Flow Meta Conventions

Read these Resources first

1. The official Flows best practice doc. Note we agree on most things. Specifically the need to plan out your Flow first.
2. The Flows limits doc. If you don't know the platform limits, how can you build around them?
3. The Transactions limits doc. Same as above, gotta know limits to play around them.
4. The What Automation Do I Create Flowchart. Not everything needs to be a Flow.
5. The Record-Triggered Automation Guide, if applicable.

Best Practices

These are general best practices that do not pertain to individual flows but more to Flows in regards to their usage within an Organization.

On Permissions

Flows should **ALWAYS** execute in the smallest amount of permissions possible for it to execute a task.
Users should also ideally not have access to Flows they don't require.
Giving Setup access so someone can access `DeveloperName` is bad, and you should be using custom labels to store the ids and reference that instead, just to limit setup access.

> **Use System mode sparingly. It is dangerous.**
> If used in a Communities setting, I REALLY hope you know why you're exposing data publicly over the internet or that you're only committing information with no GETs.

> Users can have access granted to specific Flows via their Profiles and Permission Sets, which you should really be using to ensure that normal users can't use the Flow that massively updates the client base for example.

Record-Triggered Flows, and Triggers should ideally not coexist on the same object in the same Context.

> "Context" here means the APEX Trigger Context. Note that not all of these contexts are exposed in Flow:
>
> **- Screen Flows** execute outside of these contexts, but Update elements do not allow you to carry out operations in the `before` context.
> **- Record Triggered Flow** execute either in `before` or `after` contexts, depending on what you chose at the Flow creation screen (they are named "Fast Record Updates" and "Other Objects and Related Actions", respectively, because it seems Salesforce and I disagree that training people on proper understanding of how the platform works is important).

The reason for the "same context" exclusivity is in case of multiple Flows and heavy custom APEX logic: in short, unless you plan *explicitly* for it, the presence of one or the other forces you to audit both in case of additional development, or routine maintenance.
You could technically leverage Flows and APEX perfectly fine together, but if you have a `before` Flow and a `before` Trigger both doing updates to fields, and you accidentally reference a field in both... debugging that is going to be fun.

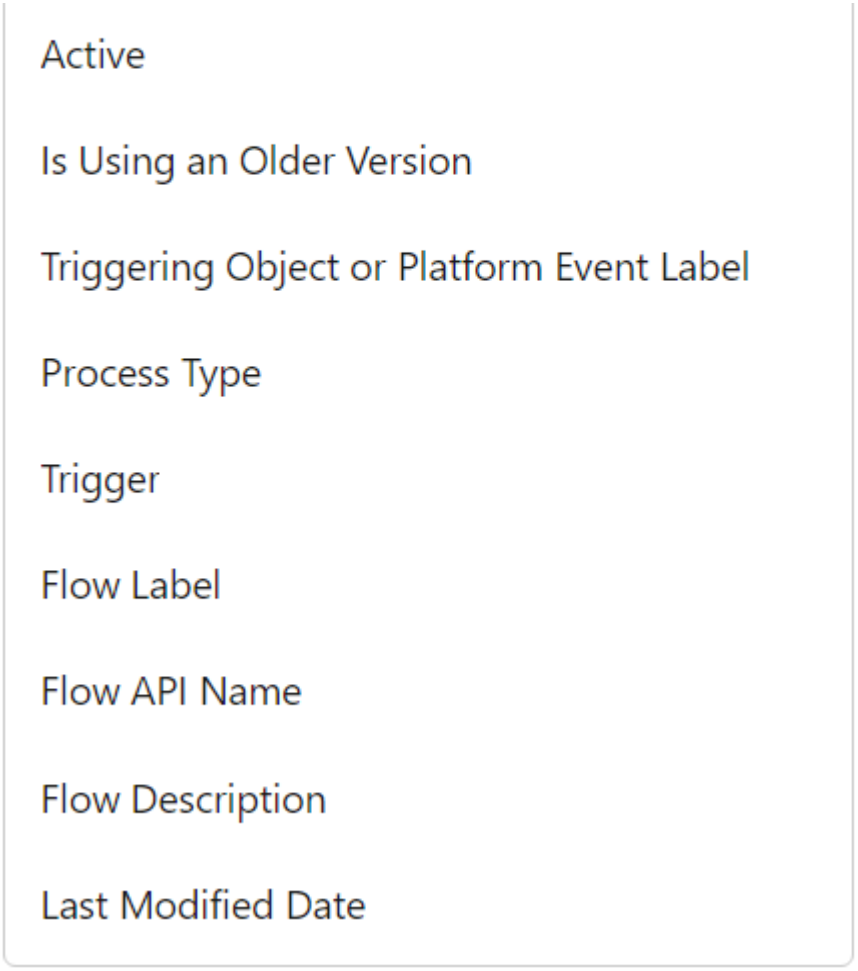
So if you start relying on APEX Triggers, while this doesn't mean you have to change all the Flows to APEX logic straight away, it does mean you need to plan for a migration path.

> In the case were some automations need to be admin editable but other automations require custom code, you should be migrating the Triggers to APEX, and leveraging sub-flows which get called from your APEX logic.

Flow List Views should be used to sort and manage access to your Flows easily

The default list view is not as useful as others can be.
We generally suggest doing at minimum one list view, and two if you have installed packages that ship Flows:

- ◦ One List View that shows all active flows, with the following fields displayed: `Active, Is Using an Older Version, Triggering Object or Platform Event Label, Process Type, Trigger, Flow Label, Flow API Name, Flow Description, Last Modified Date`



This will allow you to easily find your flows by Object (apart from Scheduled Flows or Sub-Flows, but this is handled via Naming Conventions), see if you started working on a Flow but didn't activate the last version, and view the beautiful descriptions that you will have set everywhere.

- One List View that shows all Package flows, which contains `Active, Is Using an Older Version, Flow Namespace, Overridable, Overridden By, Overrides`
  This allows you to easily manage your updates to these Flows that are sourced from outside your organization.

Flows are considered Code for maintenance purposes

> Do NOT create or edit Flows in Production, especially a Record-Triggered flow.
> If any user does a data load operation and you corrupt swaths of data, you will know the meaning of “getting gray hairs”, unless you have a backup - which I am guessing you will

> not have if you were doing live edits in production.

No, this isn't a second helping of our note in the General Notes.
This is about your Flows - the ones you built, the ones you know very well and are proud of.
There are a **swath** of reasons to consider Flows to be Code for maintenance purposes, but in short:

- if you're tired, mess up, or are otherwise wrong, Production updates of Flows can have HUGE repercussions depending on how many users are using the platform, and how impactful your Flow is
- updating Flows in Production will break your deployment lifecycle, and cause problems in CI/CD tools if you use them
- updating Flows in Production means that you have no safe reproducibility environment unless you refresh a sandbox
- unless you know every interaction with other parts of the system, a minor update can have impact due to other automation - whether it be APEX, or other Flows.

In short - it's a short and admin-friendly development, but it's still development.

On which automation to create

In addition to our (frankly not very beautiful Flowchart), when creating automations, the order of priority should be:

1. **Object-bound, BEFORE Flows**
   These are the most CPU-efficient Flows to create.
   They should be used to set information that is required on Objects that are created or updated.
2. **User-bound Flows**
   Meaning Screen flows. These aren't tied to automation, and so are very CPU efficient and testable.
3. **Object-bound, Scheduled Flows**
   If you can, create your flows as a Schedule rather than something that will spend a lot of time waiting for an action - a great example of this are scheduled emails one month after something happens.
   Do test your batch before deploying it, though.
4. **Object-bound, AFTER Flows**
   These are last because they are CPU intensive, can cause recursion, and generally can have more impact in the org than other sources of automation.

On APEX and LWCs in Flows

- **APEX or LWCs that are specifically made to be called from Flows should be clearly named and defined in a way that makes their identification and maintenance easier.**

- **Flows that call APEX or LWCs are subject to more limits and potential bugs than fully declarative ones.**
  When planning to do one, factor in the maintenance cost of these parts.
  Yes, this absolutely includes actions and components from the wonderful UnofficialSF. If you install unpackaged code in your organization, YOU are responsible for maintaining it.
- On a related note, **Don't use non-official components without checking their limits.**
  Yes UnofficialSF is great, and it also contains components that are not bulkified or contain bugs.

> To reiterate, if you install unpackaged code in your organization, YOU are responsible for maintaining it.

Flow Testing and Flow Tests

If at all possible, **Flows should be Tested**. This isn't always possible because of these considerations, (which aren't actually exhaustive - I have personally seen edge cases where Tests fail but actual function runs, because of the way Tests are build, and I have also seen deployment errors linked to Tests). Trailheads exist to help you get there.

> A Flow Test is **not** just a way to check your Flow works. A proper test should:
> - Test the Flow works
> - Test the Flow works in other Permission situations
> - Test the Flow **doesn't** work in critical situations you want to avoid [if you're supposed to send one email, you should *probably catch the situation where you're sending 5* mil]
> ... and in addition to that, a proper Flow Test will warn you **if things stop working down the line.**
>
> Most of these boilerplates are *negative bets against the future* - we are expecting things to break, people to forget configuration, and updates to be made out of process. Tests are a way to mitigate that.

> We currently consider Flow Tests to be "acceptable but still bad", which we expect to change as time goes on, but as it's not a critical feature, we aren't sure when they'll address the current issues with the tool.

Note that proper Flow Testing will probably become a requirement at some point down the line.

On Bypasses

Flows, like many things in Salesforce, can be configured to respect Bypasses.

In the case of Flows, you might want to call these "feature flags".

This is a GREAT best practice, but is generally overkill unless you are a very mature org with huge amounts of processes.

Revision #15
Created 2021-01-13 16:27:36 UTC by Windyo
Updated 2023-08-28 07:06:59 UTC by Windyo