Flow Structural Conventions - Record-Triggered

As detailed in the General Notes section, these conventions are heavily opinionated towards maintenance and scaling in large organizations. The conventions contain:

- a "common core" set of structural conventions that apply everywhere
- conventions for Record Triggered Flows specifically (this page!)
- conventions for Scheduled Flows specifically

These Record-Triggered Conventions expect you to be familiar with the tools at your disposal to handle order of execution and general Flow Management, including the Flow Trigger Explorer, Scheduled Paths, Entry Criteria (linked: a page that should document entry criteria but doesn't).

> This page directly changes conventions that were emitted by SFXD in 2019, and reiterated in 2021.
> This is because the platform has changed since then, and as such we are recommending new, better, more robust way to build stuff.
> If you recently used our old guides - they are still fine, we just consider this new version to be better practice.

Record-Triggered Flow Design

Before Creating a Flow

Ensure there are no sources of automation touching the Object or Fields

If the same field is updated in another automation, default to that automation instead, or refactor that automation to Flow.
If the Object is used in other sources of automation, you might want to default to that as well, or refactor that automation to Flow, unless you can ensure that both that source of automation and the Flow you will create will not cross-impact each other.

> You can leverage "where is this used" in sandbox orgs to check if a field is alreadyreferenced in a Flow - or take the HULK SMASH approach and just create a new sandbox, and try to delete the field. If it fails deletion, it'll tell you where it is referenced.

Verify the list of existing Flows and Entry Criterias you have

You don't want to have multiple sources of the same entry criteria in Flows because it will make management harder, and you also don't want to have multiple Flows that do almost the same thing because of scale.

> Identifying if you can refactor a Flow into a Subflow that will be called from multiple places is best done before trying to build anything.

Ask yourself if it can't be a Scheduled Flow instead

Anything date based, anything that has wait times, anything that doesn't need to be at the instant the record changes status but can instead wait a few hours for the flow to run - all these things can be scheduled Flows. This will allow you to have better save times on records.

Prioritize BEFORE-save operations whenever possible

This is more efficient in every way for the database, and avoids recurring SAVE operations.
It also mostly avoid impacts from other automation sources (apart from Before-Save APEX).
Designing your Flow to have the most possible before-save elements will save you time and effort in the long run.

Check if you need to update your bypasses

Specifically for Emails, using bypasses remains something that is important. Because sending emails to your entire database when you're testing stuff is probably not what you want.

Consider the worst case

Do not build your system for the best user but the worst one. Ensure that faults are handled, ensure that a suser subject to every single piece of automation still has a usable system, etc.

On the number of Flows per Object and Start Elements

- **Before-Save Flows**
  Use **as many before-save flows as you require**.
  You *should*, but do not *have to*, set Entry Conditions on your Flows.
  Each individual Flow should be tied to a **functional Domain**, or a specific **user story**, as you see most logical. The order of the Flows in the Flow Trigger Explorer should not matter, as a single field should **never** be referenced in multiple before save flows as the

target of an assignment or update.
- **After-Save Flows** Use **one** Flow for actions that should trigger without entry criteria, and orchestrate them with Decision elements.
  Use **one** Flow to handle **Email Sends** if you have multiple email actions on the Object and need to orchestrate them.
  Use **as many additional flows as you require, as long as they are tied to unique Entry Criteria**.
  Set the Order of the Flows **manually in the Flow Trigger Explorer** to ensure you know how these elements chain together.Offload any computationally complex operation that **doesnt need to be done immediately to a scheduled** path.

> Entry Criteria specify when a Flow is *evaluated*. It is a very efficient way to avoid Flows triggering unduly and saves a lot of CPU time. Entry Criteria however do require knowledge of Formulas to use fully (the basic "AND" condition doesn't allow a few things that the Formula editor does in fact handle properly), and it is important to note that the entire Flow does not execute if the Entry Criteria isn't met, so you can't catch errors or anything.

To build open what's written above:

- Before-Save flows are very fast, generally have no impact on performance unless you do very weird stuff, and should be easy to maintain as log as you name them properly, even if you have multiple per object. "Tieing" a flow to a Domain or Object means by its name and structure. You can technically do a Flow that does updates both for Sales and Invoicing, but this is generally meh if you need to update a specific function down the line.

> Logical separation of responsabilities is a topic you'll find not only here but also in a lot of development books.

> Before-Save Flows don't actually require an Update element - this is just for show and to allow people to feel more comfortable with it. You can technically just use Assignments to manipulate the `$Record` variable with the same effect. *It actually used to be the only way to do before-save, but was thought too confusing.*

- After-Save flows, while more powerful, require you to do another DML operation to commit anything you are modifying. This has a few impacts, such as the possibility to re-run automtions if you update the record that already triggered your automation. The suggestions we make above are based on the following:
  - Few actions on records should not have entry criteria set. This allows more flows to be present on each object without slowdowns. The limit of One flow is because it should pretty much not exist, or be small.
  - Emails sent from Objects are always stress inducing in case of data loads, and while a proper bypass usage does not require grouping all emails in a Flow,knowing that all email alerts are in a specific place does make maintenance easier.

- Entry-Criteria filtered Flows are quite efficient, and so do not need to be restricted in number anymore.
- Ordering Flows manually is to avoid cases where the order of Flows is unkown, and interaction between Flows that you ahve not identified yields either positive or negative results that can't be reproduced without proper ordering.
- Scheduled Paths are great if you are updating related Objects, sending notifications, or doing any other operation that isn't time-sensitive for the user.

> We used to recommend a single Flow per context. This is obviously no longer the case.
>
> This is because anything that pattern provided, other tools now provide, and do better.
>
> The "One flow per Object pattern" was born because:
> - Flows only triggered in `after` contexts
> - Flows didn't have a way to be orchestrated between themselves
> - Performance impact of Flows was huge because of the lack of entry criteria
>
> None of that is true anymore.
>
> The remnant of that pattern still exists in the "no entry criteria, after context, flow that has decision nodes", so it's not completely gone.
>
> So while the advent of Flow Trigger Explorer was one nail in the coffin for that pattern, the real final one was actual good entry criteria logic.

> Entry Criteria are awesome but are not properly disclosed either in the Flow List View, nor the Start Element. Ensure that you follow proper Description filling so you can in fact know how these elements work, otherwise you will need to open every single Flow to check what is happening.

On Delayed Actions

Flows allows you to do complex queries and loops as well as schedules. As such, there is virtually no reason to use wait elements or delayed actions, unless said waits are for a platform event, or the delayed actions are relatively short.

Any action that is scheduled for a month in the future for example should instead set a flag on the record, and let a Scheduled Flow evaluate the records daily to see if they fit criteria for processing. If they do in fact fit criteria, then execute the action.

A great example of this is Birthday emails - instead of triggering an action that waits for a year, do a Scheduled flow running daily on contacts who's birthday it is. This makes it a lot easier to debug and see what’s going on.

Revision #4
Created 2023-08-10 11:30:39 UTC by Windyo
Updated 2023-08-21 14:58:12 UTC by Windyo