Technical Policy

[christian@schladetsch.com](mailto:christian@schladetsch.com)

This document is issued for free commercial and non-commercial use under the [MIT](https://opensource.org/licenses/MIT) License.

If you make generally good changes, it is hoped that you will push them back.

## Abstract

This document prescribes a set of policies to follow when going about daily technical work within a multiple-disciplinary technical team.

It also serves a guide for third-party users of such systems.  
  
Lastly, it has a free *MIT* license so it can be used as a basis for other companies that do not have any such documentation.

**Glossary**

|  |  |
| --- | --- |
| *May*: | May be followed and should not be flagged. |
| *Must*: | Must be followed and should be flagged. |
| *Required:* | Required to be followed and required to be flagged. |
| *Should:* | Should be followed and may be flagged. |

If *May/Must/Required/Should* is omitted from a policy term, then it is considered as *Must*.

Terms starting with “Do” or “Do not” are considered as *Must*.

*Must* becomes *Should,* and *Required* becomes *May* in relation to third-party sources.

## 

## Version History

|  |  |
| --- | --- |
| 0.2 | Peer reviewed & published. |
| 0.3 | Iterated by team. |
| 0.4 | Add numbered references to clauses. |
| 0.5 | Simplified Abstract and Introduction text, new recursive Glossary, added and refined terms. |
| 0.6 | Made publicly editable. |
| 0.7 | Made public. |

## 

## Introduction

This is the canonical standards document for how things are created, stored and referenced within a workforce.

Standards such as these help reduce communication error, inform developers on how to make decisions regarding technicalities like naming conventions and project structure, and ease transition across projects.

You may use this document as a reference when asking for a change to be made by another developer, especially when reviewing code via a *Pull Request*.

Everyone in the company - or anywhere else - is free to edit this document. Just be ready to defend any such change.

These rules apply only for internal development purposes.

## 

## 1. General

There are some common rules that apply in all cases:

1. Bullet points should start with a capital and end with punctuation.
2. Must add new clauses or sub-clauses at the end of any list in this document.

Rationale: Existing references to clauses must not break due to reformatting of this source text.

1. Should use correct English: e.g. Realise rather than Realize, etc. Consider your language usage.
2. All general documents must be stored on [*GoogleDocs*](http://docs.google.com).
3. Should not use spaces or any punctuation in any file-name or object-name.
   1. Such names should consist of alpha-numeric characters only, starting with a non-numeric character, and use UpperCamelCase.
4. Technical documentation should be concise and correct.
   1. This means updating older documentation when things change.
5. Internal documents stored in any git repository must use [*MarkDown*](https://en.wikipedia.org/wiki/Markdown).
6. Should provide links whenever you can. Read about how to add links to GoogleDocs and MarkDown.
7. Should use *Italics* when referring to companies or when initially referring to some external library or technology like [*Microsoft .Net*](https://dotnet.microsoft.com/).
   1. Should also use this as a link.
8. Do not repeat yourself.
   1. Use links to refer to other [documentation](https://docs.google.com/document/d/1w4MYPKsW-Prj6z9XlJjeuN-h4Fbtoas38aWgHTlMEjg/edit#).
9. *GoogleDocs* should use the default font: Arial 11-pt.
10. Should use fixed-width fonts, preferably Courier New 11pt, when referring to
    1. Files.
    2. [*Unity3d*](http://www.unity3d.com) Object names.
    3. Script names or script contents, or
    4. git branch names.
11. Should use UseUpperCamelCase.ext for all file names.
12. Should not use any underscores, dashes, spaces, or fullstops in any names.
13. Acronyms such as ABC should be spelled Abc.
    1. Note that SI units are not acronyms. So, to indicate mega-bytes, it is correct to say “MB” and incorrect to say “Mb” - which is a different thing.
14. Do not use emoticons or curse in any formal writing, including code or scripts, or when communicating with clients.
15. Should use indicative and consistent avatar images across all platforms, like *Trello*, *Slack*, *GitHub*, *Gmail* etc.
16. Text files should be UTF-8, UTF-16, or ASCII encoded, in that order of preference.
    1. Should not use UNICODE for anything.
17. Line endings on all files created or used by [*Windows*](https://www.microsoft.com/en-au/windows)must use \r\n line-endings.
18. Line endings on all [*Linux*](https://www.linux.org/)- or [*macOS*](https://www.apple.com/au/macos/)-based text files must end with \n.
19. Should use *WorkFolder* system.
20. Must use [*Iso-8601*](https://en.wikipedia.org/wiki/ISO_8601) for dates and times.
    1. Store all date-time values as [*Utc*](https://en.wikipedia.org/wiki/Coordinated_Universal_Time). Do not use local time, which may include daylight savings. At Melbourne, Australia we are typically Utc+11.
    2. YY-MM-DD for dates.
    3. YY-MM-DDTHH-mm-ss for date and time for filenames.
21. Internal copyright notices must be of the form *© [year] [Company] Pty Ltd.*
22. Do not include any whitespace at the ends of any text lines.
23. Should use [*IEC*](https://en.wikipedia.org/wiki/International_Electrotechnical_Commission) binary prefixes for base-2 quantities and [*SI*](https://en.wikipedia.org/wiki/International_System_of_Units) prefixes for base-10 quantities.

Rationale: Disambiguation. See [Mebibyte](https://en.wikipedia.org/wiki/Mebibyte). Basically, if you mean 1000 bytes, say a kilobyte (KB). If you mean 210 or 1024 bytes, say [kibibyte](https://www.google.com/search?q=kibibyte) (KiB).

## 

## 2. Unity3d

The current state of *Unity3d*, especially as it relates to Vr, is chaotic. This is not expected to change as new hardware and software is rapidly introduced.

Even so, many entire companies are based on Unity3d. We have to manage this complexity collectively.

To that end:

1. Must consult your lead before making a new project.
2. Do not use random assets from e.g. [*Asset Store*](https://assetstore.unity.com/) without first contacting your lead.
3. Third-party files must live wherever causes the least harm.
4. Do not include *Examples* or *Tests* from any third-party library or package in any first-party repo.
   1. Do include a Readme.md that points to original asset including any samples etc.
5. Should use /w/bin/new-project to make a new project.
   1. This will become a *must* from September 2019.
6. Application source files (specific to current application) must live in Assets/App.
7. Should include a Readme.md in every major folder that briefly describes the purpose and contents of that folder.
8. Internal libraries are supplied as Unity3d [*Packages*](https://docs.unity3d.com/Manual/Packages.html).
   1. Use master branch for all local packages.
9. Do not call objects in the scene or prefabs GameObject.
10. Should Learn *bash*, *vi* and *git* from the command-line.

## 

## 3. Git

Using [git](https://git-scm.com/) is considered a part of professional competency**.** You should feel comfortable to use it without breaking things - just as much as using *Maya* or *PhotoShop* or *Visual Studio* or *Unity3d*.

Thankfully, there’s a worm-hole within git itself. What makes it so hard to use means you can’t really irrevocably break anything or lose work unless you go out of your way and do things you do not understand.

At least read about [git stash](https://git-scm.com/book/en/v1/Git-Tools-Stashing) if you’re unsure about things. Yes, this is extra thought and knowledge required from you.

Git is assuredly complicated. There are ways to make it easier to use:

1. master branch must always build and run.
   1. Do not touch master branch.
2. develop branch should always build and run.
3. Do not do things you do not understand.
   1. Be very careful with rebase.
4. Should use *WorkFolder* system.
   1. Will become a *must* soon.
   2. Makes the next two points redundant.
5. Should use [git-flow](https://www.google.com/search?q=git+flow) (nothing to install, but a step to prepare).
   1. git flow init
6. Must use [git-lfs](https://git-lfs.github.com/) (nothing to install, but a step to prepare).
   1. git lfs install
7. Should branch from develop.
   1. Unless in a small isolated team.
   2. Rather, use branches via *git-flow*.
   3. You can branch from a specific branch using git-flow.
      1. git-flow **-b source-branch** …
      2. The default source branch is develop.
8. Do not make any changes to any git work-flow without discussing with Cto first.
9. Must not touch master branch.
   1. The only thing to do on master is either merge from develop if you’re team lead, or
   2. Make a hotfix that goes through a *Pull Request*.
   3. As a corollary, the git log for master must only have merge, tag and hotfix commits.
   4. This excludes Unity3d Packages which anyone can pull master.
   5. Package development is only done by leads on develop or *git-flow* branches from develop.
10. Must go through a peer-reviewed *Pull Request* when merging to develop.
    1. Except for changes to scene or prefabs, then it’s a should.
11. Do what you want on branches other than develop or master.
    1. You can share branches! Just pull someone else’s branch.
    2. You do not have to ‘share’ only via develop or master.
12. Must use lower-dashed-case for git branch names.

### 3.1 Make a Pull Request

*Pull Requests* (PRs) are an important part of the development process.

1. Always merge develop before you make a PR.
2. To make a Pull Request, say
   1. **λ mpr**
3. Or browse to:
   1. https://github.com/**[company]**/**[repo-name]**/pull/new/develop
4. How to do this via a Gui is left up to you.
5. Do not make PRs to master. Team leads will deal with with master.

### 3.2 Review a Pull Request

1. As a reviewer, you should first pull the branch and ensure it builds and runs locally.

## 

## 4. Maya/Modelling

**HELP REQUESTED**

1. Must use 1 unit = 1 meter.
2. Must export using [*Left-handed Coordinates*](https://www.evl.uic.edu/ralph/508S98/coordinates.html)
   1. Right is +x, Up is +y, Forward is +z.
3. ~~Should~~ Must use [.fbx](https://en.wikipedia.org/wiki/FBX) for source.
4. Items ~~should~~ must import and work without needing to be scaled/rotated/offset Unity3d-side.

## 

## 5. Photoshop/Images

**HELP REQUESTED**

1. Should use [.png](https://en.wikipedia.org/wiki/Portable_Network_Graphics) for source images.
2. Must use [*ASTC* compression](https://en.wikipedia.org/wiki/Adaptive_Scalable_Texture_Compression) for import.
   1. If in doubt, default to 4x4 compression.
   2. Note that 4x4 is smaller (more compressed/noisy) than 8x8 etc.
   3. Do not confuse with [ATSC](https://en.wikipedia.org/wiki/ATSC_tuner) (as I often do).
3. Should not include .psd files as assets in Unity3d.
   1. Break out components into separate well-named assets.

## 

## 6. Audio

**HELP REQUESTED**

1. Should use lossless format for source audio, such as flac or very high 320+bps compressed.
2. Must use [.ogg](https://xiph.org/vorbis/) for import.

## 

## 7. C#

See reference code.

We support *Visual Studio 2019 Community Edition,* [*NDepend*](https://www.ndepend.com/) and [*ReSharper Pro*](https://www.jetbrains.com/resharper/).

For example of style requirements, see other canonical repositories like [Flow](https://github.com/cschladetsch/Flow). Note that these are a living breathing libraries, so there will be some inconsistencies.

Feel free to correct any problems and submit a *Pull Request*.

1. All coders must use [*ReSharper Pro*](https://www.jetbrains.com/resharper/).
2. Must use spaces rather than tab characters.
   1. Tab must be 4 spaces.

Rationale: Consistent readability across all platforms and editors, both local and remote, such as GitHub Pull Requests.

* 1. You will be fired if you mix tabs and spaces in the same file.

1. If a given code-line has over 90 characters, it should be wrapped.
   1. If it has over 110 it must be wrapped.

Rationale: Readability within the context of a vertically-split tab in *Visual Studio*.

1. Do read other internal code bases.

Rationale: Learn the specs.

1. Do follow syntax and formatting conventions used in any file you modify, even if doing so contravenes conventions stated in this document.
   1. If it's an internal file, follow convention of the local file and make a separate PR for the formatting change.

Rationale: Commits should be atomic. Separate logical changes/additions from formatting changes.

1. Do use your intuition after reading common libraries such as [*Flow*](https://github.com/cschladetsch/Flow) etc.

Rationale: I am not going to write pages and pages of explicit instructions about how to write and format every aspect of C# code.

The standards for C#, and C# itself, changes over time, so that is a pointless endeavor in any case.

Read existing code and always follow conventions around any changes you make, even if they contradict what is in this document.

1. Do not make arbitrary formatting changes which pollute the git commit history.
   1. Put these in a new commit called “Reformatting”.
   2. Such commits must not change any logic.
2. Should provide complexity guarantees using [*big-O notation*](https://en.wikipedia.org/wiki/Big_O_notation) for all properties or methods that are prefixed with *Calc*.

Rationale: Allows a developer to know what he’s in for.

1. Namespaces must use UpperCamelCase.

Rationale: Consistency in naming.

1. Assemblies that may be consumed by other projects must use a unique top-level namespace.

Rationale: If an assembly is not consumed by another project, it doesn’t matter what namespace it uses. Conversely, it does matter.

1. All symbols should be in a namespace.
2. All public classes, interfaces, delegates, and enumerations in an assembly must have [*Xml comments*](https://docs.microsoft.com/en-us/visualstudio/ide/reference/generate-xml-documentation-comments?view=vs-2019).

Rationale: Provide more information to the reader.

1. All other public symbols should also have Xml Comments.

Rationale: Provide more information to the reader. Also will help any automation tools used to produce documentation.

1. **Enumerations** must start with the letter ‘E’.

Rationale: Makes it clear to the reader that the given symbol name is an Enumeration type.

1. **Interfaces** must start with the letter ‘I’ (capital-i).

Rationale: Makes it clear to the reader that the given symbol name is an Interface type.

1. **Public** methods must use UpperCamelCase in verb-noun form.
   1. GetFoo implies either a fast lookup or an immediate return.

Consider a property Foo instead.

* 1. CalcFoo implies that work has to be done to get a result.

Do not use a property that does much work every time it is referenced.

* 1. FetchFoo implies that work *may* be done to get a result.

1. **Protected** fields must use \_UnderUpperCamelCase.
2. **Private** fields must use \_underLowerCamelCase.
3. **Arguments and local variables** must use lowerCamelCase.
4. **Constant/Static** fields must use the same conventions as above.
5. All expression bodies must go on the next line.
   1. Longer expressions must use a code-block.

Rationale:

* + 1. Consistency.
    2. Stops situations like: one method has an expression body on same line, the next one has an expression body on the next line, the next has a code block.

1. ~~Should~~ May use braces {} on single-line code-blocks.
2. Must add a space after keywords.
3. No spaces around multiplicative operators like \* and /.
4. Space around additive operators like + and -.
5. Group expressions in parentheses if evaluation order is not clear.
6. Must use [Allman-style](http://syque.com/cstyle/ch6.7.htm) braces.
7. Do not curse or use disparaging remarks or slang or emotes in code or comments.

// wtf is this shit :( ← um, no.

1. Should not commit commented-out code.
2. Sentences in log messages and comments should end with a full-stop.

Rationale: This makes it clear when the message has correctly ended and was not abruptly terminated.

1. Do not use a property that does much work every time it is referenced.
2. All usings must be within a namespace.

Rationale: This avoids any ambiguity early.

1. Add a newline after a code block or `var` statement sequence.

Rationale: Consistency.

1. Must not have empty newlines in empty blocks.
2. Should not have newlines in using sequences.

Rationale: Save vertical whitespace.

## Finis