# Contributing to the Windows Community Toolkit

The foundation of the \*\*Windows Community Toolkit\*\* is simplicity.

A developer should be able to quickly and easily learn to use the API.

Simplicity and a low barrier to entry are must-have features of every API. If you have any second thoughts about the complexity of a design, it is almost always much better to cut the feature from the current release and spend more time to get the design right for the next release.

You can always add to an API, you cannot ever remove anything from one. If the design does not feel right, and you ship it anyway, you are likely to regret having done so.

That's why many of the guidelines of this document are obvious and serve only one purpose: Simplicity.

- [Questions](#question)

- [Issues or Bugs](#issue)

- [Submitting a pull request](#pr)

- [Quality assurance for pull requests for XAML controls](#xaml)

- [General rules](#rules)

- [Naming conventions](#naming)

- [Documentation](#documentation)

- [Files and folders](#files)

## <a name="question"></a> Questions

Please do not open issues for general support questions and keep our GitHub issues for bug reports and feature requests. There is a much better chance of getting your question answered on [StackOverflow](https://stackoverflow.com/questions/tagged/windows-community-toolkit) where questions should be tagged with the tag `windows-community-toolkit`

## <a name="issue"></a> Found a Bug?

If you find a bug, you can help us by

[submitting an issue](https://github.com/windows-toolkit/WindowsCommunityToolkit/issues). Even better, you can

[submit a Pull Request](#pr) with a fix.

## <a name="pr"></a> Submitting a pull request

For every contribution, you must:

\* test your code with the [supported SDKs](readme.md#supported)

\* follow the [quality guidance](#xaml), [general rules](#rules) and [naming convention](#naming)

\* target master branch (or an appropriate release branch if appropriate for a bug fix)

\* If adding a new feature

\* Before starting coding, \*\*you should open an [issue](https://github.com/windows-toolkit/WindowsCommunityToolkit/issues/new?assignees=&labels=feature+request+%3Amailbox\_with\_mail%3A&template=feature\_request.md&title=%5BFeature%5D)\*\* and start discussing with the community to see if your idea/feature is interesting enough.

\* Add or update a sample for the [Sample app](https://github.com/windows-toolkit/WindowsCommunityToolkit/tree/master/Microsoft.Toolkit.Uwp.SampleApp)

\* If creating a new sample, create a new icon by following the [Thumbnail Style Guide and templates](https://github.com/Microsoft/UWPCommunityToolkit-design-assets)

\* Add or update unit tests (if applicable)

PR has to be validated by at least two core members before being merged.

Once merged, you can get a pre-release package of the toolkit by adding this ([Nuget repo](https://dotnet.myget.org/F/uwpcommunitytoolkit/api/v3/index.json) | [Gallery](https://dotnet.myget.org/gallery/uwpcommunitytoolkit)) to your Visual Studio.

### <a name="docs"></a> Adding Documentation

Documentation is \*\*required\*\* when adding, removing, or updating a control or an API. To update the documentation, you must submit a seperate Pull Request in the [documentation repository](https://github.com/MicrosoftDocs/WindowsCommunityToolkitDocs) (use the \*master\* branch). \*\*Both Pull Requests (code and docs) must be approved by the core team before either one is merged.\*\*

Make sure to update both Pull Requests with a link to each other.

If adding a new documentation page:

\* Copy the [documentation template](https://github.com/MicrosoftDocs/WindowsCommunityToolkitDocs/blob/master/docs/.template.md) and follow the same format.

\* Update the [Table of Contents](https://github.com/MicrosoftDocs/WindowsCommunityToolkitDocs/blob/master/docs/toc.md) to point to the new page

## <a name="xaml"></a> Quality assurance for pull requests for XAML controls

We encourage developers to follow the following guidances when submitting pull requests for controls:

\* Your control must be usable and efficient with keyboard only

\* Tab order must be logical

\* Focused controls must be visible

\* Action must be triggered when hitting Enter key

\* Do not use custom colors but instead rely on theme colors so high contrasts themes can be used with your control

\* Add AutomationProperties.Name on all controls to define what the controls purpose (Name is minimum, but there are some other things too that can really help the screen reader).

\* Don't use the same Name on two different elements unless they have different control types

\* Use Narrator Dev mode (Launch Narrator [WinKey+Enter], then CTRL+F12) to test the screen reader experience. Is the information sufficient, meaningful and helps the user navigate and understand your control

\* Ensure that you have run your xaml file changes through Xaml Styler (version 2.3+), which can be downloaded from [here](https://visualstudiogallery.msdn.microsoft.com/3de2a3c6-def5-42c4-924d-cc13a29ff5b7). Do not worry about the settings for this as they are set at the project level (settings.xamlstyler).

You can find more information about these topics [here](https://blogs.msdn.microsoft.com/winuiautomation/2015/07/14/building-accessible-windows-universal-apps-introduction)

This is to help as part of our effort to build an accessible toolkit (starting with 1.2)

## <a name="rules"></a> General rules

\* DO NOT require that users perform any extensive initialization before they can start programming basic scenarios.

\* DO provide good defaults for all values associated with parameters, options, etc.

\* DO ensure that APIs are intuitive and can be successfully used in basic scenarios without referring to the reference documentation.

\* DO communicate incorrect usage of APIs as soon as possible.

\* DO design an API by writing code samples for the main scenarios. Only then, you define the object model that supports those code samples.

\* DO NOT use regions. DO use partial classes instead.

\* DO declare static dependency properties at the top of their file.

\* DO NOT seal controls.

\* DO use extension methods over static methods where possible.

\* DO NOT return true or false to give sucess status. Throw exceptions if there was a failure.

\* DO use verbs like GET.

\* DO NOT use verbs that are not already used like fetch.

## <a name="accessibility"></a> Accessibility Guideline

We'll follow this guideline to ensure the basic accessibility features for each control.

### Color & High Contrast themes

\* Controls must support the 4 high contrast themes by default on Windows, in addition to changing the theme while the app is running.

\* Controls must have a contrast ratio of at least 4.5:1 between the text (and images with text) and the background behind it.

### Keyboard

\* Controls must support keyboard navigation (tabs and arrow keys), the tab order must be the same as the UI and non-interactive elements mustn't be focusable.

\* Composite elements must ensure proper inner navigation among the contained elements

\* Clickable UI elements must be invokable with the keyboard (The trigger keys are enter and space).

\* Focusable elements must have a visual focus indicator. It's usually a rectangle shape around the control's normal bounding rectangle.

### Narrator

\* Controls must support narrator.

## <a name="naming"></a> Naming conventions

\* We are following the coding guidelines of [.NET Core Foundational libraries](https://github.com/dotnet/corefx/blob/master/Documentation/coding-guidelines/coding-style.md).

## <a name="documentation"></a> Documentation

\* DO NOT expect that your API is so well designed that it needs no documentation. No API is that intuitive.

\* DO provide great documentation with all APIs.

\* DO use readable and self-documenting identifier names.

\* DO use consistent naming and terminology.

\* DO provide strongly typed APIs.

\* DO use verbose identifier names.

## <a name="files"></a> Files and folders

\* DO associate no more than one class per file.

\* DO use folders to group classes based on features.