Title:	Technical Agenda 2023	Created:	2022-11-20
Status:	Active Collection	Author:	Lion Kimbro

Intent

I want to always know what to program next.

Missions (M)

This is a collection of things I want to be able to do, and what they require.

Flags: (A)ctive, (B)ack Burner (C)omplete, D:1 Near – D:5 Far, (?)-In Doubt, (X)-Discarded

Flag	*Flag Name*	*Meaning*
A	Active	I am actively working on this.
B	Back Burner	I have worked on this, but it is on the back burner for now.
☑	Complete	This task is basically complete.
D:1	Immanent	This is a task I desire to work on immanently.
D:2	Soon	This is a task I desire to work on very soon.
D:3	Year	This is a task that's a candidate for working on this year.
D:4	Far	This is a task that I'd like to do soon, but I think it'll be more than a year before I've got it.
D:5	Very Far	This is a task that I am working up towards, but it will be a long time before I get there.
D:6	Fantastical	I may never get to this task.
?	In Doubt	I don't know if I will ever write this.
N#	Notes	There are further notes on this project below, see note# below
X	Discarded	I've given up on this. Or: I don't think it was a good idea to begin with. Or: It doesn't fit my plans any more.

ID	*Added Here*	*Mission*	*ref P#*	*Supported By*	*Flags*
M1	2022-11-20	Generic Object Format An generic basic objects of any type. Features a unique ID (a GUID, a TAG URI, and a serial number,) a programmer friendly name, a title, a hook, tags, a URL, a description, a space for notes, an image, an attached log including creation. All elements aside from a unique ID are optional.	P#0055	M2, M3, MQTalk/M4	D:4
M2	2022-11-20	Native Python Object General Logging Database A general database for collections of native Python objects. These could be created freely, and can be provided by requests.py. The primary log is kept in a single log file, and the entire database state can be recreated by playing the log forward. Database snapshots can be produced, and a subsequent log file followed to continue the development from the snapshot point. Reversion logs can also be kept, (or built,) optionally. Basic operations, just as if you were operating on the Python data in a native environment, are all supported and work.		MQTalk/M4, M19, M27	D:3
M3	2022-11-20	Location and Registration Service A system for locating paths. I want to only ever use a SINGLE environment variable – to point to the starting point for everything else. Then everything else is located based on that starting point. It's important that file resources can be created dynamically, and that the location of things can be shaped. So: Some form of jump tables, and safe editing of jump tables. Safe editing implies repairable database technology.		M19	D:3
M4	2022-11-20	Filetalk I've gone back and forth over component messaging systems, and what I have come to realize is that I really do think that files are *the best* mechanism for communication between components. Files are incredibly versatile. Every programming language can relate with them. File deletion is a manual process. File systems have a long history of permission systems for files. Files can be read by multiple consumers, or just a single consumer. They are easy to look at. They are durable. Every way I look at it, save one (or two), files are pretty optimal – the one way that they are not optimal, is that there's no immediacy: when creating a file, there's no notice to the intended target that: "Hey, there's a file here." (The second way that they are not optimal is beat only by shared memory: they must be exported, and they must be imported. But *only* shared memory lacks this limitation.) So my thought is to use filetalk, and as an auxiliary, run the simplest notification system I can imagine, which I call, "rattle." (M5)		M5, M24, M25	D:3
M5	2022-11-20	Rattle A simple system of inter-process update notification. The idea is extremely simple: to establish a mechanism for a process to be "rattled," and for a process to rattle another process. The means by which a process is rattled can be varied: OS Signals, polling a directory for a file, sockets, TCP/IP, anything. As long as it			D:3

		can be done without a human operator, as long as it works reliably, it can be "the rattle."			
M6	2022-11-20	Tk Framework Adapt the Tk code from pamphlet22, into a generally usable system, that includes custom paths for different needs, and include code that is particular for multiple different programs. The documentation is particularly important, and more complex than most of the documentation I have written to date – I estimate that there will be 2-3 pages long of API and global variable documentation, and that the document will remain open.		multiwork	D:1
M7	2022-11-20	Mini-Cubes (Tk) Recreate Mini-Cubes, but within the tcl/tk framework.		M6	D:4
M8	2022-11-20	Tagged Text A giant collection of tagged text. You can emit the tagged text to a text file, or even bundle it into collections exported as LSF. You can also absorb entries from text files or LSF files. You can search for entries, you can also look at the context of an entry by its import bundle, and also by date. Entries are stored with the date that the tagged text was entered into the archive, but it can also have a notion of when the tagged text was written. Support for classifying the data on the way in and on the way out.		M2, M3, M4, M6	D:4
M9	2022-11-20	File Archive A generic file archive. Files go in, files go out. There's a "receiving" folder, though you can of course also name folders. There's also an "export" folder. The files can be tagged. It's very much like "tagged text," and maybe I'll even want to combine the two projects, given that there's so much similarity between them. It all comes down to "store this" and "retrieve that" and "search for things matching: XYZ."		M2, M3, M4, M6	D:4
M10	2022-11-20	Graphical Python Editing Environment Arrange functions and global data items and imports in a Python module, visually. Icons represent functions, data, and imports. You can add arbitrary text, and lines. Labels appear on the right of icons, telling the name of the function, but it can be moved, and abbreviated, or be made mouse-over-visible only. You can preview and edit the code within the viewer, you can copy to the clipboard, you can import from the clipboard, you can also open up a separate window with just that code. I don't know how to do syntax highlighting, and I'm not so interested in figuring out how – somebody else would have to add that. Or, better: the system could interact with code that DOES do that, written by somebody else.		M2, M3, M4, M6	D:4
M11	2022-11-20	URLStore7/8 Adapt my raw link store to be queryable via new systems.		M2, M3, M4, M6	D:4
M12	2022-11-20	Contextual URL Bookmarking System A link store that stores bookmarks not just in terms of their solitary status (date recorded, tags, Title,) but also in one or more mapping contexts, which can be named bundles assembled in a time-frame, or deliberately assembled named bundles, or visual 2-D maps including labels.		M2, M3, M4, M6	D:4
M13	2022-11-20	Visual Internals Debugging Environment The capacity to see the internals of a program represented in a visually appealing and revealing way. Position watch variables visually, have bars representing the lengths of lists (or whatever,) – in various ways, see the system state. The person who wrote the program under examination has to create the different ways that the program state can be visualized. Then the program needs to be executed with reporting or querying access to the Visual Internals debugging system.		M4, M6, M18	D:4
M14	2022-11-20	Live Web Publishing System A system that makes it super-easy to publish online. Not just web pages, but entire databases worth of content. It needs to be relatively "live" – and I need to not have to think of a "file" at all. I just enter text into a space, say "commit," and it gets published to the web. It needs to be extremely low-tech. It needs to operate under the premise that the web site is hosted remotely. That said, since IPv6, that might not be such a requirement these days…		M4, M5, M15	D:4
M15	2022-11-20	Cross-Network File-talk Bridge Make it so my friends on Lion's Internet Office, and I, can all connect our programs, with ease, and use Filetalk+Rattle locally, and still communicate with our peers. There needs to be some kind of a secure cross-network bridge for filetalk. "Crazy," I know. Because File-talk garners the vast majority of its benefit from working specifically on a local system. So this starts to resemble more, perhaps, a network synchronization bridge: Different clients say, "I have this shared space," and then the protocol makes it possible for anybody on the network to share or not share the space. Is this just network file sharing? Probably both "Yes" and "No." "Yes," as in network file sharing likely does everything that we need, in a way. "No," as in network file sharing was not		M4, M5	D:4

		designed for these specific purposes, and will likely interrupt or interfere with our plans – especially the assumptions of homogeneity in operating system, and especially in the assumptions of corporate environment. I bet they are “too much,” and would be incredibly complex to install and properly configure. That won’t work – it’s got to be easy to configure, easy to run securely, and it’s got to work in a highly heterogeneous operating system environment.			
M16	2022-11-20	Data Type Documentation and Type Language Standardization There are both primitive types (such as a “string”) with specific conventional sub-types (“ISO-8601 YYYY-mm-dd / %Y-%m-%d”, or “name”, or “Title”) and also more complex data types, including message formats, protocols, and even app-internal data types.	projlist 0006		D:3
M17	2022-11-20	Python Dynamic Reloading Hosting Environment An environment in which Python code can be written, executed, debugged, and reloaded, at the same time. Python code should be able to be run purely anonymously – that is, without having a specific user-assigned name. Python code should be “bankable” – stored and retrieved by tag, without a unique name (but likely with a unique identifier – such as a GUID.) This makes possible an ecology of very small Python programs for performing all kinds of various functions, but that can be supported with a user interface, invoked by tag and other pluggable systems, and easy to work with and change.			D:5
M18	2022-11-20	Canvas-based Drawing Environment Inspired by my favorite “Programmer’s Drawing Environment” – tkpaint 1.6. The idea is to have a paint program based specifically around the tkinter Canvas, so that layouts can be created that perfectly map to Canvas layouts. That way, complex 2-dimensional graphic representations can be easily placed directly into software projects, and edited and adjusted in real-time within programs. That is, if you were developing a user interface in a program, the user interface could be exported from the live running program into the Canvas-based Drawing Environment, edited within the drawing environment, and then exported back to the live running program, all without stopping or stopping either of these programs. You could then “sculpt” the user interface, without interruption. This program synergizes with M17, the Python Dynamic Reloading Hosting Environment. The overall idea is to maximally make it so that programs can be sculpted in real-time, without having to go through an edit, load, execute, crash, examine cycle, over and over again. If the system can run and the problematic bits fixed mid-execution, it should dramatically speed up development time. If the user interface can be tweaked in real-time without having to go through a cycle, it should dramatically speed up the formation of useful GUI designs.		M2, M3, M4, M6, M17; see also M31	D:4
M19	2022-11-20	JSON Lines Transactor A system of writing to a log, line by line, including a completion marker. If the completion marker is not found, the line is ignored. The system may be as simple as: “Each line starts with a space, and then the content.” “When the line is finished, a line containing only a star (*) is added.” Or, for the ease of reading: “When the line is finished, the space at the front of the line is replaced with a star.” Or perhaps a separate file contains a single character, and that character says “W” for “writing,” or “C” for “complete.” When a line is being written, “W” is written there and flushed to completion. Then the line is written. Then the “W” is replaced with a “C”. For performing a multiline transaction, it could just be that the last known valid file position is recorded, and that value is updated upon completion. Care must be taken to A/B it, because the process of recording the file position is interruptable. Note that: This MAY be as simple as – collecting lines, and not using them, until you get a “COMMIT” line. Take advantage that “COMMIT” or “*” are not valid JSON string, and will never be encoded. You can simply queue lines until you get to the “COMMIT.”			D:3?
M20	2022-11-20	Stopgap 2023 A universal interface to stopgap services. As those stop-gaps are filled in, the stopgap module replaces the call with the call to the actual service. And when I am revisiting a program, I can pull out the stopgap call, and replace it with the call to the actual service as well. Until I no longer have a need for the stopgap module. But anything that I would stopgap, I work through a stopgap module function call. Stopgap codes does *not* need to be generic. For example, MQTalk is fairly particular, and there will be elements that can’t be “faked” in the transition to the next iteration of Filetalk+Rattle (M4+M5). Just do what you can – it’s as much about making the code work in a transition zone, as it is noting and calling out those transition zones, for future replacement.			D:2
M21	2022-11-21	Pamphlet22 Improvements 1: Filetalk+Rattle & Tk Multiwork + Stability			D:4

		Specifically, making it so that Pamphlet22 files and pages can be opened via FileTalk+Rattle (so: REQUIRES M4+M5), stabilizing Pamphlet22 for basic editing, integrating Pamphlet22 work with the Tk system via multiwork tool, writing manual tests for key functional areas,			
M22	2022-11-21	Pamphlet22 Improvements 2: Network Publication That is, making Pamphlet22 publish to the Internet.		M14	D:4
M23	2022-11-21	Lion's Internet Office Communications System The capacity to communicate with other's in Lion's Internet Office, via a message system that leverages FileTalk and a Network Bridge. May or may not be visible from the web, using M14.		M4, M5, M6, M14, M15	D:4
M24	2022-11-21	Implementation of JSON-RPC for FileTalk JSON-RPC is a relatively mature technique of implementing RPC via JSON. There are numerous implementations existing for Python already, if my memory serves me right. It includes exception capture and delivery. Utilizing this method could speed along my development, and also make my work more compatible with that of strangers. There is a potential cost in mismatch with the ways in which I intend to communicate between processes in FileTalk.		supports M4	D:3?
M25	2022-11-21	Conceptual FileTalk Model Write a document that outlines the type of communication envisioned for FileTalk. Communication from process to process, bus communication, status panels, and the general ecology of components around the FileTalk system. Issues that may arise (partial writes to files, while mid-read.) Fault tolerance requirements.		supports M4	D:4 ?
M26	2022-11-22	Project Manager Create a project manager, -- which would, effectively, render this document obsolete. Projects would be tracked by a variety of criteria, including maturity along different dimensions of execution. Project log notes, next steps, links to other projects, all would be visible. The capacity to view different plans for development. Projects may make use of multiple repositories.	WISH2	M1, M2, M3, M4, M5, M6, M14, M18; see M39	D:5 N1
M27	2022-11-24	Boxed/Dict/List Manipulation Python code for encoding manipulations to data that is boxed, or in a dict, or in a list. I believe I already have a generic box system in my repository, so check that for reusability. The immediate goal is to support M2 (General JSON Logged Object Manipulation Database) – that's why lists and dicts are included with the boxing protocol.	P#007D		☑
M28	2022-12-01	History Browser A tool for browsing human history on planet Earth. Features a globe, a collection of maps, timelines, and a logical tree browser. The user can load histories into the program's execution space, and trace and browse history in different ways.			D:6
M29	2022-12-02	Universal Console A console that can talk to any of the programs. You target a program, and then send console messages to them. They can send back text per Kartik's system – that is, it can include images and diagrams as well.			D:3
M30	2022-12-02	Canvas Renderer for Kartik's Image Format Using a Tk Canvas widget, represent Kartik's image format.			D:3
M31	2022-12-02	Canvas Editor Kartik's Image Format Using a Tk Canvas widget, edit graphics and keep track of them in Kartik's image format.		M30; see also M18	D:4
M32	2022-12-02	Integrated Text + Editable Image, per Kartik's Format Make an integrated system within M6 (or another Tkinter ecology) for working on Kartik's text+integrated image format.		M6	D:4
M33	2022-12-02	Interactive Sculpture Environment An environment where you can easily make crafty models of things behave in all kinds of interesting interactive ways. It would feel a bit like a computer game. You could draw an object, copy it, and make it do something when you click on it – open up it's interior, or something. You could have tree lists that drop down, and are programmatically populated when you expose them. There'd be an integrated graphical editor, and you can name and attach data and semantic meanings to the drawn objects. A use case would be writing documentation – you'd represent the various elements of the software environment as visual depictions, and then document the API in little "cubbies" in that visual depiction. You could mouse over parts, and it would expose the different cubbies, and you'd see the API documentation inside. As preparation for this project, document programs you're writing in a crafty way with cardboard paper, much like you'd do with the actual environment.			D:5
M34	2022-12-06	Frederick Douglass Lifemap An interactive diagram of Frederick Douglass's life – where he was when, what events happened, major life decisions and changes, what he wrote, what he said. The goal is not to be complete, at this point (that would be a work that could span		M35	D:6

		decades,) but rather, to understand the man's beliefs, how they formed, what he said when and why, and how they distilled and contributed to the shaping of the United States of America. I envision that I would want this to appear on the web. However, I really hate programming for the web. So what kinds of technologies would I want, in order to create it for the web? And I think that I'd want a programming language that could be compiled for consumption by a JavaScript interpreter. That way, I could just write a short interpreter in JavaScript, hook it up to the key capacities of a Canvas, and therefore deeply limit my time in the JavaScript ecology. Besides, I've been wanting my own virtual machine and programming language, regardless. My ideas about programming languages and programming and debugging are highly unusual anyways.			
M35	2022-12-06	System-Focal Programming Language & Virtual Machine This programming language is system-focal. Key ideas that I would want in this programming language include the following: • Expressed by line, symbol, and text in a generic environment that presents a medium beyond text. That environment itself should be communicative, so that other programs can command and receive signals from the environment. • Primitive programming language. It does not focus on efficiency, and it does not have complex features. It is tailored to the specific modes of expression that I make use of. • Designed for visualization, live debugging, *health monitoring*, and embedded semantic knowledge of what's what, what's where, and why. • I could hardly care less about types and static type analysis. • Compiles to a byte-code that it is easy to write an interpreter for in any language and for just about any environment.			D:6
M36	2022-12-06	Tk Framework Filetalk Plug-In Make it so that you can communicate with the Tk Framework via the Filetalk plugin.			D:3
M37	2022-12-06	General Talk Adaptor I often can't seem to make up my mind on the communication technology – FileTalk, MQTalk, COMTalk, -- which shall it be? But one thing remains constant, always, in my process – the use of JSON data blocks for communicating. So I think I should work out how to publish for a standard interface, and then just plug in whatever system I'm communicating via, at the last moment. This might not be "perfect" – there are some systems that are dependent on the communications platform. For example, filetalk has a concept of files and file location. MQTalk has the idea of named channels. And so on. However, I should be able to program against an abstraction of what it is desired to offer, and what it is desired to communicate with, and then configure how the particular system is interacted with, within the isolation of the General Talk Adaptor's caller. M37a – an attachment for M6 (Tk Framework), so that other programs can interact with something that uses the Tk Framework. Perhaps this would include notions of sending messages to individual windows or individual objects within the program, or subscribing for notifications upon certain events.			D:2
M38	2022-12-07	Resources.py I've already written this; I'm just including it here because I reference it so frequently. Resources.py is a general allocator for basic resources needed by Python programs. One day it should handle allocating for TCP/IP services or FileTalk services, but presently it focuses on permanent durable file resources, that are automatically created, loaded into, and saved out from a Python context. When the data appears in Python, it can appear as raw bytes, as a Unicode string (UTF-8 on disk,) or as Python objects that represent JSON data.	P#0071		☑
M39	2022-12-07	Trigger2023/Repo2023 New Trigger/Repo system – perhaps with a different or renewed division of labor between the two functionalities (Trigger: Workflow User Interface, Repo: Repository Directory Management.) This would involve a few major changes: 1. Use a Tk GUI interface for browsing and operating, kicked off by the kicker. 2. Support 5 development bays, rather than 3. (Expand my other tools to make use of 5, as well. That is: EMACS macros, and desktop .bat files.) 3. Keep a single running Repo agent (Repo2023). 4. Perform operation communications via FileTalk. *This means that other programs can trivially query and interact with the project & repository system.*		see M26	D:3

		5. Support for isolating .doc editing to a specific temporary directory (due to my inability to control Microsoft Office's folder locking.) 6. Isolation of Projects from Repositories. A Repository can be used by multiple Projects, and a Project can make use of multiple Repositories. It's not necessarily 1:1. The mappings between projects and repositories should be navigable. 7. Projects should be able to be linked with other projects. Links are all bi-directional. It should be possible to navigate between these relationships within the tool. ⚠ It may be critically important for my development integrity to keep or otherwise maintain Trigger2/Repo2 as well, so that should I ever need to develop on a platform that does not have support for filetalk, I can still safely access the repositories and continue development. The simplest way to do this would be to continue to utilize Repo 2 for essential repository interactions. Fortunately, Repo 2 uses its own miniature filetalk-like system, and is extremely easy to make use of in this way. So that should not be too difficult. I can still have Repo2023, but it could still utilize Repo2 for core repository access manipulations.			
M40	2022-12-07	Kicker The "kicker" is a program that starts other programs. It's comparable to the Windows window-key R "Run" dialog box. Some functional qualities of the kicker – it's designed to start, get a command from the user, issue the command through FileTalk, and then die. If it's left hanging for too long, it should just disappear – there's no need to keep these things hanging around. They should not have any durable state, and it should be perfectly safe to have 20 of these running at the same time. I have already started development on this, but it requires attention and love and it needs to actually be made to work. The protocol matters. The filetalk system matters, too. Critically, it relies on FileTalk reaching at least a certain degree of maturity. It requires that the TK Framework is in place, too.	P#007A	M4 (filetalk)	D:3
M41	2022-12-07	Taxonomist A tool for developing taxonomies. "Terms" form the fundamental unit. A term is often associated with a single "word." A term can be positioned within multiple different hierarchies as part of its participation within one unified, integrated, comprehensive system. Terms can be tagged. Tags can be identified as mnemonic tags, or as bona fide tags. Two different tags can share the exact same text – homomorphic and ambiguous, but acceptable regardless, because tagging is about convenient location, and not absolute data demarcation, unless you want to specify that the system should not accept homomorphic tags, or if a specific tag is reserved as unique and occupying the entire space of its own name. (In which case, a notice will indicate that a forbidden tag name collision has occurred.) Back to terms – Terms can be described, detailed. (So can tags.) All data from the taxonomy must be freely and easily accessed by other programs. Information should be arranged into a JSON (or Jump JSON) document.		M1 (generic objects db), M4 (filetalk), M6 (Tk framework)	D:4
M42	2022-12-07	Vision of Computing Document A document that describes my vision of computing and note-keeping. It is both architectural & about programming, and about vision & information ecology. It describes features of the agent-based systems I believe we need, the limitations of our present popular programming languages (and focusing on Python) in terms of data access, graphical user interface, inter-process communication, network synchronization, virtual machines and DSLs, and the general lack of a culture around user-controlled programs that talk with other same-user-controlled programs. In terms of larger vision, it paints a picture of social information construction, a culture of social information construction, and describes the things that need to be easy to program into computers, using accessible languages (like Python or Lua,) in order to explore those spaces easily. It also describes specific features of philosophies that, if adopted by groups of people, I believe will lead them to create information socially. This is a vantage point for looking at the world and our activities within it, that when adopted, would naturally lead people to socially construct information and share it with others on the Internet in a way that connects and adds up – much like the philosophical realizations that lived behind the "Encyclopédistes" of old. Attitudes towards history, science, knowledge, desire, conscience, and existence are part of this – a kind of "social existentialism." This paper is very much in the tradition of HCI in the old-school sense – not the modern day focus on creating "good UI." Thanks to Luke Stanley for pointing out that I need such a document.			D:4

Work Queues (W)

Work#	*Project*	*Added Here*	*Task*
W1	multiwork (P007C)	2022-11-20	add "minimizing" support – clearing the files for a project, after verifying that the code's been exported
W2	resources.py	2022-12-07	secure all writes – write first to a "b" version, and then write it onto the "a" version, and know how to recover if "a" turns out to be unwritable because the disk write was interrupted; delete the "b" version after the "a" version is written, or after the "a" version is read successfully – do not worry about the lack of this functionality until I feel its time to implement it – and trust that it will work for everything, at that point; note: also make it so that there is a secure, reserved location, that these "b" files will be written into, safe from user meddling Additional Note: I may need to create a multi-file transactor to make this work right... Well, just make sure that ALL of the "b" files are in the "b" position, then create a "B COMPLETE" state somehow, and start copying over, and make sure that if THAT is interrupted, that it is completed before the B's are deleted.

Log (L)

- If something already has a log number on it, like M or I table entries, there's no need to log it here. This is for logging things of overall character, or that otherwise are not date logged.

Date	*Log*
2022-11-20	created tables M, I, W, L
2022-11-21	+plan (P), -ideas (I) [just use M for ideas]
2022-11-22	located and imported notes on project planning (N1); further developed plan
2022-12-06	added Critical Tech Trees (TT), Proximity Radar (PR), and Document Wishlist (WISH)

Document Wishlist (WISH)

These are wish-list items for this very document

WISH#	*Added Here*	*Wish*
WISH1	2022-12-07	bi-directional links of different types – "Supports," "Supported By," "Alternative To," ...
WISH2	2022-12-07	see M26 Project Manager
WISH3	2022-12-07	use codes to name certain projects more than just an "M#" – there are certain projects that I refer to quite extensively, such as FileTalk and the Tk Framework – those should be abbreviated as M#FT and M#TK, or something like that, so it's easier to read and type them, until such a point as I have the M26 Project Manager up and running with some kind of support to decode M#'s while browsing
WISH4	2022-12-07	Document Numbers (LDOC#..., where "LDOC" stands for "Lion's Documents"

Planning Log (P)

PLAN2022-12-06/2022-12-07:

My attention is centering on M6, the Tk Framework. I have spent the last few days documenting it on paper. I have been debating a great deal on how to architect the system. The two major decisions I am looking at are:

1. Shall I make it one single Python module? Or should I keep it as a collection of interacting modules?
    a. Advantages of a single Python module: Simpler, and self-contained.
    b. Advantages of multiple Python modules: Recomposable, supporting more variation and exploration.
2. Shall I make it editable in place, as is presently the plan? Or shall I treat it as more rigorously solid, and have clients customize its use?

I seem to be stalling here, somehow... Something about the hugeness of the task. I've presently inventoried about x6 files from pamphlet22, and – it just seems to be ballooning out, indefinitely. In fact, it might be something more than just a Tk framework – I have a general need for a framework for how my programs come together. File resources loading and saving, menu configuration (pure-textual execution,) Tk GUI system and configuration (Tk UI,) filetalk interaction, interactive looping, -- it's all the same problem. Perhaps the issue is that I've just been looking at this the wrong way. Perhaps what I need to do is compose a document that describes my general strategy and mechanisms for solving these problems, and then work on conceptually integrating my approach. So everything is that one thing.

That means that I have a certain problem, though: My programs are no longer single module executables. Sharing what I work on with people means that they need to make space for the entire system I have worked out. At a minimum, they need to configure the stop-gap, and set that right. The programs that make use of data, will need the general data directory to be pointed out, for instance. I'd be distributing a multi-component system to people who have no idea why it's a component system, or what that means, or what it benefits them. Because essentially, they not knowing what it is or why it is, will have no use for it, and therefore it will be of no benefit to them, at the cost of great complexity. So, that's a problem.

Programming for an agent-based environment with multiple components talking to one another via a protocol, and programming for a single process on a single computer, is just a completely different beast.

I guess to a degree, none of this can be helped. I'm working towards a radically different paradigm of programming, with radically different tools and modes of basic expression, and there's no way that it can fit into the box of "single text file, single process, single program." It's just too different.

So I think I have to inwardly just commit to "Yeah, it's going to be a mutant," and embrace that, and go forward with that. People are going to say "I don't know how to install it," and people are going to say "I don't get why it's doing all this extra stuff, when I just want to..," and I'm just going to have to say "I recognize that, but this really is something different than you were expecting; It's pattern of use is something radically different..."

OK, so I think my plan going forward here is:

- Create project "tkframe". Start clean slate. Put pamphlet22 in bay 2, and copy files and portions of files into tkframe.
- Adapt tkframe to make use of resources. On the way here, make sure that resources works as expected.

But UGH, I hate, hate, hate all of this writing. It's seriously boring me. I just want to write code.

PLAN2022-12-02:

Today I'm in the mood to do something graphical.

- M6 – Tk Framework
- Do crafty stuff (see M33) while documenting the pamphlet22 system, and my modifications to it.
- Keep the crafts stuff in my physical paper archive.

PLAN2022-11-21:

Stop-gapping (via M20:stopgap2023):

- M3 – resource location
- M4/M5 – next iteration of filetalk w/ rattle support – using MQtalk for now

Middle View:

1. Phase 1. Get M2 (Generic Native Python Logging Database) working.
    a. ☑ Write out my concept of how I would implement M2.
    b. ☑ Second, review entries(004S), for suitability. (COMPLETE) Entries is not suitable. It's focused on COMTalk, uses a bag system (was meant to run for ALL databases at a time,) is paged, and focused on specifically formatted notes. This project, I divide and conquer it in very different ways: One process = one database, all-in-memory, journal+snapshot based, and with generic JSON. There is very little code overlap.
    c. Segregate M19 (Transactor).
    d. Write M2 (Generic Native Python Logging Database)
        i. work out communications (stop-gapped M4/M5, using MQTalk for now)
        ii. may require M24 (JSON-RPC integration)
        iii. uses M3 location services – which are stop-gapped
        iv. ref M20: stop-gapping
2. Phase 2. M8 (Tagged Text)
    a. Will build on Phase 1's M2 work, directly.
    b. Will require M6 (Tk System) to be isolated and developed.
    c. Will require the stop-gapped M4/M5 system using MQTalk for now. Possibly M24.

Now:

- Scatter Level 1:
    - Look at what I want to do. Start working on it.
        - Do I want to make JSON transactor something explicit and separate?
            - Yes, ..? And then multiwork it in.
    - Create Transactor project.
    - Create jsondb project.
    - ~~Review entriesdb.~~
        - ~~Is it suitable, as it is?~~
        - ~~How did it work?~~
        - ~~Does it transact?~~
- Scatter Level 2:
    - M20 – stop-gap (which should be developed through multi-work)
    - multi-work – which will need the minimization functionality

Notes (N)

Note N1

Note#	N1
Title:	Organizing Projects
Originally Written:	2021-12-20, log.txt, Project 0006 (projlist) “Documenting & Developing My Data Formats”
tags:	project organize organizing organizer

- pre-2021-12-20, research.txt: began outlining this, at some point
- 2021-12-20, transferred to projlist project 0006 – “Documenting & Developing My Data Formats”, log.txt
- 2021-12-23, created projlist
- 2022-03-22, resumed work on projlist
- 2022-05-03, revisited, further work (and now current)
- 2022-11-22, transformed and transferred found notes from “Documenting & Developing My Data Formats” (projlist project 0006) to “Technical Agenda 2023” (this document; projlist project 0033)

This note should probably be relocated to projlist. Leave an out-card when you do.

For each data format, track the following:

Project Complexity	1 (Very Simple) – 5 (Extremely Complex)
Design Status	0: Almost Nothing 1: Initial Sketch 2: Accumulating Body 3: Sufficient 4: Mostly comprehensive 5: Complete
Product Status	0: Almost Nothing 1: Initial Sketch 2: Accumulating Body 3: Functional, Not Ready 4: Useful 5: Complete
Released to the Public?	Yes (tag: released)
Private/Public?	(tag: private)

GOALS:

- be able to extremely rapidly create projects (project information + folder) *(FULFILLED)*
- be able to extremely rapidly locate projects *(FULFILLED)*

SOMEDAY:

- be able to structure the file management of a project
  - single-folder (which will be used presently) *(FULFILLED)*
  - time-versioned folders (like code repository)
  - (other?)

Note N2

Note#	N2
Title:	Implementing a Native Python Objects General Logging Database
Originally Written:	2022-11-23

How I would implement a Native Python Object General Logging Database, now, today?

Do I make it truly about Native Python Objects, or JSON objects? (There ARE some differences between the two.)

Write up how I would envision using the tool.

function	*flags*	*by direct call*	*by protocol & server*	*by stub call (to protocol)*
naming/selecting db				
setting up db				
writing to db (in a transaction)				
reading from db				
recovering db				
playing logs forward to a time point (can defer)				
playing logs backward to a time point (can defer)				
playing logs forward totally				
constructing revision logs (can defer)				
retrieving history information				

Documents to write:

tabulate write operations	written below	
tabulate read operations	written below	
JSON <-> Python native types mapping		
API by direct call		

API by protocol & server		
API by stub call (which relays to protocol)		

File assets:

- primary log file (live.log.json)
- archival log files (YYYY-mm-dd.log.json)
- revision log files (YYYY-mm-dd.log.json)
- snapshot files (YYYY-mm-dd.snap.json)

Naming system for files:

- "live" – immediately in-use log, since either (A) the last snapshot point, or (B) the beginning
- "YYYY-mm-dd" – logs up to, and snapshot on, this date, since either (A) the last snapshot point, or (B) the beginning

Technologies to use in construction:

- Transactor

Tabulating Write Operations:

- start transaction
- end transaction
- set([chain…], value)
- insert([chain…], value)
- append([chain to list…], value)
- extend([chain to list…], value)
- rm([chain…]) -- removes an item at an index or a key or the root
- read([chain…])

I need to be able to locate an item – by index or by key.

Lists: Append an item, insert an item (before an index). Delete an entry at an index. Both positive, and negative, indexing. Nav into an index.

Dictionaries: Set/Reset a key to a value. Delete a key. Nav into a key.

Anything: Replace with a different value.

Start at the root. Optionally index to a location. This can include an append location (on a list). Set the value, delete the value, read the value.

Assumptions:

- Append to a Journal file.
- Snapshots allowed, usable.
- Plays from last snapshot, plus current journal, to hold everything.
- All data is stored in RAM. In the future, I might write one that preforms paging in and out of disk, but that is not the present objective.

Pseudo-code

- global (g) – general globals about the data collection
- global (h) – h[DATA] is the large data collection
- global (transaction)
- fn: load(p) – p is a folder containing the various files involved
  - fn: load last snapshot (or blank)
  - fn: load matching log (or 1st log)
    - reads via Transactor
- fn: apply transaction
- fn: snapshot db
- operations – creates transaction entries
  - start transaction
  - end transaction
  - set
  - insert
  - append
  - extend
  - rm
  - read
- interface to MQTalk

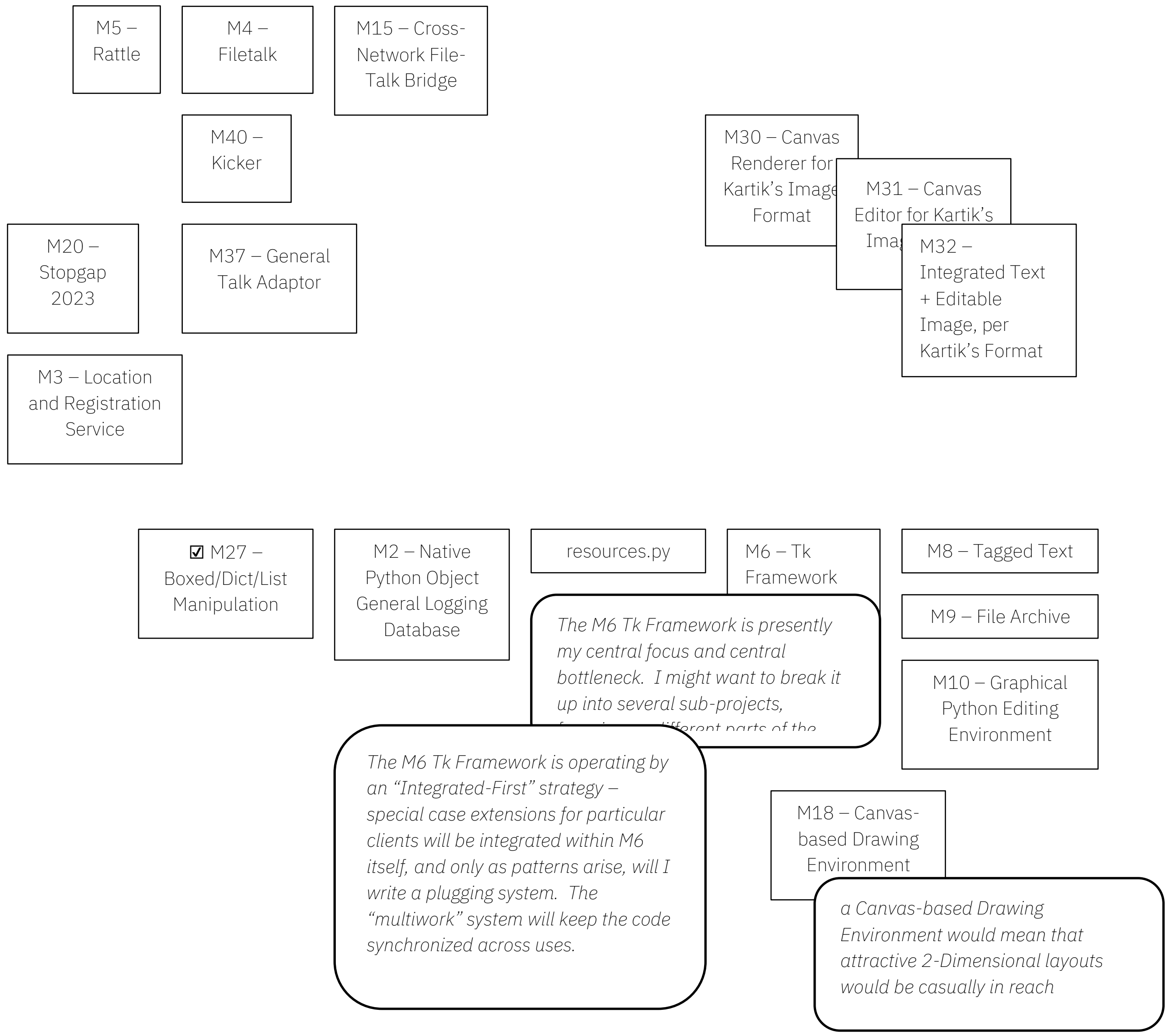
    - use stopgap ..?
- interface to resources.py ..?

Other Possible Approaches:

- Divide And Conquer #1: Boxed/Dict/List Manipulation. M27. COMPLETED. Make a project specifically based around a language for dictionary/list/boxed manipulation. Get the essential interactions and mode of expressing such interactions encoded in there, tests, completeness, and so on, and then utilize that system in this system.
- Divide And Conquer #2: Transactor. This can include navigation forward, backward, to a specific time, etc.,. The actual model and data that changes the model can be independent.

Critical Tech Trees (TT)

Note: Not only is this highly incomplete, it is even incomplete within its own limited subset of coverage. Which is just a fancy way of saying, “Under Construction.”


M5 – Rattle
M4 – Filetalk
M15 – Cross-Network File-Talk Bridge
M40 – Kicker
M20 – Stopgap 2023
M37 – General Talk Adaptor
M3 – Location and Registration Service
M30 – Canvas Renderer for Kartik’s Image Format
M31 – Canvas Editor for Kartik’s Imag
M32 – Integrated Text + Editable Image, per Kartik’s Format
☑ M27 – Boxed/Dict/List Manipulation
M2 – Native Python Object General Logging Database
resources.py
M6 – Tk Framework
M8 – Tagged Text
M9 – File Archive
M10 – Graphical Python Editing Environment
M18 – Canvas-based Drawing Environment

The M6 Tk Framework is presently my central focus and central bottleneck. I might want to break it up into several sub-projects,

The M6 Tk Framework is operating by an “Integrated-First” strategy – special case extensions for particular clients will be integrated within M6 itself, and only as patterns arise, will I write a plugging system. The “multiwork” system will keep the code synchronized across uses.

a Canvas-based Drawing Environment would mean that attractive 2-Dimensional layouts would be casually in reach

Proximity Radar (PR)

Current to: 2022-12-07

D:1	D:2	D:3
M6 – Tk Framework	M20 – Stopgap 2023	M2 – Native Python Object General Logging Database
unassigned – General Programming Framework Outline (see PLAN2022-12-06/2022-12-07)	M37 – General Talk Adaptor	M3 – Location and Registration Service
	W1 – “minimize” function for multiwork	M4 – FileTalk
		M5 – Rattle
		M16 – Data Type Documentation and Type Language Standardization
		M19 – JSON Lines Transactor
		M24 – JSON-RPC for FileTalk
		M29 – Universal Console
		M30 – Canvas Renderer for Kartik’s Image Format
		M36 – Tk Framework FileTalk Plug-In
		W2 – secure resources.py writes

CONCURRENT DEVELOPMENT:

- M6
- General Programming Framework 2023 – started in F:\bigfiles\onedrive2022\OneDrive\origins\2021\data\lists\PROJ\0036