Dev Documentation

Fork, Clone, Branch and Create your PR

Once you've discussed your proposed feature/fix/etc. with a team member, and you've agreed an approach or a spec has been written and approved, it's time to start development:

1. Fork the repo if you haven't already
2. Clone your fork locally
3. Create & push a feature branch
4. Create a Draft Pull Request (PR)
5. Work on your changes

Rules

- **Follow the pattern of what you already see in the code.**
- Coding style.
- Try to package new ideas/components into libraries that have nicely defined interfaces.
- Package new ideas into classes or refactor existing ideas into a class as you extend.
- When adding new classes/methods/changing existing code: add new unit tests or update the existing tests.

Github Workflow

- Before starting to work on a fix/feature, make sure there is an open issue to track the work.
- Add the `In progress` label to the issue, if not already present also add a `Cost-Small/Medium/Large` estimate and make sure all appropriate labels are set.
- If you are a community contributor, you will not be able to add labels to the issue, in that case just add a comment saying that you started to work on the issue and try to give an estimate for the delivery date.
- If the work item has a medium/large cost, using the markdown task list, list each sub item and update the list with a check mark after completing each sub item.
- When opening a PR, follow the PR template.
- When you'd like the team to take a look, (even if the work is not yet fully-complete), mark the PR as 'Ready For Review' so that the team can review your work and provide comments, suggestions, and request changes. It may take several cycles, but the end result will be solid, testable, conformant code that is safe for us to merge.
- When the PR is approved, let the owner of the PR merge it. For community contributions the reviewer that approved the PR can also merge it.

- Use the `Squash and merge` option to merge a PR, if you don't want to squash it because there are logically different commits, use `Rebase and merge`.
- We don't close issues automatically when referenced in a PR, so after the PR is merged:
  - mark the issue(s), that the PR solved, with the `Resolution-Fix-Committed` label, remove the `In progress` label and if the issue is assigned to a project, move the item to the `Done` status.
  - don't close the issue if it's a bug in the current released version since users tend to not search for closed issues, we will close the resolved issues when a new version is released.
  - if it's not a code fix that effects the end user, the issue can be closed (for example a fix in the build or a code refactoring and so on).

Compiling PowerToys

Prerequisites for Compiling PowerToys

1. Windows 10 April 2018 Update (version 1803) or newer
2. Visual Studio Community/Professional/Enterprise 2022
3. Once you've cloned and started the `PowerToys.sln`, in the solution explorer, if you see a dialog that says `install extra components`, click `install`

Get Submodules to compile

We have submodules that need to be initialized before you can compile most parts of PowerToys. This should be a one time step.

1. Open a terminal
2. Navigate to the folder you cloned PowerToys to.
3. Run `git submodule update --init --recursive`

Compiling Source Code

- Open `PowerToys.sln` in Visual Studio, in the `Solutions Configuration` drop-down menu select `Release` or `Debug`, from the `Build` menu choose `Build Solution`.
- The PowerToys binaries will be in your repo under `x64\Release\`.
- You can run `x64\Release\PowerToys.exe` directly without installing PowerToys, but some modules (i.e. PowerRename, ImageResizer, File Explorer extension etc.) will not be available unless you also build the installer and install PowerToys.

Compile the installer

Our installer is two parts, an EXE and an MSI. The EXE (Bootstrapper) contains the MSI and handles more complex installation logic. - The EXE installs all

prerequisites and installs PowerToys via the MSI. It has additional features such as the installation flags (see below). - The MSI installs the PowerToys binaries.

The installer can only be compiled in `Release` mode, step 1 and 2 must be done before the MSI will be able to be compiled.

1. Compile `PowerToys.sln`. Instructions are listed above.
2. Compile `BugReportTool.sln` tool. Path from root: `tools\BugReportTool\BugReportTool.sln` (details listed below)
3. Compile `WebcamReportTool.sln` tool. Path from root: `tools\WebcamReportTool\WebcamReportTool.sl` (details listed below)
4. Compile `PowerToysSetup.sln` Path from root: `installer\PowerToysSetup.sln` (details listed below)

Prerequisites for building the MSI installer

1. Install the WiX Toolset Visual Studio 2022 Extension.
2. Install the WiX Toolset build tools.

Locally compiling the Bug reporting tool

1. Open `tools\BugReportTool\BugReportTool.sln`
2. In Visual Studio, in the `Solutions Configuration` drop-down menu select `Release`
3. From the `Build` menu, choose `Build Solution`.

Locally compiling the Webcam reporting tool

1. Open `tools\WebcamReportTool\WebcamReportTool.sln`
2. In Visual Studio, in the `Solutions Configuration` drop-down menu select `Release`
3. From the `Build` menu, choose `Build Solution`.

Locally compiling the installer

1. Open `installer\PowerToysSetup.sln`
2. In Visual Studio, in the `Solutions Configuration` drop-down menu select `Release`
3. From the `Build` menu choose `Build Solution`.

The resulting `PowerToysSetup.msi` installer will be available in the `installer\PowerToysSetup\x64\Release\` folder.

Supported arguments for the .EXE Bootstrapper installer Head over to the wiki to see the [full list of supported installer arguments][installerArgWiki].

Debugging

The following configuration issue only applies if the user is a member of the Administrators group.

Some PowerToys modules require being run with the highest permission level if the current user is a member of the Administrators group. The highest permission level is required to be able to perform some actions when an elevated application (e.g. Task Manager) is in the foreground or is the target of an action. Without elevated privileges some PowerToys modules will still work but with some limitations:

- The `FancyZones` module will be not be able to move an elevated window to a zone.
- The `Shortcut Guide` module will not appear if the foreground window belongs to an elevated application.

To run and debug PowerToys from Visual Studio when the user is a member of the Administrators group, Visual Studio has to be started with elevated privileges. If you want to avoid running Visual Studio with elevated privileges and don't mind the limitations described above, you can do the following: open the `runner` project properties and navigate to the `Linker -> Manifest File` settings, edit the `UAC Execution Level` property and change it from `highestAvailable (level='highestAvailable')` to `asInvoker (/level='asInvoker')`, save the changes.

How to create new PowerToys

See the instructions on how to install the PowerToys Module project template. Specifications for the PowerToys settings API.

Implementation details

`Runner`

The PowerToys Runner contains the project for the PowerToys.exe executable. It's responsible for:

- Loading the individual PowerToys modules.
- Passing registered events to the PowerToys.
- Showing a system tray icon to manage the PowerToys.
- Bridging between the PowerToys modules and the Settings editor.

Image of the tray icon

`Interface`

Definition of the interface used by the `runner` to manage the PowerToys. All PowerToys must implement this interface.

`Common`

The common lib, as the name suggests, contains code shared by multiple PowerToys components and modules, e.g. json parsing and IPC primitives.

`Settings`

Settings v2 is our current settings implementation. Please head over to the dev docs that goes into the current settings system.