashboard**, but others require you to **send user data from your code to Apptimize** first.

Let’s break it down step by step.

**1. Conditions You Can Configure Directly in the Apptimize Dashboard**

Apptimize automatically collects **some attributes** for every user session.  
So you can directly create segments using these (no coding needed):

✅ **Directly Configurable in Apptimize UI (No Code Required):**

* **Device Type** (iOS, Android, Web)
* **OS Version**
* **App Version** (if available)
* **Country/Region** (derived from IP)

**Steps in Apptimize UI:**

1. Go to **Audience Tab** for your feature.
2. Click **Add Segment** → name it (e.g. "Android 13 Users").
3. Choose **Filter** → Select Device Type = Android.
4. Add another filter if needed → OS Version = 13.
5. Assign this segment to a Variant and set allocation percentage.
6. Save → Launch.

This works out-of-the-box, no code changes required.

**2. Conditions That Require Sending Data from Code (Custom Attributes)**

If you want to target:

* Specific users by **SSO IDs**
* Employees by **role, department, subscription plan**
* Users by **email domain**, **employee ID**, etc.

You need to **send these attributes from your app** to Apptimize first.

**React Example (Passing SSO IDs / Attributes)**

Let’s say you have a user object after login:

const user = {

ssoId: "EMP12345",

role: "Manager",

department: "HR",

location: "India",

};

You send these attributes to Apptimize like this:

import { Apptimize } from "apptimize-sdk";

// Send user attributes to Apptimize

Apptimize.updateUserAttributes({

ssoId: user.ssoId, // Unique ID for targeting specific users

role: user.role, // Use in segments like role = Manager

department: user.department,

location: user.location, // You can filter by this in dashboard

});

Now, these attributes are available for segmentation in the dashboard.

**Steps in Apptimize Dashboard (After Sending Attributes)**

1. Go to **Audience Tab** of your feature.
2. Click **Add Segment** → Name it "Managers in HR".
3. Choose **Custom Attribute** filter → Select role = Manager.
4. Add another filter → department = HR.
5. Assign Variant → Set percentage to 100% (or less if gradual rollout).
6. Save & Launch.

Now only HR Managers will see that variant.

**Targeting by Specific SSO IDs**

You can even target **specific users** by ID.

Example: Roll out a feature to only 5 employees with known SSO IDs.

**Code:**

Apptimize.updateUserAttributes({

ssoId: user.ssoId

});

**In Apptimize:**

* Create Segment
* Add Filter: ssoId IN ["EMP12345", "EMP67890", "EMP24680", "EMP13579", "EMP11223"]
* Assign them Variant 1 (feature enabled)

This is perfect for pilot testing with a small group of employees.

**Summary Table**

| **Condition** | **Where to Configure** | **Steps** |
| --- | --- | --- |
| Device Type | Apptimize UI (no code) | Add segment → filter → Device Type = Android/iOS/Web |
| OS Version | Apptimize UI (no code) | Add filter → OS Version >= 13 |
| Country/Region | Apptimize UI (no code) | Add filter → Country = IN |
| App Version | Apptimize UI (no code) | Add filter → App Version >= 2.0.0 |
| **SSO ID / Employee ID** | **Code + UI** | 1) Send ssoId via updateUserAttributes() 2) Filter ssoId IN [...] |
| **Role / Department / Subscription** | **Code + UI** | 1) Send attribute via code 2) Filter in UI role = Manager |

**Best Practice for 1000 Employees**

If you have SSO IDs for all 1000 employees:

1. **Send ssoId + role + location to Apptimize** on login.
2. **Create segments in dashboard** based on:
   * Specific users (ssoId list)
   * Roles (Managers, Developers, HR)
   * Location (India, US)
3. **Allocate rollout percentages** for gradual release.

This way you have **fine control** — you can target exactly who gets the feature.

Would you like me to give you a **sample rollout plan for 1000 employees using SSO IDs** — including JSON-like example of how you would upload or manage those segments in Apptimize?  
(It can help you visualize what to pass from code + what to configure in dashboard.)

# FINAL STEPS

Perfect! Let's put this all together in **clear step-by-step instructions** for your use case.  
We'll configure Apptimize from scratch, set up a feature variable (isEnabled), create a segment based on **SSO IDs**, and then use it in React code.

**✅ Goal**

We want:

* A single **feature variable** (isEnabled) that acts like a feature flag.
* **Only users with certain SSO IDs** can use the file upload feature.
* Everyone else will not see file upload.

**🛠 Step 1: Set Up Feature Variable in Apptimize**

1. **Log into Apptimize Dashboard**
   * Go to your Apptimize project.
2. **Create a New Feature**
   * Click **Create → Feature Variables**.
   * Name it: File Upload
   * Add description: "Controls whether file upload is enabled for users"
   * Click **Continue to Configure**.
3. **Add Dynamic Variable**
   * Click **Define a New Variable**
   * Name: isEnabled
   * Type: **Boolean**
   * Default value: false (feature OFF by default)
   * Click **Save**.
4. **Create Variants**
   * Variant 1 → isEnabled = true (feature ON)
   * Control Variant (or Everyone Else) → isEnabled = false (feature OFF)
   * Click **Continue to Target**.

**🎯 Step 2: Create Audience Segment**

1. **Go to Audience Tab**
   * Click **Add Segment**
   * Name the segment: My New Segment
   * Select **Custom Attribute**
   * Choose attribute: ssoId
   * Condition: IN
   * Add values: "EMP1001", "EMP1002", "EMP1003" (replace with your real SSO IDs)
   * Click **Save**.
2. **Assign Variant to Segment**
   * Choose Variant 1 (where isEnabled=true) for this segment.
   * Set allocation = 100% (all these users will get the feature).
   * For **Everyone Else**, select Control variant (where isEnabled=false).
3. **Save & Launch**
   * Click **Continue to Launch**
   * Review setup → Click **Launch**.

**💻 Step 3: Send SSO IDs from Your Code**

To let Apptimize know which user is using the app, you must send **SSO ID** when the user logs in.

**Example in React:**

import { Apptimize } from "apptimize-sdk";

// Call this after user logs in and you have their SSO ID

function identifyUser(user) {

Apptimize.updateUserAttributes({

ssoId: user.ssoId, // This should match the IDs you put in Apptimize

});

}

This ensures that Apptimize knows which **segment** this user belongs to.

**🎨 Step 4: Use the Feature Variable in React**

import React from "react";

import { Apptimize } from "apptimize-sdk";

const FileUpload = () => {

// Feature variable from Apptimize

const isEnabled = Apptimize.getBoolean("isEnabled");

if (!isEnabled) {

return <p>File upload is currently disabled for your account.</p>;

}

const handleFileUpload = (file) => {

console.log("File selected:", file.name);

// Add your upload logic here

};

return (

<div>

<h3>Upload your file</h3>

<input type="file" onChange={(e) => handleFileUpload(e.target.files[0])} />

</div>

);

};

export default FileUpload;

**🧠 How It Works**

* When a user logs in, you call Apptimize.updateUserAttributes({ ssoId: user.ssoId })
* Apptimize checks the dashboard rules:
  + If user's ssoId is in "My New Segment" → They get isEnabled = true
  + Otherwise → They get isEnabled = false
* Your React code simply calls Apptimize.getBoolean("isEnabled")
  + Returns true → Show file upload
  + Returns false → Show message that upload is disabled

**📊 Example for 1000 Employees**

| **Segment** | **Condition** | **Variant** | **Feature State** |
| --- | --- | --- | --- |
| My New Segment | ssoId IN ["EMP1001","EMP1002",...] | isEnabled=true | File upload ON |
| Everyone Else | No match | isEnabled=false | File upload OFF |

This allows **only selected employees** to access the feature.

**✅ Benefits**

* **No app redeploy needed** — you can add/remove SSO IDs anytime from dashboard.
* **Safe rollout** — start with small group (e.g., 10 users), then expand to all.
* **Control at runtime** — you can turn the feature OFF immediately if needed.

Would you like me to give you a **JSON-like sample structure** of the segment you’d configure in Apptimize (so you have a visual reference of how it looks in the dashboard)?  
This will make it super easy for you when you actually configure it.