EDLM: proposal

132nd Virtual Wing
Documentation overhaul

APPLIES TO:	**132nd Virtual Wing**
TYPE:	**Enhancement Proposal**
VERSION:	0
PUBLISHED DATE:	unpublished
DOCUMENT RESPONSIBLE:	etcher
SUMMARY OF CHANGES:	nil

Contents

Figure 1: Documentation

1 Enhancement Proposal

1.1 Abstract

This document contains an enhancement proposal submitted to CMD for review.

The proposal is about documentation in the 132nd Virtual Wing. I would like to explore possible ways to make it better, easier to produce and easier to maintain.

1.2 Rationale

I was working a while back on the 696th TURNSKINS TRP (which produced no useful end-result), and was taken aback by many of Microsoft Word pitfalls. Why does it decide to auto-format so many things on my behalf ? And why can't it keep it consistent ? Reznik and I were editing with different locale settings (fr_BE and da_DK), which added even more fun to the mix: different paper size, different quote characters, and so on ...

I've also been offered the chance to review the new 176th EYES OF THE NORTH SOP before it officially came out, and the whole review process, using a proxy Google Docs temporary document seemed archaic and painful to me. This extra step adds a lot of undesirable overhead to the process.

Last example: as a side-project, I'm maintaining a knee board document written with Microsoft Excel, and that takes the pain to a whole new level.

I would like to propose a solution to get rid of those issues, and implement a better work flow for everyone. This document is published for review in order to assess how interested/willing people are interested to give it

a try.

1.3 Objective

What I'm trying to achieve:

- Increase quality and consistency of the documentation
- Allow for easier collaboration and review
- Maintain an easy work flow for everyone

What I'm trying to get rid of:

- Variations in style or formatting
- Older documents not updated to the latest layout/formatting
- Passing documents around during review/editing process

1.4 Getting there

This section conceptually describes how to achieve the goals outlined above.

1.4.1 Decoupling the content from the format

Writing a document with Microsoft Office Word implies formatting it as well. Despite the use of a template, guaranteeing a consistent format across a library of dozens of documents, written by many authors, sometimes even multiple authors for the same document, is almost impossible.

Due to the inherent complexity of Microsoft Word Office, as well as the debatable pertinence of some choices it sometimes seems to automatically make for us, some subtle format changes are bound to sneak unnoticed within the document over time.

My first goal is to get rid of this ambiguity, and have one and only one way to express a given construct (a paragraph, a heading, a list, a table, . . .).

This would also have the beneficial side effect of freeing editors from worrying about formatting their content while they are writing it, giving them more brain power for the actual content creation

Decoupling content and format also means that any older document, even one that hasn't been touched in years, will be automatically updated to the latest format/layout whenever the format/layout is updated, since their content has not changed.

1.4.1.1 Specification

Once the content has been created by the editors, my goal is to provide a system that will take that "raw" content, and format it, consistently, into different formats that will later be published. A choice format is of course PDF, but we can also convert to Microsoft Word format, HTML (create a website automatically for our documentation), EPUB (books that can be read easily on readers/mobile), you-name-it.

Here is a non-exhaustive list of the *output* format supported: (more *input* formats are available still)

> Pandoc can write plain text, Markdown, CommonMark, PHP Markdown Extra, GitHub-Flavored Markdown, MultiMarkdown, reStructuredText, XHTML, HTML5, LaTeX (including beamer slide shows), ConTeXt, RTF, OPML, DocBook, OpenDocument, ODT, Word docx, GNU Texinfo, MediaWiki markup, DokuWiki markup, ZimWiki markup, Haddock markup, EPUB (v2 or v3), FictionBook2, Textile, groff man pages, Emacs Org mode, AsciiDoc, InDesign ICML, TEI Simple, and Slidy, Slideous, DZSlides, reveal.js or S5 HTML slide shows. It can also produce PDF output on systems where LaTeX, ConTeXt, or wkhtmltopdf is installed.

The output should be:

- Consistent across builds: the same content must **always** yield the same result, even on different computers, operating systems, or software versions
- Uniformly formatted: **all** the documents in the library should have the same general layout, giving all documentation published by the 132nd Virtual Wing a visual identity of their own
- Retroactively managed: all documents that have been published in the past should be **updated without human intervention**. If a logo changes, if we decide to change the title page, or the space after paragraphs, those changes should be **automatically propagated across the entire library**
- Adapted to our needs: the documentation should not look "generic" or bland; each document should bear ##

1.5 Pros and cons

This section objectively (as much as I could) describes the pros and cons of the method I propose to implement.

1.5.1 The cons

Allow me to start with the cons, and provide, for each of them, a way to mitigate them.

1.5.1.1 No WYSIWYG ("What you see is what you get")

Markdown is pure text, therefore an editor who is busy writing documentation does not see the result appear as he types. Font does not resize for headings, pictures do not appear, tables do not build, etc.

1.5.1.1.1 *Mitigation*

- Converting from Markdown to Word/PDF/HTML is trivial, and the tool that I will be providing can be installed on any modern computer. Output can therefore be refreshed as often as needed to get a "sneak peek" into the actual result.
- Numerous tools exists *online* to edit Markdown. Some of those tools are bare editor, providing a side by side "Edit" window and a "Result" window. Their offline equivalent are available as well. Moreover, and this gets really interesting, Markdown is mature enough that many *free* online editors offer amazing synchronization capabilities with Google Drive, Dropbox, Github, ..., **effectively allowing anyone to work on any document from any computer connected to the web, and directly sends their work for review into the Github repository of the 132nd Virtual Wing**.

1.5.1.2 Less liberty when it comes to customizing the format/layout

Having a common template for the layout/format of our document effectively "castrates" editors, denying them the liberty to get creative with the way their content is rendered. This could get on the nerves on some, especially the most perfectionists among us.

1.5.1.2.1 *Mitigation*

- The rendering, formatting and layout is done by a professional (although free) typesetting application that has been in existence since 1985: Latex. 32 years of development, testing and improvements have made it quite robust. It has been in use for decades by the scientific and teaching community all around the world for papers, essays, reports, etc. Even if we might disagree with some of the minor formatting choices it makes when it comes to typesetting the document (I sometimes do myself, with the placement of pictures for example), we can at least be sure that the standard it follows is accepted world-wide, and is the result of decades of professional work.

- The layout/format will be 100% identical for all documentation published by the 132nd Virtual Wing, branding our documents with a unique "personality", and giving an overall "neat" picture of the Wing to the external world.
- In case it becomes necessary, when part of the output does not fit a specific need of ours, we can take advantage of the maturity of the tools and customize every little detail to our needs (this would of course be my job, unless someone else feels like diving in the gory implementation details =)).

1.5.1.3 Resources like pictures, videos, sound, are to be included on the side

All the files that are to appear in the final document will be referenced in Markdown as links only, pointing to a file that exists near the Markdown source document (I'll come back on the structure later). To give an example, if I wanted to include a file named `picture.png` in a document named `index.md`, I would have to write the following in `index.md`: `[Picture caption](picture.png){width: 8cm}`. "Picture caption" simply describes the picture, and can be any text. The `path` part is irrelevant, since that will be handled by the tools I am writing, but the problem persists: **media only appear as text and are to exist alongside the source** (the `picture.png` must exist in a `media` folder next to `index.md`).

Note: the `{width: 8cm}` part is optional, but is amazing to "force" the same width (or height) on all pictures throughout a document (which looks really nice).

1.5.1.3.1 *Mitigation*

- Declaring pictures by name offers a finer grained controls on their size, and allow for dynamic resizing of pictures from all origins: one could, instead of providing a picture file in the `media` folder and give its name, include an arbitrary URL, for example: `[My cool picture](https://www.all-about-atc.com/the_pattern.png){width:8cm}`. The system is very flexible.
- Updating pictures can be done without even opening the source. A file named `picture.png` will be included in the final document, whatever that file contains. Updating batch of pictures is thus very easy, and does not require updating them one by one in every Word document (imagine how easy it would be to be able to run batch updates on the pictures' library ...).
- Pictures are automatically indexed and referenced in the final document, and an automatic "Table of figures" is automatically generated, with hyperlinks to the pictures within the text.
- Pictures (and other media) are shared across document (if we want to). There is a `media` folder per document, containing pictures that are relevant to that specific document only. Then there is a `media` folder for the 696th TURNSKINS (for example), which contains all media relevant to the 696th TURNSKINS and is accessible to all documents of the 696th TURNSKINS. And, finally, there is a "root" `media` folder, that contains pictures shared across *all* documents (for example, the squadrons logos). If a logo picture is updated (for any reason, this is just an example), updating the picture file in the root folder would propagate to *all* the documents of the 132nd Virtual Wing, updating that picture in each one of them.

1.5.1.4 New technologies

While all the cons so far are of relatively small import, I fear this one might raise the most shields in our community. Please bear with me for a little while ? For this project, I plan on using two pieces of software for the front-end (the parts people are expected to interact with): **Markdown** and **Git**. For more information about them, see the "Technical" section of this document.

(for information only) The back-end, which editor should never interact with (unless they want to), consists of **Python**, **Pandoc**, **MikTex**, **Latex**, **PyPi**, **Miniconda**, **Anaconda**, **Appveyor**. I will not be talking about those in this document; if you want to know more, send me a ping =)

1.5.1.4.1 *Mitigation*

- While switching to new software always implies somewhat of a **learning curve**, I used the following criteria to select them:
- Free (as in *no dinero*)
- Open source
- Mature
- Widely used across the world
- Well documented
- Will be supported for years to come
- Easy enough for the intended usage
- Has a luxuriant and flourishing ecosystem of tools around them
- Resilient

The initial learning curve should be very much dampened by the *huge* amount of documentation around both *Markdown*, and *Git/Github*. Tutorials and documentation is all over the web, and, more importantly, **mistakes are free**. Even if someone completely nukes all our documentation (and the chances for that to happen are almost none) while toying around trying to learn, it can be restored in seconds.

1.5.2 The pros

This section explains why I'm making this proposal in the first place

1.5.2.1 Intro: the build process

Before I can present you with examples, I need to explain a few basics about the build process I'm proposing.

Every document lives in its own folder, and is composed of three parts:

1. The document itself, `index.md`; this file can contain text, pictures, and include other Markdown files in the same folder (that's useful to split the documents into different parts, and assemble them later, but totally optionla of course).
2. A `settings.yml` file; this file contains information about the document. It is optional too, but can be used to change some settings. For example, you can define aliases, stating that all "//jf" in the document should be replaced by "J-TAC/FAC(A)" (it just saves a lot of typing and help reduce typos).
3. A `media` folder; his folder contains resources for the document, mainly pictures.

In addition to those three parts, there are also the global settings and the global media folder. Those are simply settings and pictures that are shared with *all* the documents.

Example:

- media/
- picture1.png
- picture2.png
- settings.yml (global)
- SOME_DOCUMENT
- media/
- picture3.png
- picture4.png
- settings.yml
- index.md
- part1.md
- part2.md

This architecture allows defining settings and provide pictures at the global level, and overrides them at the document level if desired.

Same thing goes for the media folder: when a picture is included in the document, it will be looked up first in the media folder of the document, and then in the global media folder.

Basic work flow

1.5.2.2 Automation

This is one of the biggest pro in my book. The starting point is a raw text file, editable from anyone on with access to the Internet, and, from there, PDF documents are automatically created and published. Even a full-fledged website if we want to !

The build process is triggered every time a file changes on Github. That means we get to see what the PDF would be like every time someone creates a commit. This is of course very desirable when we are actually publishing the document, but it happens too when we're *working* on it. Every commit on the "develop" branch (the branch used for "work in progress") creates PDF. Every commit on a "pull request" (a "proposal to publish" pending review) creates PDF. In the case of the pull request, the PDF produced are available in the pull request thread, allowing for everyone to review and discuss the change. If the pull request is edited (new changes are added to it), the new files will show as well.

The publication happens every time someone commits on the "master" branch ("master" is the most up-do-date, official version). The documents are published on our FTP, replacing older versions. That means the link to the documents stays the same; only the content gets updated. That also means that anyone with Internet access can, at any point, create a new release of any document (by committing changes, or accepting a pending pull request, etc.) at any time, even from their phone should they wish so.

Note: using the FTP to host documents is my current idea of the implementation, but can be changed very easily. I like the FTP idea because it means that the same document will always have the same link pointing to it, even between versions, which is very convenient for writing briefings, or the documents page of our website.

Automation also gives us a lot of flexibility. Let us say we decide to change the font we use for the documentation, and examine the two following scenarios:

- First scenario, we're using Word. We need to open each document, change the style to use the new font, and hope that all the other styles used in the document depend on the main style. We visually check for that, and hope we won't miss a line. This will take a little bit of time (pun intended) and is very error-prone.
- Second scenario, we're using Markdown. We pull the repo (one click), change one line (10 seconds), then push the repo back (one click). This change is absolute over the entire document library, every single character is guaranteed to have been updated.

This stands not only for the font, but for pretty much everything else too. Another use case: the application I'm writing lets us define "aliases" for recurrent terms. For example, the words "132nd Virtual Wing" can be abbreviated to "132nd Virtual Wing" in the Markdown text file. Those aliases can be defined globally and for each document in a settings files. If we ever decide to become the 131st Virtual Wing, all it takes to update *all* the documents in the library is to change the alias *once* in the root settings file (this is a silly example of course, but you get the gist).

The same goes for pictures: imagine we decide to include the GRG made by Looney (respect, sir) into the 617th DAMBUSTERS TRP. We drop the file "dush_grg.png" into a "media" folder next to our markdown, and simply type `[Dusheti GRG](dush_grg.png){width: 6cm}` in our Markdown text (the part between `[]` is the "caption" of the picture). Note as well that the `{width: 6cm}` is totally optional and included here only because I'm a nerd =) Now every time Looney updates his GRG, all there is to do is drop the new file in place of the old one, and commit the change. No worries about resizing, aligning, formatting, publishing. Pull, change, commit, push, and grab a coffee; 2 minutes top, coffee included, worry free.

Now let's take the example above a tad further. Imagine the 765th THE DREAD PANTHERS decides to include the GRG too. They move the "dush_grg.png" file into the "media" folder of the 132nd Virtual Wing,

making it available for every document in the library, and include it in their TRP too. Now, whenever anyone updates "dush_grg.png" in the root media folder, it automatically gets updated in every document that uses it. How cool is that ? =)

This integrated and automated publication process also gives us a lot of information about everything that is going on, at any given time. For example, I could see who modified a specific line in a specific document. I could also comment directly on *that specific line* and start a discussion about it. I could revert a specific set of changes, if it turns out they're not as good as we thought. Since every commit is independently built, and Git branches are cheap, multiple people can work at the same time on different chapters of the same document, without ever colliding with each other, and have a immediate snapshot of their work every time they push it. Once finished, integrating the change set back into the "develop" or "master" branch is as simple as a click of the mouse.

A side effect of this system is that absolutely every change ever pushed on Github are recorded. Every little one of them can be retrieved, analyzed, and reverted if we want to. Which is, essentially, free backup.

Finally, hosting the documentation on Github gives everyone access to it. Anyone can jump in at any given point, and suggest a change, fix a typo, etc. Owners and maintainers are responsible for accepting or rejecting those changes, but virtually anyone willing to create a Github account can turn into a contributor.

Note: if it turns out that having our work publicly available is a show stopper for some of us, please note that Bitbucket offers the same kind of functionality for an unlimited number of private repositories. I'm a strong advocate of going public, though.

1.5.2.3 Unified format

All markdown documents are transformed into PDF with a master template. This means that:

- all documents will have the same layout and general format, giving them a distinct 132nd Virtual Wing look and feel
- updating the template will reflect the changes on all documents (ex: title page format, heading size, paragraph spacing, bullet lists format, etc.)
- content and format are decoupled
- metadata like table of content, table of figures, table of tables (yes), heading numbers, etc. are automatically generated for all documents

Basically, what happens during the conversion is:

1. Grab the settings from the global `settings.yml` file
2. Update those settings with the document 'settings.yml' file

- All settings that were in the global settings are preserved, unless a setting with the same name is declared for the document, in which case the global settings are overwritten
- All settings present in the document file that are not in the global settings will simply be added

3. Using those settings, pre-process the markdown text files; settings may include:

- aliases to replace some text in the document with predefined string
- a new title for the document, different from the folder name; maybe some characters you want in the title aren't allowed as a folder name on Windows? **Note**: at the time this proposal were written, those were the only two possible settings; many others will probably come in the future

4. Still using the settings, pre-process the Latex template; this will, for example:

- compute the path to the media file

5. Using the resulting template and markdown text, build the final PDF.

Processing

Why such a complicated system ?

First, we want to be able to declare and access settings that are common to all documents, and are susceptible to change in the future (new settings may be added later without requiring changing the document text, no worries). That is the role of the global `settings.yml`.

Then, we also want to be able to create settings that are specific to the document itself. Maybe some abbreviations are valid only in the scope of a specific squadron ? That is the role of the `settings.yml` that is in the document folder itself.

With those settings, we are able to pre-process the Markdown document. Aliases are replaced withon the text, etc.

Finally, we need a global `template.tex` to create PDF. That template ensures that all documents received the same formatting. But, each document also has specific needs: their `media` folder will be at a different location, and some other settings may be different too. So we also pre-process the template using the current settings (and information gathered automatically, like the `media` folder path.

We now have a pre-processed Markdown, and a pre-processed template. We can use Pandoc to get a PDF out of those two, and voilà !

1.5.3 Centralized repository

All documents live in one repository, that is itself hosted on Github.

The advantages are plenty: * Automated recording of all changes * Automated and constant backup * The library is accessible to all; everyone can suggest a change or add content * review process greatly facilitated: changes are incremental, and their authors are clearly displayed; discussions about those changes are centralized around the pull request; merging the changes once they are ready is a one-click operation * concurrent edition: many people ca work on the same document at the same time, sharing their progress along the way (conflicts are handled on a per-line basis) * ease of access: editing a document or suggesting a change is available in all web browsers * **internal links and bibliography**: a document may include another, which may in turn include another, which may in turn ...; this makes it very easy to propagate changes in all "sub-documents" to their "parent" document (for example, a procedure that is commont to the whole 132nd Virtual Wing might be outlined in a dedicated document, which will be "included" in most TRPs; updating the sub-document would then upgrade all associated TRPs); building a bibliography that references all documents published by the 132nd Virtual Wing would be automated as well, allowing for easier reference to other documents * media files like pictures, logos, maps, etc. are shared across all documents; updating them propagates to all documents using them

1.5.4 Extensibility

Once the content has been created (i.e.: the Markdown texts are written), we can output in a lot of different formats. This means that if, at any given point in the future, we find that we would need to publish books with our documentation on the Google Play Store and its Apple counter-part, it would take us about 5 minutes of manual work. The *content* is there already, all we would need to do is add the *output* to the pipeline.

This is of course a silly example, but I think extensibility is still a very strong pro.

1.6 Technical

This section describes the tools chain that would be used to build and publish the documentation.

1.6.1 Overview

Despite the vast possibilities that those tools permit, with a staggering amount of options, configurations and features, the whole process will be mostly automated, and the editors/reviewers will have to deal with very few technicalities.

1.6.2 How it is built

1.6.3 Specific tools, part1: the front-end

The front-end is what editors will have to work with.

1.6.3.1 Markdown

Markdown is a markup language widely used across the Internet. Its syntax is simple (similar to the syntax we use on our forums) and is meant to be readable. This very document is written in Markdown; click here to see the "raw" Markdown text.

Excerpt from Wikipedia:

> Markdown is a lightweight markup language with plain text formatting syntax. It is designed so that it can be converted to HTML and many other formats using a tool by the same name. Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor. As the initial description of Markdown contained ambiguities and unanswered questions, many implementations and extensions of Markdown appeared over the years to answer these issues.

Here is a cheat-sheet with the available formatting.

1.6.3.1.1 Why Markdown ?

I selected Markdown mainly because **it decouples the content and the format**, but also for the following reasons:

- It's readable and easy to "learn"
- It's used all over the web; tutorials are plenty, and all questions have been answered
- It's supported by all the major players
- It's very easy to transform into PDF, HTML, EPUB, RST, Microsoft Word, etc.

1.6.3.1.2 Try it out

You can try Markdown right now, without installing anything. Just head to one of those online editor, and write away:

- dillinger (my favorite)
- classeur
- stackedit
- jbt.github.io/markdown-editor
- markdownlivepreview
- hackmd
- etc. . .

1.6.3.1.3 Offline editors

If you prefer to work offline, there are tons of editors for Markdown:

- Notepad++ with plugins

- Atom with plugins

And millions others...

1.6.3.2 Git & Github

This is where things get a little bit hairy. It will seem very complicated at first, but I can assure you that after doing it a few times it makes a lot of sense and becomes actually quite easy (we're in the business of simulating a very complex environment, I reckon a few commands won't scare you that much =)).

Here is an introductory tutorial about Git & Github.

Here is another tutorial about the Git philosophy, and one way to use it.

Please don't be scared by all the commands, there is a GUI application that lets you do all those things with a click of the mouse.

I selected Git & Github because it allows for a *outstanding way of working together on the same project.* With Git & Github, you can:

- Track *all* the changes, and roll back whichever one you'd like (constant backup of everything)
- Associate changes with their author at a glance
- Collaborative editing (many people can work on the same document at the same time)
- Incremental review process supporting discussion
- Issue tracker
- Allows for a lot of automation under the hood
- Git is the *de facto* Source Control Manager nowadays (alternatives: Subversion, Mercurial; compare them)
- Github is the *de facto* hosting website for open source Git projects (alternatives: Bitbucket, Gitlab; compare them)

Note: I'm leaving out alternatives that are not open-source, self-hosted and free to use.

1.6.3.3 What you'll need

This is the list of what you would have to install on your computer in order to be able to work on the documentation.

1.6.3.3.1 Option 1: work online only

This is absolutely possible. For example, with http://dillinger.io/, I'm writing this document in Markdown, I get to have a live preview of what I'm writing, and I can push it to Github with one click of the mouse. No fuss, no problemo.

1.6.3.3.2 Option 2: work offline

While option 1 is perfectly fine, you can also decide that you need more control of what is going on. In that case, you'll need a minimal suite of tools.

Edit Markdown

This one is easy: you don't need anything. Markdown is pure text, so any text editor will do

Work with git & Github

First, make sure you followed this tutorial and are a little bit familiar with Git.

Then install the following: * Git for windows * Github windows client application (alternatives: smartGit (my favorite, more complex), Gitkraken)

That's it ! Create an account on https://github.com/, start your local Git client, and you're ready to rock !

1.6.3.4 The work flow

1.7 Transition

Transitioning from the current documentation library to EDLM will not be painless, but it should not be painful either, nor take an unreasonable amount of time.

I'm providing tools to convert from Word/PDF to Markdown, but that process is error-prone. So, far, the issues I've been able to identify are:

- The format
- Tables
- Lists
- The pictures (the pictures are correctly extracted from the Word/PDF document, and put in the media folder, but their format and/or positioning might need to be adjusted) On big document, like the 617th DAMBUSTERS SOP, this could take a couple of hours. On more simple document, it might take only a few minutes.

1.7.1 Process

The actual transition would happen like this:

1. Freeze the current state of the library (no more edition of current Word documents)
2. Convert documents in Markdown
3. Build a sane structure for the new library

- Place pictures in the correct folder (maybe some of them should be shared between document? Or even maybe with the whole Wing?)
- Define settings and metadata for the documents (title, authors, ...)

4. Build the new library incrementally, green-lighting documents one by one until they all pass QA

In all honesty, I'm unable to give an objective estimate of the time this will take, mainly because I don't know who will be able to spend time on this. Older documents, whose maintainers are gone or inactive, can be updated by myself or anyone willing.

1.7.2 Crash course

I plan on offering crash course with Git/Github/Markdown to the people who are interested in it, and to write a few pages long tutorial to help to get newcomers started as well.

The crash course should take about 30 minutes on TS, getting people to install the necessary tools, and get started with them ("hands on" tutorial).

The written tutorial would be much of the same, covering the installation of tools, the publishing process and a little syntax related guidelines.

List of Tables

List of Figures