# Contributing to Windows Template Studio

The foundation of \*\*Windows Template Studio\*\* is get a developer's File->New Project experience up and going as fast possible.

A developer should be able to quickly and easily add features, pages, and have a solid foundation to start with. The starting code and XAML will be best practices, follow design guidelines and be commented to help aid in enabling everything a developer to get started and understand \*\*why\*\* something is like it is.

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

Also remember that the Pull Requests must be done against the \*\*[dev branch](https://github.com/Microsoft/WindowsTemplateStudio/tree/dev)\*\*.

## Before you begin

While we're grateful for any and all contributions, we don't want you to waste anyone's time. Please consider the following points before you start working on any contribution.

\* Please comment on an issue to let us know you're interested in working on something before you start the work. Not only does this avoid multiple people unexpectedly working on the same thing at the same time but it enables us to make sure everyone is clear on what should be done to implement any new functionality. It's less work for everyone, in the long run, to establish this up front.

\* The code that is output in the generated projects may end up in thousands of apps so it must be of the highest quality. Expect it to be reviewed very thoroughly and it must meet our standards for standards for style, structure, and format. There are details below and automated tests to verify their use.

\* Get familiar with the automated tests that are part of the project. With so many possible combinations of output, it's impossible to verify everything manually. You will need to make sure they all pass.

\* When adding anything new it should be created to work with all supported frameworks. If this is going to be a problem, discuss it before beginning work.

\* We support templates for apps built with both C# and VB.Net but appreciate that not evevryone wants to work in both languages. For this reason we have a C#-first approach. This approach means that new functionality if first created in the C# templates and the VB.Net version is created after. If contributing something it is ok to submit a PR that just contains the C# version. For all non-code files (xaml, images, etc.) the different language versions should use identical copies of files. The VB templates contain a copy of such files so that it is possible to change the templates for each language separately.

## A good pull request

Every contribution has to come with:

\* Before starting coding, \*\*you must open an issue\*\* and start discussing with the community to see if the idea/feature is interesting enough.

\* A documentation page in the [documentation folder](https://github.com/Microsoft/WindowsTemplateStudio/tree/release/docs).

\* Unit tests (If applicable, or an explanation why they're not)

\* If you've changed the UI:

- Be sure you are including screenshots to show the changes.

- Be sure you have reviewed the [accesibility checklist](docs/accessibility.md).

\* If you've included a new template:

- Be sure you reviewed the [Template Verification Checklist](https://github.com/Microsoft/WindowsTemplateStudio/wiki/Template-Verification-Checklist).

\* You tested your code with two most recent Windows 10 SDKs. (Build 18362 and 19041)

\* You've run all existing tests to make sure you've not broken anything.

\* PR has to target dev branch.

PR has to be validated by at least two core members before being merged. Once merged, it will be in the pre-release package. To find out more, head to [Installing / Using the extension](docs/getting-started-extension.md).

## Quality insurance for pull requests for XAML

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

\* Your XAML 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.

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 building accessible templates from the start.

## General rules

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

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

\* DO NOT seal controls.

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

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

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

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

\* DO use verbs like GET.

## Naming conventions

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

## Documentation

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

\* DO provide great documentation with all new features and code.

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

\* DO use consistent naming and terminology.

\* DO provide strongly typed APIs.

\* DO use verbose identifier names.

## Files and folders

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

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