Knowledge Markdown Workflow

Presentation No.1 (Rev. 0.5)

Charles Zhang

February 7, 2020

Abstract

In this article I will introduce a workflow that targets efficient management of personal text data, in an effort to promote effective knowledge acquisition and discovery. This presentation will not cover the whole workflow (which isn't complete and not mature yet), but just to give a quick taste of what it is intending to solve, and its current approach and progress.

Overview

The digital age is said to bring an explosion of information. To me, that's an understatement of the impact. The amount of information doesn't simply surpass the capacity of human processing since 20th century - it has been so ever since there is written records of human activities (or even earlier). *Knowledge has always been a fortune possessed by more well informed minds.* The difference Internet makes nowadays is that information is cheaper and easier to access - instead of going to libraries and finding paper documents we can obtain them easily online. However we must recognize: 1) The abundance is filled with redundance, 2) Truly valuable data and information - or any knowledge for that matter are still well protected and expensive to obtain. I propose individual investigation into formulating one's original ideas and understanding of the world, starting with the information we have at hand. A greater utilization of information that came across us means to extract value from such abundance of information instead of getting *overwhelmed and distracted* by it - even if the information itself is considered incorrect when compared with more correct version of it, it can still serve as evidence. The first thing is to **consume** information, then we need a way to **digest** it. Clearly to handle this we need mechanisms to help us **organize information** - actively, rather than rely on Google (the reason for this is *it's too late when you need to search on Google* for information to be useful, though that needs a bit more justification and I do not intend to get into that just yet in this article). Bookkeeping of information is hard, much as accounting is hard - to ensure consistency and completeness among all records requires not just awareness and the ability to identify information, but also training and descipline of personal habits, which makes this practice essentially a form of art - in this case not necessarily aesthetically appealing but epidemiologically inductive to our future thinking. I am working on developing a framework that makes this process easy and scalable. In this article I shall discuss one of the recent discoveries I have made.

Jump to Method and Conclusion sections **now** if you just want to see how it's done.

We cannot remember everything forever, but through training, I hypothesize it is possible to achieve what I call "**externalized memory**" - by utilizing digital means, we can put *non-critical information* into digital forms, and instead of memorizing data itself, we memorize the **structures of data**. For instance one used to memorize specific telephone numbers but with the advent of mobile phones we have contacts readily available. **The more structure there is to our data, the easier it is for data to be retrived and discovered**, so the first problem we are trying to solve is to **give structures to personal data**. This serves two purposes/advantages: 1) It relives our burden to memorize less cricial information and risk forgetting them; 2) It allows easier transferring and sharing of data and maybe even passing knowledge to next generation (since digital data are more transferable than analog ones). Traditionally when data takes form of digital contents, one would use a computer to manage them with underlying **file system**, or for organizations data usually take form of **structured records** in database. In this article, we are treating data as **"items"**, and it can take any form of the following: *ideas, sentences, quotes, statements, articles, regular files, multimedia contents, database records and websites* - the identifying characteristic is to treat those features/aspects/concepts as concrete **units of information**, irrelevant of its underlying representation or format. On the highest level, this can be divided roughly into: **texts**, **files** and **records**. Texts include *everything that we can articulate in natural language*; Files are *everything that we can store on disks*; Records are *things that exist in database.* My solution for managing files is *Somewhere* (Zhang 2019) and other related tools; I don't have any solution for directly interfering with database records yet. However, in this article we shall focus on **texts**.

Texts are defined as **collections of sentences that form meaningful expressions**. The articles we see online, the ideas that people exchange with each other, the writings that we compose - those are all texts. The fundamental unit of text is sentences, which consist of phrases and words. On the lowest level, we shall limit our attention to **sentences** only, and we must notice that sentences can be **typed or written as strings and thus stored in a computer** (this is in contrast to, for instance, paintings, which cannot be directly expressed in such granularity - paintings must be stored as pictures or photos, and its semantics is more ambiguous). What we intend to solve is to devise a solution that can **give structure to texts**, such that when we later wish to retrieve anything - be it a *specific idea, a concept, a discussion, a speech*, we shall be able to achieve it in a timely fashion. For more information on specific scenarios and limits of existing methods, see section on Motivation. We wish to organize our texts such that 1) we can recall it **almost instantly** when we need it; 2) we can easily elaborate, modify, expand and add remarks to our texts.

In the following scheme we deploy a mixture between **centralization** and **distribution**. **Centralization** refers to keeping everything in a single place (or to infitestimally approximate this; we shall note it's not always possible to do this, but we devise ways to work around that and make it work in general) and making accessing information - no matter what kind - as consistent as possible. **Distribution** means we want units of information not interrring with and not dependent on each other. Centralization enables greater oversight and coherence and inspiration (data discoverability) by putting everything hemogenous together (thus easier to extract similarities and underlying rules and patterns), while distribution enables more managable operations on individual units of information and it is realized through self-containedness.

Motivation

I used to learn from many different sources: *YouTube, online courses, textbooks, classmates, personal observations, lectures, tutorials*; And I take notes on *files, scratch papers, notebooks, Google Keep (or Evernote), Microsoft OneNote*; I create illustrations and record ideas and write articles on a wide variety of subjects, in different forms and places. To give you some context, we are talking about years of accumulated notes and contents, not just some piece of note I take while attending a lecture or watching a video - **we are really trying to manage all those knowledge in our life, not just something more specific and localized**. The worst part of this is my medium of note taking constantly change while I adapt to new tools and due to the availability of tools at hand (e.g. while I am away from computer I have to take notes on mobile phones or scratch paper), and nowadays I barely take notes on paper (because *I still have more than a dozen paper notes that I have no idea when I am going to convert them and make them "useful again"*, and also because *lots of paper notes are simply lost in all those years or not even accessible because they are far away in China*).

I used to have **layers of layers of folders** to contain my text notes (both on disk and in YunNote, the Chinese version of Evernote) - and these produce troubles that is beyond explanation so I won't even attempt to elaborate here.

What I actually wanted is really really simple: **I take notes once, and I should never bother anything outside the content of the note itself - I don't wish to manage archives, put things in folders (both physical and digital), or have trouble searching**. And I want that to universally apply to all notes I shall ever take: in the past, in the present, and in the future.

The derived scheme revolves around what I call "**tags**" - like subject names, and not like twitter hashtags. In this article I talk specifically about **text notes**, but in practice similar ideas will be applicable to **digital files**, and that is what I call "to transform information to knowledge" by **first turning those personal data into (identifiable) information**. The method I shall propose in next section is a framework to help **identify, extract, and manage information**. Dealing with other mediums is a bit trickier, but some underlying principles seem to apply universally, and I have successfully applied the idea to manage (website) bookmarks. I am also explorings ways to extend this idea to manage information from **physical books**.

What's the issue with traditional, bullet-proof, thousand-year-history, free-style pen & paper methods? First of all, I guess (without explicit evidence) *there are always more desciplined ways of making notes*, however some people just don't pay much attention to it as long as it works. My observation mostly lies in classroom settings (or maybe business/conference meetings) and textbooks: it is one thing to be able to present and copy presentations as is, another thing to be organized and be able to make annotations and record important thoughts, yet a totally different thing to make sure those information are *accessible in the future*. By that I mean **how long does it take for you to scan through a textbook in order to find that *one particular formula or theory* that you are interested in**? That is one of the more practical reason why we make notes at first place (on top of existing presentations, notes, and textbooks) - we try to make highlights and make things clearer, not just to make records. Some textbooks provide indexes for *digrams, keywords, and even formulas and theories* - however that is not enough, and we can do better. How so? When we think of that "one particular formula or theory" *we don't really care **where or how it was presented***, we just want the gist of it: we just want the "first law of thermodynamics", or we just want "the mass of earth" - Google does a pretty decent job providing "just the information you want" nowadays for things like "mass of earth": if you search that in Google it will tell you right away *5.972 × 10^24 kg*. It's not capable of providing an **exact answer** to searches like "first law of thermodynamics" just yet - it does have a summary card providing both an excerpt of the first paragraph from Wikipedia (which serves as **definition** for the scope of the term) along with a **formula**. **That's pretty much what we would expect**. Let's make it slightly more difficult: what if we want “the *history* of *mass* of *earth*, and all the *peoples* and *experiments* that are ever done on it”? No doubt **clever as search engine and technology can be, Google is not yet capable of answering that**. I am not going to show you how exactly that will work in this article, but the method I am going to present right now can be a remedy to that issue (the method below serve as just the first step in a more involved workflow), and it’s designed specifically for *individuals*.

Method

KMD in Files

When all of your personal information/data satisfy the following conditions (or converted into those formats): 1) *All notes are recorded inside* ***a single file***; 2) *All ideas are stated in a* ***single line*** *or a* ***single paragraph***; 3) *There is* ***no inherent logical order*** *between different paragraphs.*

We shall further convert them with the following rules:

1. Apply **a set of tags** to those lines or paragraphs according to the *topics contained within*;
2. Nest **relevant discussions under successive lines** by applying incremental layers of indentation.

The above format is herein termed "**Knowledge Markdown (KMD)**".

For example I have a file called "*Questions of All Sorts.md*" and it looks like this:

```
1. (University, education, software development) If all those design patterns are not taught at
school, where are people supposed to know and learn about it?
    * Idea from: Game Programming Patterns by Robert Nystrom
2. (Kungfu, Taekwondo) Defence pose: Why defence uses fists/arms instead of palms? It seems natural
to me when someone kicks you, you use you palm to dissolve his force, instead of hard contact?
3. (disease, epidemiology, 20200129, public health) So China has closed Wuhan for Jan, what does
it plan to do next? Without an antibody vaccine how can it settle this epidemic? On a relevant
note, what happened to SARS after 2003? Did it just disappear? Does that virus still exist on
chickens/birds and why it no longer seem to have any effect on us? For me personally, does that
mean SARS will never be able to infect me because I survived 2003? Is it necessary for someone to
get infected in order to in the future be able to fight with this disease or public health/government
just somehow eliminated that disease?
    * (remark, method) Currently we are searching answer by looking into books on Epidemiology.
    * (question, genetics) Anyone other than Chinese get affected by coronavirus and SARS? What
    role does genetics play in this case?
```

You should feel the immediate difference of this sort of note taking when compared to more traditional formats or fancier UI (User Interface) provided by Google Keep or Microsoft Sticky Notes. In this discussion you *shall be flexible with the aforementioned three conditions* in order for things to work (e.g. seperate notes in multiple files by some "topic" instead of stick with a single file), but in the actual framework (which is more than what I am presenting in this article), it has **more strict conventions** (and conventions make things efficient). I must also point out that **how to define tags is a descipline in itself**, and some notes are available in (Zhang 2019) (though it's not a polished article). It's enough to know that the tags **should be relevant to yourself**.

See Example section for a practical and complete example.

KMD on Paper

It's worth noting that the same process applies even if you are NOT using a computer, however you may need to adjust your formats a bit for efficiency of note taking. In below illustration I am going to show an excerpt of part of my lecture notes.

First you should have a blank sheet of paper to serve as **index page**, and write something like the following in your index page:

```
ECE 311 Introduction to Control System (Index)
1. Course
2. People
3. Reference
4. Strategy & Technique
5. Remark
6. Theory & Model
7. Equation
8. Subject Matter
9. Terminology & Annotation
10. Robotics
11. Question
12. Component
13. Example
14. Airi
15. Analysis & Case Study
16. Concept
17. Collection
18. Identification & Patter & Observation
19. Algorithm & Procedure & Process Trviality (Important Links)
20. Mathematics & Computational Details
21. (Final) Exam
22. Todo
*. (Somewhat) key, conceptual, new, surprising and significant
23. Global Entity Table
24. MATLAB
```

Then you can take notes (almost) as usual KMD files:

```
2       Kam (Incomplete name)
1 2     Systems Control Group UofT (Professor)
1 3     Textbook for additional reading and extra comparison
        This course is not covering the textbook.
    16 5 4  This might apply to other courses as well.
            Maybe we should learn to pick up self-contained fragments.
            11 *    Or is there a necessary contextual background link?
1       Tutorial is not required? Just check posted materials online is enough.
1       Homework are important. Attempts will be rewarded full mark.
8       Robotics, Aerospace engineering, ECE470 (Kinematics) (Movement Constraints)
10 9    Links, Joint (Robotics); Motor at each join. Actuator, Plant.
11      Why can't we just assign arbitary voltage to control robotic arms?
        Do we have measured information of current angle?
        Not robust enough? Can't we do it like moving our arms, sort of like applying
        constant adjustments?
        What's the issue with such simple schemes? Stability? High controller frequency?
12      Sensor is encoder on the motor: Angle and angular velocity.
    6       Current state is assumed known. (In practice measured results can differ from
            reality)
...
```

(Warning) You must notice that such a scheme as presented so far have a serious draw back - what if our notes contains diagrams and other information? When we are using Microsoft Words or plain scratch paper we seem not limtied to the layout of our notes, but in **KMD the line-based layout seems to restrict our notes to text-only contents**. I shall state here that **this is a side effect of a text-oriented approach**, but it does have **curious implications**. For solutions of this issue, I will mention it next time when I talk about the concept of something called **"Global Entity Table"**. Another serious drawback of this text based method is that it forbids **invention of notation** which is never ideal but with the help of **LaTeX** this can be remedied to some degree. In other cases, *a more flexible notational method is desired.*

You should also be aware that paper-based notes suffer from usual limitations of paper: you cannot easily make modifications (unless you write in pencil of course). I would argue for note taking purpose if you are using KMD you don't need that (why?).

What Happens After?

There are many details covered in KMD specification (which is a work-in-progress and not published yet), but here are the *three* most important parts:

1. A *Vipiler* for **automatic presentation**: Vipiler stands for "**View Compiler**", which converts your KMD notes into more familiar formats (it can be anything from TiddlyWiki to single-page searchable HTML app to PDF, depending on technology);
2. **Additional specifications** that help link everything else together (e.g. files, PDFs, images). Also the purpose of using KMD on paper is not to keep it that way - it's so that the recorded paper notes can be easily transferred to digital forms;
3. You can write **manual presentations** based on, but not as a substitute for, KMDs.

Presentations are meaningful for communication purpose, but the knowledge content is saved within KMD for ourselves.

Summary

In summary, KMD stands for "Knowledge Markdown", it's essentially a **text-based markdown file** ("Markdown," n.d.) with some deliberate restrictions on the format. KMD also stands for the **methodology** that's used to organize the contents in those markdown files using in-line tags, in this case specifically for note taking purpose, but can also be applied to other scenarios as well.

Through the workflow of KMD, an individual's organizing effort is turned into *investigating into the subject itself* (aka. I call this "*study* or *acquire/discover knowledge*"), instead of physically cataloging and preparing contents - which in our terminology here is called a **presentation**. This very article is *talking about* KMD, but it's not written in KMD itself - so it's just a presentation. For a document that's written in KMD, see Example.

Example

Below is an example of original draft (and the ever-evolving concept) of **Knowledge Markdown** (notice line wraps are marked explicitly with “->”). The details of this draft is not inteded to be comprehensible to any general reader because it involes things that’s very specific. It merely serves as an example of what a KMD document will look like.

```
--------------------------------------------------------------------------------
This document itself will be written in KMD.

* (observation, tool, Somewhere, MC) KMD (Knowledge Markdown) - or what we used
 ->to call "Indexed Note" collection - seems to encourage a very curious phenomena
 ->(in terms of its impact on Somewhere item): the more items we add to the KMD, th
 ->e more collective tags we will need to add to Somewhere note - as such we encour
 ->age outsourcing KMD outside a Somewhere Notebook, and put it as physical Markdow
 ->n file.
* (Tool, workflow, proposal, Somewhere, MC, conception, inception) (Original con
 ->ception in Tiddly MC, where we created first "indexed notes", then come up with
 ->"KMD" while creating more indexed notes) Somewhere though designed to handle Kno
 ->wledge I tend to use it for tagged large notes (i.e. articles) and it just feels
 -> more natural - for Knowledge what we can do instead is to just create a *CSV ta
 ->ble with 2 columns*: piece of knowledge, tags. Or something just like this one (
 ->i.e. this line), **putting all tags (case insensitive) at the beginning of line
 ->and content immediately follows on the same line** - only if we can find a way t
 ->o handle indented bullet points and numerical list for more natural look just li
 ->ke MD!
    * (Proposal) We can actually have a very restricted form of MD, let's call i
 ->t **KMD** (stands for "Somewhere Knowledge Format Markdown Document - With Stric
 ->t Conventions"), which is basically just MD (and for IDE purpose we will just gi
 ->ve it file suffix as `.md`), but with those *conventional restrictions* (notice
 ->some of the restrictions are relaxed in formal specifications of KMD): 1) The wh
 ->ole document must be a single numbered list. Notce this rule is relaxed in forma
 ->l specification (pending definition). 2) Whenever there is **a bracket with comm
 ->a separated keywords** at the beginning of an **item** it represents tags for th
 ->e item. 3) Each line, despite indentation, shall represent an individual **item*
 ->* - when there is indentation however, the item is linked to a **parent item**,
 ->the whole KMD thus forms **a tree of tagged nodes**. 4) By such convention, **ea
 ->ch item must have a tag**. 5) By such convention, ~~bullet points~~ (I meant "in
 ->dexed lists") really should be forbidden anywhere it's a **root item** (I.e. not
 -> a child item, aka. no indentation) - just like this item: we use numerical inde
 ->xing to enumeration parallel discussions **within the item rather than as siblin
 ->g** (Semantically this is for the purpose of **self-containment** and **complete
 ->ness** of items). 6) (This rule is completely deprecated) We further allow at th
 ->e end of the item an **entity link** which represent a semantical equivalence of
 -> current item like this: `{{Entity-Link Statement (undefined yet}}` - this is in
 ->spired by and so that we can deprecate (for concept organization purpose, rather
 -> than for geological management purpose) **Canvas**.
```

```
    * (Motivation) The purpose for a workflow or format for that matter, is I am
 -> thinking about programmatically accessible parsing and thus query and searching
 -> - but maybe that's a triviality and not necessary at all as long as the convent
 ->ion is followed and text format itself is clear.
    * (Concept, Discussion) Conceptually, all **child items** inherit whatever t
 ->ags that applies to its **parent item**.
    * (Technique) This is especially helpful with Sublime text which can expand/
 ->collapse subitems.
* (Specification) Paragraph Interpretation Rule: Only lines that begins with lis
 ->t syntax (e.g. `1. something` or `* something`) are considered as items - plain
 ->texts in a paragraph is ignored by KMD. Also list syntax are indifferent to KMD.
 ->
* (KMD, concept, distinction, definition) **knowledge/idea first approach** vs *
 ->*source first approach**: our original personal ideas e.g. those in MC and in Pr
 ->oject Nine **records ideas first, elaborate a bit, then record an "idea from" pa
 ->rt**, this is knowledge first approach; Our Images Media somewhere repository (a
 ->nd SW Repos in general), or PDF Apprentice annotations (or **annotations in gene
 ->ral**) are **source first**. Both serve referencing purpose. The distinction is
 ->the former is original, while the latter is largely building on top of existing
 ->information.
    * (KMD, management, personal, issue, proposal, unsolved, presentation, proje
 ->ct nine, MC, file, note, definition, concept, analysis, data, opinion) I am thin
 ->king that maybe it makes sense to have **everything I author** (including variou
 ->s projects and notes and annotations on PDF materials) (ideally) inside a single
 -> KMD - currently we are explicitly separate some contents by their **leading cat
 ->egory** (which is itself a tag) e.g. under folder ProjectNine - but the problem
 ->is: 1) We sometimes make a cross and mention stuff on MC in ProjectNine notes be
 ->cause they are tightly integrated, 2) In previous case we *should* just in-time
 ->make a separate entry in MC but that rarely happens, 3) technically speaking all
 -> my notes belongs to me so they should all reside under something like **My.All*
 ->*. Notice if we shall do that, we still need to be clear that this is **just for
 -> personal notes** - **for presentation purpose it's always advisable to have a s
 ->eparate folder with dedicated contents** (in that case and only on that case, cr
 ->eating a ProjectNine project folder for crafted visual contents starts to make s
 ->ense) - however the point being made here is during design, it might be helpful
 ->to lump things together all under **Me**.
    * (Remark) This note is out here simply because we first mentioned concept o
 ->f KMD in MC indexed notes and there is no better place to put further elaboratio
 ->n on it.
    * (Implementation) We can have something like (or even directly use) TiddlyW
 ->iki - allowing tag outlines and whole document search. What's more, it has trans
 ->clusion. We can just save as TiddlyWiki JSON and then import it. However do noti
 ->ce Tiddly Wiki requires a title which is not what KMD expects. However we could
 ->just assign an item ID.
* (issue, unsolved) A huge issue with current KMD - despite its cleanliness and
 ->effectiveness - is that it's limited to **single line of text**. This has the in
 ->herent incapability to express a text **item** that requires a multiline express
 ->ion (e.g. a snippet of configuration file), or even a simple diagram (e.g. a min
 ->d graph).
    - (proposal, solution, property) A partly solution is to put the *part* into a
 ->file, and put that file either into *Reference SW Repo*, *Textbook SW Repo*, or
 ->*GET Repo* - **effectily turning that thing into an item in itself**. With prope
 ->r **internal annotation**. However this applies only when that item is rather co
 ->mplete and informative and self-contained somehow, and doesn't apply for things
 ->that are really fragmented (e.g. a very few lines of a configuration file, or a
 ->draft sketch that's not meaningful in itself). We must notice the nature of such
 -> kind of **fragments** cannot be tagged (i.e. we don't want to tag it, and that
 ->we just can't tag it because it's meaningless) - otherwise it becomes a meaningf
 ->ul item which by definition means it's not a fragment at all.
        + (proposal, solution) One potential solution for solving the issue with not a
 ->ble to having fragments directly inside a KMD item is to create yet another kind
 -> of global repository - just like GET, but this time reserved purely for things
 ->that are meaningless, and **provides only numerical index ID**. However the mana
 ->gement overhead and workflow difficulty for this is not acceptable. Bedies, how
 ->do we store that? As seperate files, or inside a data base? Do we use Markdown f
 ->or multiline texts or what? And how in the future do we promote it? How do we kn
 ->ow who is its user and maintain meaningful connection?
        + (solution, preferred) Another solution is to use a **local fragment referenc
 ->e**: Assign a ID to the fragment being referenced, and provides a KMD document w
 ->ise definition for that fragment **at the end of the document** inside a dedicat
 ->ed header section `# Local Fragment Reference` or `# LFR`, then put whatever thi
 ->ng there directly. Specifically, we require: 1) The fragment is referenced **as
 ->is**, effectively copying-paste its whole content (though KMD Vipilers might hav
 ->e more meaningful ways to present it, e.g. as mouse-hover popups or rich text ob
 ->jects) into the *referencing site*; 2) All fragment under `# LFR` section is nam
 ->ed with a secondary header, which can be **either a numer or any valid text**, w
 ->hich must be unique because that's the name for the fragment; 3) The value of th
 ->e fragment is the content of the secondary section, be it a table, a code block,
 -> or just plain file path; 4) The *referencing site* shall use the syntax `{{LF:
 ->Unique Name}}` (i.e. indicate a transclusion) to refer to it.
            * (example) As an example of local fragment reference, consider 3 cases: 1) T
 ->able: {{Table Example}}, 2) Image: {{Image Example}} (Notice putting image refer
 ->ence inside the LF instead of using `![Inline]({{Name}})` syntax to avoid causin
 ->g rendering error by usual Markdown converters), 3) Multi-line text: {{Multi-lin
 ->e Text Example}}.
            * (Elaboration) For practical purpose, we shall be aware that LFRs can furthe
 ->r transclude other LFRs (e.g. See Example "University/(Project Analaysis) Assign
 ->ment UofT.md").
    - (remark, reference) A relevant discussion can be seen in `GEO`.
* (convention, deprecated) Before using LFR syntax, when specifically for multi-
 ->line fragments, we shall consider using this custom convention for **single-line
 -> in-line code-style representation of multi-line configuration files** (for huam
 -> nreading only, KMD parsing will not specially handle this in any sense): multil
 ->ine indicated by `\n`, `/**/` indicates custom comment.
    - (example) `[Nginx] /*Name of repository*/\nname="Nginx Repository"\nbaseurl="
 ->http://nginx.org/packages/centos/7/$basearch"/*The url yum will be using to sear
 ->ch for the repository, the basearch part means it's going to download a proper v
 ->ersion that suits the system architecture e.g. 32bit or 64bit, I guess this is s
 ->ubstituted by yum automatically*/\ngpgcheck=0\nenabled=1`;
    - (remark) I consider this an inherent issue/property of KMD notation itself: w
 ->e want a single line to represent **a single piece of information item**, but so
 ->metimes the information itself (though rightfully can be considered a single lin
 ->e) need to be expressed in multiple lines (due to the nature of file system and
 ->digital word) - in this case our only solution is to reference it somewhere else
 ->, e.g. in *Reference SW repo*, in *Textbooks SW repo*, or in *GET* - however we
 ->should not forget: 1) Having to referencing it externally in this case is not th
 ->e intention, but merely a compromise; 2) **Whenever something is externalized, i
 ->t means it have additional meanings or is standalone or self-contained in itself
 ->**, whihc may or may not be the case at all - ideally **it should be contained w
 ->ithin the context it's used when its only intended user is that context**.
* (tool, solution, proposal, name) The first KMD Vipiler would be a simple .net
 ->core application that gathers a bunch of KMD and generates a single HTML file (w
 ->ith scripts and css embedded) that supports proper searching (just like Tiddly W
 ->iki). It shall be called "WKMD" (Web KMD).
    - (implementation) For identification of tags, we can hide those information as
 -> custom CSS classes for each item (as as `div`) so it's easier to detect when us
 ->ing Javascript (i.e. JQuery `hasClass()`), e.g. as `kmd-tagname` (to avoid name
 ->collision with used CSS library e.g. bootstrap)
    - (remark) Talking about Tiddly Wiki, actually I was thinking about exporting K
 ->MD directly into Tiddly Wiki - but it is not clear how to interface with Tiddly
 ->Wiki to directly populate the tiddlers without require human clicking import but
 ->ton then using JSON format import.
    - (remark, philosophy, design, concept, MC, software, workflow) Notice KMD is o
 ->ur attempt to employ a "workflow-based" software development altitude, instead o
 ->f a "program-limited" (or "tool-dependent") software development (and content cr
 ->eation) altitude - which means we **try our based to build automation process up
 ->on existing solutions** instead of **devising and maintaining custom solutions**
 -> (the core idea of which is to *use markup based solutions*). As such, **the fra
 ->mework itself should enable efficient navigation**, without necessarily requirin
 ->g an actual tool being developed any time soon.
* (specification, Vipiler) Whenever a child item is present/visible, its parent
 ->item must be visible as well (we might want to hide parent item because of an ap
 ->plied tag filtering condition which matches only the child but not the parent).
 ->This is for the completeness of context.
* (specification, Vipiler, Implementation) Advice: When an item has nothing but
 ->a **transclusion**, it might make sense instead of using a popup, show that tran
 ->scluded content directly in the *item view area* (for instance one might transcl
 ->ude a poem as an item).
    - (feature, advanced) Similarly, as a syntactic sugar, we might allow **text se
 ->ctions** consisting of *only a single code blocks*, e.g. a collection document o
 ->f poems, with each poem as a seperate section and its tags as the header (the ti
 ->tle of the poem, if needed, should be placed inside the code block in this case,
 -> instead of putting it as header, because header is used for tags, this is also
 ->consistent with general header interpretation rule).
* (specification) Header Interpretation Rule: A header is generally considered a
 ->s a tag that is universally applied to all items within the section of contents
 ->under that header. Secondary and more layers of headers simply add tags to it. W
 ->e may call those "headers as tags" as "category" (which we consider as hierarchi
 ->cal structures of tags). Though such concept have no significant in KMD itself,
 ->and is only meaningful for presentation purpose. The only exception is **LFR** s
 ->ection, which have special usages.
* (methodology) KMD represents a seperation between knowledge (or information) a
 ->nd presentation. As an example: a book is a presentation, yet the ideas and info
 ->rmation behind the book can be expressed in KMD.
    - (idea, pending) Ideally, presentations should be compiled from knowledge, wit
 ->h some minimum amount of connectors (I am not sure how yet).
        + (remark) This can help avoid the price or penalty of "intertia of legacy" wh
 ->ich means "if someone starts with presentation and intend to develop idea around
 -> that, then his ideas will be limited to the format, and with ever more expansio
 ->n of the idea, the format will become harder to change dramatically". Which may
 ->or may not be true, well I am just saying this might be the case. Any way, sourc
 ->e control will help to some extent.
    - (remark, application, observation, pattern) KMD intends to capture **the natu
 ->ral growth of ideas**, rather than *formal presentation of concepts* (which shou
 ->ld be organized in such a way that's more friendly to readers, who expect *appar
 ->ent structures* so there is *less to memorize in working memory*).
    - (theory) The same set of knowledge can be presented in many different ways. (
 ->Example) It's like the same subject of micro electronics might be organized in d
 ->ifferent ways and treated in different books - but the "core ideas" are the same
 ->. KMD is intended to capture only this "core ideas".
    - (theory) As such, KMD is **structure-free**, which gives its **self-contained
 ->edness**. (Remark) Notice the sole purpose of parent/child item is to save effor
 ->t (so we don't have to repeat the inherited tags and small semantic contexts), n
 ->ot to create structure.
        + (remark) In a sense, such nature makes KMD sounds almost functional (program
 ->ming language paradigm) - and it's procedural counterpart is step-wise instructi
 ->ons of procedures (which we usually express as embedded collapsed indexed lists
 ->inside the item using `1) ... 2) ... 3) ...` syntax). Notice **parent/children r
 ->elationship don't imply order among siblings**.
* (specification) Title/Filename Interpretation Rule: A filename (without extens
 ->ion), or we call it a "title", is considered (and parsed and added to all items
 ->within the file as) a tag.
* (specification) Special Section Handling: For a section named "**General Index
 ->ed Notes**", it's ignored as a tag (this might be useful if for aethetic purpose
 -> we don't want to start a document with lists directly, but instead want to have
 -> a section header).
* (convention, vipiler, functionality, requirement) When using text-based comma-
 ->delimited filter specification of tags, we should allow begining a tag with `!`
 ->to indicate an exclusion. In which case we should avoid using `!` for our tags.
* (idea, pending, issue) Considering the overwhelming amount of **online informa
 ->tion** and normal textbook contents (along with their PDf versions), news, etc.,
 -> it can be hard to make sure *efficient consumption and complete collection*. I
 ->am thinking about something like a **"digital footstep" workflow/automation**, w
 ->hich records - in an information oriented way - the things I have visited, in a
 ->**structural manner** (most likely identified by tags and order). However how to
 -> do this efficiently and in a complete manner is a question - something like Mon
 ->ey Manager mobile app is already hard enough to use because **recording simply t
 ->akes too long** and like accounting, it takes **too much descipline**.
    - (example, proposal, workflow) For instance when I was browsing all that many
 ->documentation pages for React JS (as browser tags), it might not make sense to a
 ->ttempt to **record those by hand**. In other cases, when I am done with the page
 ->, we will usually save it in Bookmark Ninja already. Maybe it thus make sense to
 -> instead of having ONE centralized place for my own digital footprints - spread
 ->it out in context specific environments?
* (convention, somewhere) To be in concordance with Somewhere's scheme, we now a
 ->ssign special format interepration to file names: filenames used to on its own (
 ->without extension) denote a tag, now we have 3 rules to further divide this: 1)
 ->A leading bracket with comma seperated keywords can be used to create file-scope
 -> tags for all items contained in the file, the rest of filename is treated as us
 ->ual; 2) A filename (exclude tag specification) with a leading `@` symbol dismiss
 -> the file name itself as a tag: e.g. this is useful for some very generic file n
 ->ames "General Indexed Notes.md"; 3) The file extension itself will be ignored, a
 ->s usual.
    - (remark, deprecated) Notice before KMD comes out, we used to use square brack
 ->ets with semicolon delimiter for file name based tag definitions, e.g. `[3D Mode
 ->ling; 3D Animation; 2D Animation; 3D Sculpting] blender-2.79b-windows64.zip`.
* (rule, vipiler, specification) For Global Entities (see **GE1**) we might also
 -> want to perform a pre-processing pass to repalce GE references in images to val
 ->id file links. This function should be optional (as an extension to basic KMD vi
 ->piler functions) in case the user machine doesn't contain valid GET repository r
 ->eferences.

# Terminology
```

~~~
## Abbreviation

* LFR: Local Fragment Reference;
* KMD: Knowledge Markdown;

## Definition

* Item: A self-contained smallest unit of expressive information, a piece of not
 ->e. **An item must be tagged, and anything that's tagged is considered an item**.
 ->
* Annotation: A piece of tagged text.
* Internal annotation: An **annotation** within a media file (e.g. an image, or
 ->a PDF).
* Fragment: A partial piece of information that isn't considered an item by itse
 ->lf, and must exist within the context of an item.
* KMD Vipilers: A KMD viewer/compiler, that can parse a single or a set of KMD d
 ->ocument and provide a unified view or output for navigating, viewing the documen
 ->t(s).
* Transclusion: An application of a local Fragment Reference through transclusio
 ->n syntax `{{Name}}`
* Local Fragment Reference: A multi-media type reference of a resource that is n
 ->ot expressed directly inside the item.

# Local Fragment Reference

## Table Example

-------     ------ ----------   -------
     12     12        12             12
    123     123       123           123
      1     1          1              1
-------     ------ ----------   -------

Table: This is just an example

## Image Example

![Popup Image](https://source.unsplash.com/random)

## Multi-line Text Example

```c#
namespace Test
{
    class My
    {
        public static void Main(string[] args)
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```
------------------------------------------------------------------------------------
~~~

The reader shall notice the following:

1. The original document is written using **Markdown** ("Markdown," n.d.), and shall be edited and browsed with usual Markdown tools or plain text editor;
2. It's just like a large collection of ideas, however its seemingly messy structure **coherently expresses some unified whole**;
3. It's **organic**, in a sense that whenever I have new ideas I can just keep adding to it, without bothering or modifying other parts of the document;
4. Due to point 3, it's very efficient for **source control** purpose;
5. Surprising enough, it's quite easy to **navigate and search** through this document - for someone who knows what it's about, i.e. me; More surprisingly, with a Vipiler (no implementation is available yet, but the process is quite logically clear), it can produce contents such that it's easy for others to read and explore as well[1].

[1] To be fair, it will require some minimal clean up of note items to make them more self-contained and to avoid confusion, but no big change is needed.

Advanced Compoents

Readers might wish to skip this section if she already has difficulty appreciating contents presented before, and jump directly to the section on A Brief Analysis of Performance of KMD When Compared to Traditional Methods of Note Taking.

As mentioned before, there are 3 types of **units of information**: texts, files, and records. At times, we might also want to deal with **physical entities** (*textbooks, resources, locations, people, and even materials*). The later subjects can be *identified* in part using databases, as is already done in real world (to give a few examples: **GIS systems**, **1000 Genomes Project**), and with some effort it's not hard to conceptualize a world where textbooks are digital (e.g. as PDFs) and people's information are described digitally (e.g. in database) as well. So in essence, **we are back to deal with texts, files, and records after all**. We want a way to **extend the idea of applying KMD on texts/personal notes to applying on files and records**. Since the key idea of KMD is to tag information, we want to be able to **tag files and database records**.

One might argue such realization might depend on more systematic integration of the tools on the OS level (Project Nayuki 2017), or for things like PDFs it requires native file format support (as what Adobe Acrobat DC is doing). However as is **proven through practice** (and I don't really want to argue why *it's as efficient as it can be without needing to go that low level*) (Zhang 2019, 2020) it's possible to **achieve tag-based annotation simply by applying a meta-layer on intended targets**. For instance Somewhere achieves tag management by tracking specific files, and PDFApprentice achieves tagging knowledge by utilizing a secondary shadow file. Better implementations exist to make things even cleaner, but the basic idea of **using metadata instead of modifying original contents** is proven to work.

Now let's consider two scenarios and see how KMD/tagging can be applied.

Scenario 1: Attending a Meeting About Something New (KMD)

Let's assume I am attending a lecture on business management and macroeconomics, which I knows nothing about. Or in your case you might be attending a GDC conference talking about using Houdini for procedural generation of forests and curved roads for Unreal Engine 4. Or watch a vide on microbes and living habitats of tardigrades.

First, it's important to recognize when applying KMD (i.e. if you want to apply it efficiently), you must **be sensitive about "information"** - and what is considered "a piece of information". In essence, everything you hear, i.e. **every sentence that people are uttering, is a piece of information**. I know we are taught ever since primary school that we must be able to filter information and in a sense, we apply filters as *culture, language, values, beliefs, attitude, expectations and intentions* (Treasure 2011) to decide **what's relevant** or **what's more interesting**. Well in the practice of KMD, **everything is relevant**. So that translates directly to: **every single sentence you hear, you see, you read, is relevant**. Just in case that's a bit too abstract, here is an example: upon hearing the word "tardigrades", I will immediate record in my carried notebook some line that goes like this: `(terminology, microbe, pending) tardigrades`.

Now in the ideal case, you will record every single thing you hear, pause for a moment to think about it, then tag it. I assure you with that practice you will **very quickly realize what is redundant** about the things you encounter. But that's too much and not exactly real-time, and certainly don't suit well in a conference or during a lecture. It might seem as such it's impossible to record such amplitude of information. Assume we are just obsessed with making notes of everything - which we should - there are 2 ways to work around that: 1) Precategorizing, 2) Discarding. **Precategorizing** refers to the action of classifying in advance the things we are going to encounter into categories or subjects. This can be a gradual process as we learn more about different things and subjects that exist; **Discarding** refers to consciously letting information go away. With those two instruments it's not hard to see how we might just record keywords and still be able to record most things that we decide to keep, in a meaningful way[2]. **Precategorizing is not possible with traditional categorization methods** because it's *hard to translate to concrete recording mediums*. With training **discarding can become second nature**.

Depending on the time you have (and maybe the energy), you don't actually record everything. The point here is that you should **be conscious about everything you hear**.

Scenario 2: Converting Existing (Paper) Documents (KMD/Tagging)

As mentioned before, free style paper documents contains a lot of **non-textual contents**: *diagrams, tables* etc. Besides those, it can also contain **noise**: *document borders, background, decorative graphics*, or simply *rhetoric statements* and *literative figures of speech*.

One must decide this before taking any action: **do I seriously want to keep everything that's on those documents**? Now this seems like a tradeoff: it might appear as if if we just take away some "key contents" and discard others we might be losing some "potentially useful stuff" or for some weird obsession you just wish to photoscan the whole document whenever you can[3]. That is not going to help. Ultimately, that *CAN depend on your purpose* - to a limited sense - but I claim here, since we know we are doing this for a greater good, i.e. our future sanity, you totally don't need any of those things that are not of "knowledge's concern". **A picture might be interesting, but if you cannot assign any useful meaning or subject to it, it's useless for you** - unless of course, you are really doing it for the **purpose of collecting**. The bottom line: try to keep only original contents.

After you have decided what to keep and what not to keep: 1) texts can be scanned, OCR-ed (Optical Character Recognition) or hand typed as KMD, and 2) Diagrams and tables can be scanned, transformed, rewritten using markup languages, and preseved as files. In worse case you might just record where the document is in a database and carefully catalog it physically. To link those two (i.e. KMD file and scanned files), use **Global Entity Tables**.

[2]As a reminder, one can look at the example of my control system lecture notes.

[3]You can do that, and that's very legit - however for some more reasonable purpose only; And you should do that with the direct help of or at least the guidelines of Somewhere (Zhang 2019).

A Brief Analysis of Performance of KMD When Compared to Traditional Methods of Note Taking

Discussion

In this section I will briefly enumerate several obvious advantages of the method proposed above, and also talk about one potential disadvantage.

Before we talk about the advantages though, it's critical to realize one difference between KMD and traditional note taking methods (i.e. Evernote and OneNote and paper note): **we don't have titles for notes, and that's important**. It's not like saying "well Evernote does support BOTH notebook based and tag based organization" or "technically speaking you can be as flexible as you want with OneNote because it's just a blank canvas after all", and presume that "titles" can be added as a feature just like Evernote has or what we sometimes do when we start drafting on paper as if it's going to become something significant - for the former it doesn't make sense to throw in every mechnism that's convenient without restriction and without thinking about consequences and lose your own unique approach, for the latter it just plainly lacks any well-curated structure after all! *It's better to postpone naming things, especially when it's not necessary.* One **fundamental assumption** for the design of KMD is that: **Ideas don't need a title; Presentations do - at which time it shall become a legitimate tag.** For now let's just call it a "hypothesis", since I don't have enough theoratical gound to back up this argument yet - but empirically, that works naturally for me (after I have committed to this). This doesn't mean we should avoid naming things, but rather it discourages abusing names for things that don't really need a name - in those cases tags should be sufficient to identify them.

Below are the obvious advantages of using tags:

1. Tags are much more **scalable** - to add or remove or clarify/modify classifications or subjects for items involves only *one action* - change the tags declaration in the leading brackets of the item;
2. Item based thinking promotes **small unit of information** - the requirement of self-containedness allows one to better focus on *articulating specific ideas* rather than bothering formulating logical arguments, in a sense, *structure determines function* of notes in this context;
3. Easy **in-time classification** - to classify an idea, there is no need to bother additional steps to create any folders etc.;
4. Tags carry **semantics** - tags alone is suffice to help us identify the idea itself, tags can be as generic as it can (e.g. "to-do"), or as specific as it can (e.g. "20200206"); tags can serve directly as searching instruments (because they have identification properties);
5. The loose form non-presentation bound structure naturally leads itself to **freestyle addendums** - there is no need to consider *where* is appropriate to put additional information because there are **only two places** to put *relevant new information*: either at places where it's relevant as indented children, or just put at the root (i.e. as siblings of other root items) with enough tags on it making itself self-contained. We can easily elaborate, modify, expand and add remarks to our texts without affecting presentation what so ever (because presentation should be in a seperate file);
6. We can **recall a piece of information almost instantly** when we need it - as a practical example, because this process necessarily involve some "mechnical steps" (assuming we are talking about "searching" using a computer), we no longer need to go through folders of files, nor do we depend on external tools (e.g. Evernote),

because we can just perform plain "full text search" and since it's text based it's super fast; Also we can search by tags and criterias (especially with the help of Vipilers);

In terms of the disadvantage of the method - **it's disruptive, it's untraditional and looks chaotic**. When we try to think about specific ideas as they are instead of just treating them as an illustrative evidence for the support of a bigger argument (because we try to contain and complete the idea while we are recording it), we naturally would be inclined to look deeper into the idea itself. It can appear as if a single document concerning some naive treatment on "waste water processing in Hongkong" somehow get hooked mysteriously on the subject of Kuwait's water processing infrastructures and keeps talking about more and more details about it in succeeding indented lines, which is nonthless permitted (and even encouraged) by the structure of KMD - because unlike traditional requirements of "focusing on one subject", we are suddenly elaborating on relevant subjects as well - directly within the writing we are working on. Such a format requirement makes it rather unconventional and might require some time to get familiar with.

A KMD document certainly doesn't look anywhere like a traditional piece of writing where we have clear headings and section divisions here and there, all seems to ultimately serve and support the title of that writing, and for this reason it is *not suitable for presentation purpose*. You shouldn't present your KMD document directly to an audience and expect her to have the patience to dicipher all the intricate thoughts you had. However I think this is a rather good thing: it completely seperates *the process of idea gathering* and *the process of preparing presentations*, it also facilitates breaking down one's problem into self-contained pieces rather than feel restricted by what should and shouldn't be put into your "draft" because you are confused by whether or not it's a "draft" for yourself, or it's a "draft" of a presentation for someone else - **a KMD document is strictly for oneself**; it thus also eliminates superficial logics that tend to treat things as if they are a sequential whole because people reads a document from top to bottom - in KMD there should be no inherent logical order for the document except maybe the indentation between lines for a somewhat narrative inclusive relationship. I claim that with some practice, this is more flexible than starting directly by writing a "draft": when all evidences and ideas of a piece of work is clear, we can decide to present them however way we want, and customize the presentation suitable for particular audiences.

Economy

One of the reasons - and it's a crucial one - KMD is **text-based** is that **this makes it not tool dependent**. This is inspired by the ubiquitous use of text files on Linux and for Text-based user interface applications, and by the success of the Internet. There are other uncountable number of benefits to this: *human readable, easily interpretable (for machines), flexible and extensible, compressable...* Unlike commercial tools like **TagSpace**, it creates no dependency on software infrastructure, and is thus free from software errors.

The storage is per-character: you pay as much storage as you actually need for the contents that you actually author. For an article that's worth *1000 words* and assume *5 letters per word* and encoded in *Unicode* (i.e. 2 bytes each letter; Unicode supports both English and other languages), the total storage is only: $\text{Total Storage in Bytes} = \text{Word Count} * \text{Avg. Word Length} * \text{Letter Byte Count} = 1000 * 5 * 2 = 10Kb$. If you create a Microsoft Word file, **without writting anything** it's already taking *12Kb*. For a 10000 words document when stored in plain text it's *56Kb* (*25Kb* after compression), versus *47Kb* in Microsoft Word (due to compression), and *215Kb* in PDF.

This one is a bit hard to argue because it depends on efficiency of individuals: **the time it takes to author the same content** is reduced when using plain texts comapred with using GUIs (Graphical User Interface) because it avoids tweaking parameters in a GUI (also depends on shortcut support of the editing environment and availability of automatic layout), letting alone free from all the distractions of a typical GUI application.

And here is my favorite: you can edit it on your mobile phone and sync with your other devices with a typical cloud based file synchronization setup (e.g. with the help of a free Markdown editor app on Android).

Speed

Nowadays companies come up solutions that can perform full text search in **PDFs** and **MS Words**. Putting the economical implication and environmental impact of such efforts aside, it goes back to the problem of "finding a formula on a textbook" - you can do it brute force and from first page to last page, or you can be more intelligent with it. Anyone with any experience trying to find a keyword by performing a full-text search in a PDF document will understand what I mean. Such methods just don't "understand" you. Why? **PDFs and MS Words are**

for presentation purpose, they are not designed for efficient navigation of information and ideas[4]. Now, we must recognize for large companies and organization it might indeed be more efficient for them to do that - because training people might otherwise take more effort and investment - but for individuals, I would perfer making ourselves more intelligent instead of relying too much on tools.

As given in next section on Tools, when appropriately applied, finding any specific collection of items takes usually less than 10 seconds (including the time it takes to type keywords, excluding software setup time). Also as a general analysis, when N items are categorized under M categories (i.e. M tags), I will argue without giving evidence that M is usually much smaller than N[5], i.e. during a searching if we use tags as criteria instead of keywords of the items the size of the (searching) problem is much smaller[6].

[4]I wish I am wrong with this - but I am not - and I do seek a solution that can harmonically merge the best of two worlds together. I have some initial concepts for that solution, but it's still in its infancy.

[5]It sounds like an interesting research problem **how many tags it is required to effectively classify N items** and **the impact on memory recall when using categories instead of keywords**.

[6]I would also argue it is actually cognitively easier to do it this way: when you forget something, it's more likely you cann't recall the details of it, but it's very likely you know **what (categories) it belongs to**.

Relevant Tools

Here I will present a rather complete process based on Windows operating system that is capable of integrating KMD into usual workflow. This can serve as a reference solution setup to illustrate the concepts described above. I am by no means affiliated with most tools mentioned below and most of them are free of charge or open source and they serve just as a reference implementation.

In this setup we deal with the following specific items: 1) All text notes; 2) All personal image data; 3) All PDF files; 4) All generic files; 5) All web pages. All those 5 types of items are unified under the same classification scheme of a common set of tags.

1. **Bookmark Ninja**: An online Chrome extension or web app called "Bookmark Ninja" is used to save website links as bookmarks under tags;
2. **Everything**: All generic files utilize a very specific naming scheme as "`(Comman delimieted tags) Filename/Foldername`"; A tool named Everything is used to search instantly on any NTFS filesystem to get those files;
3. **Somewhere**: User can put all files with arbitary names inside a single folder called **Somewhere repository** (just like a git repository) and apply tags to those files through a program named Somewhere (Zhang 2019); This mechanism can be used to keep *all image files (or videos) in a single place.*
4. **Sublime Text** + **Markdown Editing** Package for Sublime Text + **MD/KMD** + **Pandoc** + **TeX**: User can create and edit text note files using Sublime Text and export directly as PDF or Microsoft Word formats using Pandoc ("Pandoc Markdown," n.d.).
5. **PDFApprentice**: Tools like PDFApprentice (Zhang 2020) or Adobe Acrobat DC can be used to **create annotations for PDFs** however so far as I am aware only the former supports easy export directly into KMD format and natively encourages single item based annotations with tags.
6. Notice putting brackets and comma delimited list of tags in front of file names can confuse file explorer's default sorting; A custom **file explorer** can make "*sort by file name*" make sense again, or by all means, put the tags *after* actual file/folder name.

The reader needs absolutely none of above tools to start implementing KMD scheme on her own. As a reference, I have over *10,000 bookmarks* managed by Bookmark Ninja under over *2,000 tags* and retrival time is in *seconds* (due to network delay since it's an online app sometimes it can take *10 seconds* to filter through stuff); Everything is capable of finding any given item with partial names instantly; Somewhere can handle *5,000 items* with *2,000 tags* in less than *2 seconds* on a moderate machine; A usual text editor can handle text files containing over *thousand lines of items* with ease. PDF Appretice can handle PDFs that are within *50MB* in size and contains no more than *600 pages*. **Lots can still be much improved and the process can be futher optimized and streamlined** but it can serve a good starting point.

With above setup it's not hard to see how one might devise a **single set of ever-expanding collection of tags** (I estimate by experience something like *10K tags* should be easily achievable before existing solutions cause significant searching delays due to lack of optimization) to consistently apply to manage and navigate through all those diverse types of items. Since no Vipiler implementation is available yet, no specialized views can be generated for KMD format at this time. One might also notice **GMail** and **YouTube** seem to support adding "tags" (called "labels") to emails and videos but it's nonetheless not considered a viable option in our workflow because: 1) There is no obvious and integrated ways to *export tags* along with selected information for knowledge discovery purpose; 2) its *tag management* scheme is inadequate (partly because it's not designed for that).

Further Research

It seems there isn't as much research into **personal data management** beyond the scope of privacy and security issues[7]. In most cases when people talk about "personal data" it seems concerning more with personal identification information and account and banking information, rather than the **knowledge content of personal data**.

While drafting this article I came across several terms and wikipedia articles on Citation Style Language, Information Management, and Information Overload. Study the context and solutions related to those concepts might be helpful. In particular, in the article on "**Information Management**" it mentioned the following subjects of conern: "Information management embraces all the generic concepts of management, including the **planning, organizing, structuring, processing, controlling, evaluation and reporting** of information activities, all of which is needed in order to meet the needs of those with organisational roles or functions that depend on information" and "Information management is closely related to, and overlaps with, the management of **data, systems, technology, processes** and – where the availability of information is critical to organisational success – **strategy**."

In terms of selective note taking as mentioned in Scenario 1, there is something called "**theories of selective attention**" (Look in Khan Academy, Simple Psychology, Stages of Listening, Barriers to Effective Listening, or Cognitive Psychology in general) that might be worth checking out. A relevant term is Information Filtering System.

One last thing that's quite prominent and we haven't addressed properly is **emails**, and for that I do think **a custom (software) solution is needed** (though as advised this doesn't need to be a completely new software but more like a secondary layer based on existing solutions) because according to my survey so far no adequate solution is available yet (for the particular need I have at hand).

Also in case you have noticed but didn't realize, this article is written using explicit typographical *formatting emphasis* to highlight certain terms and statements and thus makes certain ideas more evident and accessible. I consider this part of **presentation** concern regarding information management.

My use of "traditional classification methods" is somewhat limited and it mostly refers to simple catagorization using hierarchies. Clearly there are researches and subjects in library categorization system (Wiki on Library Classification and Categorization) that is worth some attention, and who says techniques in neural network can't be inspiring to our personal note taking as well?

[7]Does this has anything to do with money? Seriously, what's wrong with money?

Conclusion

All personal texts should be stored in KMD format, and there is no need for **any further categorization**. This scheme can be incorporeal into existing schemes of note taking as much and as little as one desires and judging from my experience so far: gradually it will appear more attracting and natural.

Additional Reference

Ma, Shanshan. 2010. "Using Hierarchical Folders and Tags for File Management." https://pdfs.semanticscholar.org/2214/c72fc6bdc42f435eb29240e9fc6930133539.pdf.

"Markdown." n.d. Github.com. https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet.

"Pandoc Markdown." n.d. https://pandoc.org/MANUAL.html.

Project Nayuki. 2017. "Designing Better File Organization Around Tags Not Hierarchies." https://www.nayuki.io/page/designing-better-file-organization-around-tags-not-hierarchies.

Treasure, Julian. 2011. "5 Ways to Listen Better." July. https://www.ted.com/talks/julian_treasure_5_ways_to_listen_better/transcript?language=en.

Zhang, Charles. 2019. "Somewhere Design Document." Github.com. August 2019. https://github.com/chaojian-zhang/Somewhere.

———. 2020. "PDF Apprentice." Github.com. January 2020. https://github.com/chaojian-zhang/PDFApprentice.