# Versioning in PowerToys

- **Description:** How the PowerToys team will handle versioning and experimental utilities. This document covers the terminology used, tagging, release channels, how to define a version, branch and attributes.
- **Authors:** Clint Rutkas
- **Spec Status:** Draft - 2021.01.04

## Overview of solution and North star

We will migrate to release channels (like Windows and Microsoft Edge), non-overlapping version numbers, and a core of PowerToys that were we can quickly iterate on experiments. We will shift to a system where ever When needed, we'll utilize branches to keep the core stable.

## Current issue: Build process, versioning and crash reports

Right now we have a few different spots we build from and we increased the version numbers in the source tree. When we try to look at crash reports, at times, what we see are actually test candidates and development builds since their version overlaps with a production version number. This leads to pains when we start trying to figure out what failed and how. To correct this, we will shift moving forward on what branches we use and when / who updates `Version.props`.

We also face an issue of our experimental builds take effort to rebase against our main base of code. This impacts our ability to ship timely updates of those releases. We've actually skipped one experimental release due the time it took to rebase against the main core of PowerToys.

To clearly delineate between an experimental build and a stable build, we actually do a full minor release. Example is for stable builds, we are currently using 0.25, 0.27, 0.29, ... and for experimental

## Terminology

We will stick to terminology used by both GitHub and what the APIs provide.

### Release type

GitHub uses the term `Release` and `Pre-release`.

- `Release`: Stable, quality build, minimal bugs, ready for the end-user community to install.
- `Pre-release`: Stable enough to be available for use, but may contain bugs in new release-candidate features, as well as new experimental utilities that may not yet work properly in end-use environments.

### Release channels

A release channel is typically a feature or stability staged release. Microsoft Edge for example does canary, dev, beta, stable. For PowerToys, we'll break it down in to stable, beta, dev, and experimental features. Updates will be locked to that channel.

- Stable: This channel is for features that have passed all verifications and tests, with remaining bugs being minimal and considered acceptable for the end-user community.
- Beta: This channel is for features that are complete, but have not yet passed bug testing and end-use verification.
- Dev: This channel is for the active development of new features that have been committed to by the PowerToys dev team and included in the roadmap.

- Experimental: This channel is for use by the open contribution community to suggest and experiment with new features and utilities with the possibility of being included in future PowerToy releases.

### Tagging

GitHub does allow each release to be tagged. This can follow a pre-defined format that we can use to aid us in doing different release channels. Example `v0.0.1-VideoConference` could represent the Video conference experimental feature.

This tag will also be appended to the title and systray of PowerToys. (Note: The work for adding in the name has **not** been done yet)

### Version number definition

Our software includes .NET based software, we need to adhere to the versioning construct there. Version numbers consist of two to four components: major, minor, build, and revision. The major and minor components are required; the build and revision components are optional, but the build component is required if the revision component is defined. All defined components must be integers greater than or equal to 0. The format of the version number is as follows (optional components are shown in square brackets ([ and ]): This is from docs.microsoft.com

*major.minor*[.*build*[.*revision*]]

| Property | Value |
|---|---|
| Major | 0 |
| Minor | 0 |
| Build | undefined (-1) |
| Revision | undefined (-1) |

How PowerToys will handle version numbers:

- Stable: 0.*minor*[.*build*]. Example: 0.37.3
- Beta: 0.*minor*[.*build*]. Example: 0.37.3
- Experimental: 0.0.*minor*[*build*]. Example: 0.0.373

## Full transition for utilities to be independent of runner.exe

Our long term goal is to make all utilities independent of the runner executable as the hosting executable. While most are, FancyZones and Shortcut guide v1 are tightly coupled. We will still have scenarios where the runner may take a lead for handling activation, such as elevated application shortcuts, but holistically, the utilities should not take a direct dependency.

## Channels, versioning, and tag structure

This would be a breakdown of the 0.37 release and defining structure around it.

| Release Channel | GitHub release type | Tag post-fix | Example tag | Signed Installer available | Installer includes experimental | |
|---|---|---|---|---|---|---|
| Stable | Release | none | v0.37.5 | Yes | No | Stat |
| | | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| Beta | Pre-release | Beta | v0.37.2-Beta | Planned | No | Beta |
| Dev | none | none | v0.0.1 | Active main development branch. This branch will be the base for Beta. | No | No |
| Experimental Feature | Pre-release | Experimental | v0.0.373-Experimental v0.0.373-AwesomeNewFeature | Yes | Yes | Mas |
| Experimental Feature that affects core (and by default, possible stability) | Pre-release | Experimental | v0.0.373-Experimental v0.0.373-AwesomeNewFeature | Yes | Yes | Feat |

The shift for release number to be build only for experimental release would prevent them from being mistaken for a primary release.

### Setting Versions.prop

`Version.props` will only be set by the build farm.

**This work is done.**

### Local / Development build versions

The application version will be **v0.0.1** for local development. PowerToys will not actively check for a new version if it is `v0.0.1` on start to not cause unneeded notifications.

**This work is done.**

## Building signed versions

We build the main development branch every day at noon and midnight. This will allow us to know if we have a build breaking issue. These since these will continue to be built against `v0.0.1`, even when we test a build, it won't impact our ability to see crashes for a public release. We can test against these internally without an issue and won't cause any issue for what build is what. When have a confirmed commit we want a signed build from, we will bring have the `Stable` branch based against this and set the variable in the pipeline to have a pre-build event actually set the version number in `Versions.prop`. This will make the only item able to set the file be on the farm.

We will increase the *build* number for any hot fix that goes into Stable, including if that is before a public release. This directly implies we could release a `v0.15.2` without the public ever seeing a `v0.15.0` or `v0.15.1` due to finding a last minute critical bug. We did this with the 0.29 release of PowerToys.

We will only release signed builds to the channels.

## Open questions

- It would be nice to somehow validate the commit or timestamp the version number in. We could repurpose *Revision* as a timestamp. It would be [YY][DayOfTheYear][# build of the day] as a possibility since we can't leverage the commit hash. This would require additional work in the C++ project files and where they interact with version strings.
- Can we automate the auto-incremental the Build number.