

Luther's 95 Theses: The GEDCOM Deconstruction

Everything You Never Wanted to Know about GEDCOM But Were Afraid to Just Ignore

by Luther Rangely,
Uncle Buddy, and Professor U. d'Guru

as well as the entire staff of Treebard University whose motto is:

Regularity Rules!

```
0 @I1@ INDI
1 NAME Professor U. /d'Guru/
1 FAMS @F1@
1 FAMC @F4@
0 @I2@ INDI
1 NAME Mrs. /Widow/
2 DESC pretty as could be
1 FAMS @F3@
1 FAMS @F1@
0 @I3@ INDI
1 NAME Widow's /Daughter/
1 DESC red hair
1 MARR
2 WIFE
3 AGE grown up
1 FAMS @F2@
1 FAMC @F3@
0 @I4@ INDI
1 NAME Professor U. /d'Guru/
2 NSFX Sr.
1 FAMS @F4@
0 @I5@ INDI
1 NAME Baby Boy /d'Guru/
1 DESC bouncing
1 FAMC @F1@
0 @I6@ INDI
1 NAME Jed C. /d'Guru/
1 FAMC @F2@
0 @F1@ FAM
1 HUSB @I1@
1 WIFE @I2@
1 CHIL @I5@
1 MARR Y
2 HUSB
3 AGE 23
0 @F2@ FAM
1 MARR Y
1 HUSB @I4@
1 WIFE @I3@
1 CHIL @I6@
0 @F3@ FAM
1 WIFE @I2@
1 CHIL @I3@
0 @F4@ FAM
1 HUSB @I4@
1 CHIL @I1@
```

I'm My Own Grandpa by Moe Jaffe, Dwight Latham 1947
(Reduced to a GEDCOM file by the staff of Treebard University)

I'm My Own Grandpa
by Moe Jaffe and Dwight Latham (1947)
as sung by Ray Stevens

Many many years ago
when I was twenty three,
I was married to a widow
who was pretty as can be.
This widow had a grown-up daughter
who had hair of red.
My father fell in love with her
and soon they too were wed.

This made my dad my son-in-law
and really changed my life.
Now my daughter was my mother
'cause she was my father's wife,
and to complicate the matter,
even though it brought me joy,
I soon became the father
of a bouncing baby boy.

My little baby then became
a brother-in-law to dad,
and so became my uncle,
though it made me very sad.
For if he were my uncle,
that also made him brother
of the widow's grown-up daughter
who was, of course, my step-mother.

My father's wife then had a son
who kept them on the run,
and he became my grandchild
for he was my daughter's son.
My wife is now my mother's mother,
and it makes me blue,
because although she is my wife,
she's my grandmother too.

Now if my wife is my grandmother
then I am her grandchild.
And every time I think of it,
it nearly drives me wild,
'cause now I have become
the strangest case you ever saw.
As husband of my grandmother,
I am my own grandpa.

Oh, I'm my own grandpa.
I'm my own grandpa.
It sounds funny I know,
but it really is so.
I'm my own grandpa.

Luther's 95 Theses: The GEDCOM Deconstruction

Everything You Never Wanted to Know about GEDCOM
But Were Afraid to Just Ignore

by Luther Rangely writing as Uncle Buddy writing as
Professor U. d'Guru writing as Scott Robertson

as well as the entire staff of Treebard University
whose motto is:

Regularity Rules!



TABLE OF CONTENTS

FRONT END

[Prologue](#)

[About the Treebard Project](#)

[Introduction](#)

THE 95 THESES

[GEDCOM is Easy to Understand--Unless You're a Computer](#)

[Writing a GEDCOM Export Program is Easy](#)

[Writing a GEDCOM Import Program is a Real Education](#)

[SQL: Top of the Class Since GEDCOM was in Diapers](#)

[Cardinality Means Never Having to Say "Ooops"](#)

[Primary and Arbitrary are not the Same Thing](#)

[Transcending GEDCOM's Pecking-Order Approach](#)

[One Goal, One Method](#)

[There's Only One Version of GEDCOM](#)

[When GEDCOM Actually Gets Replaced](#)

[My GUI Features: Not GEDCOM's Problem](#)

[What Users Don't Know... Won't Hurt Them](#)

[One Task per Tag](#)

[Elucidom Means *the State of Making Clear*](#)

[The Only Good GEDCOM Parser is a Custom GEDCOM Parser](#)

[Custom Tags: For Workaholics Only](#)

[For Wanna-Be Authors of GEDCOM Import Programs](#)

[Tags You Should Use Wrong](#)

[Don't Group Tags in Categories--Usually](#)

[GEDCOM is Not About Relationships](#)

[Families that Stay Together... are Weird](#)

[Events & Attributes: Not One Thing; One Category](#)

A Name is a Name

AGE is Simple, Let's Keep it That Way

There's Only One SUBM Record

REFN, AFN, RIN: The Prior Database's IDs are Obsolete

Much Ado about One Thing

A Place is Really Three Things

GEDCOM Can't Handle Adjunct Roles

Keys and Values are not Reversible

UNIGEDS Needs a Transcription Element

Assertions, Sources and Citations

Source Types Pre-exist Media Types

Reserved for Future Comments About GEDCOM

Reserved for Comments About UNIGEDS, gedMOM, or Treebard

BACK END

Epilogue

PROLOGUE

November 19, 2023

I began this particular life-and-death struggle with GEDCOM on August 12, 2023. Today it's more than three months later, and this umpteenth attempt to create a finishable GEDCOM import program has finally succeeded the same way other GEDCOM importers have succeeded: by curbing my appetite for perfection.

More can always be done. More custom tags could be handled by deciding what to say about them in the exceptions report, more exceptions could be handled, more broad-stroke-gashes could be turned into the kinds of compromises that make it look like my foray into GEDCOM Territory succeeded, but what really impresses me about this, my first-ever successful breach of the Great Wall of GEDCOM, is that this far into the main structure of the code (more or less all the way), the code is still, easy to work on, easy to understand, and well-organized.

I did not get to the point, this time, where I was slapping band-aids on the code in the vain hope that it would get done quicker that way. This time, short of the usual troubleshooting, I *am* done (see "curbed goals" above), and I never even got to the part where desperate hacks came to obscure and subvert the intention of keeping the code simple and accessible to other amateur programmers who want to work on genealogy software.

In case you want to know, here was the final secret that allowed this version to get to where it got without getting all twisted up and hard to work on.

Early in the procedure, the GEDCOM file was translated line-by-line, replacing all ambiguous tags with non-ambiguous tags. This was the clean start that was needed, without which the project had been doomed in every prior attempt.

My happy dream of coding is a project that gets easier as you work on it, because it was planned so well. But I don't

plan much when I don't know what to do. So instead, I started over maybe a few dozen times, gradually gathering intuition about the task ahead of me... again. Finally, I tried something that a little voice had been urging me to try ever since I'd first started trying to import GEDCOM. I translated the tags.

With every tag assigned a single unambiguous meaning, it also helped a lot to have most tags trigger a small, dedicated function, thus assigning each tag a clear, straightforward task.

The other thing that made it possible for me to create a finishable GEDCOM import program without having it activate my internal spaghetti factory was surrendering myself to the need--even if it's just my personal need--to have Python read through the lines several times, with limited goals each time. This amounted to translating the GEDCOM in discrete stages till the final version was ready to go straight into the database.

Part of the goal was to lighten the load of the conditional if/else testing. By isolating much of the conditional testing to the earlier stages, the way was clear to just connect obvious dots in subsequent stages. It took three months of full-time work to convince myself that my import program was finishable (*this* time. It wasn't the first time I'd tried working full time on GEDCOM).

There's plenty more for devoted GEDCOM champions to do in order to handle edge cases such as tags that no one uses since they're so very close to being useless, and getting data presented by the weirdest of GEDCOM's constructs prettied up enough to show in an exceptions report that's actually readable. Currently I've backwheeled to a policy of ignoring all custom tags, for reasons mentioned herein as well as this reason: custom tags are used so much for storing vendor cruft--which is not their purpose--that each vendor has created a master's course in navigating their personal flavor of GEDCOM abuse.

Let's just blame GEDCOM for that, and not take my colorful language personally, OK?

Writing the GEDCOM export program won't take longer than two weeks. The only hard part about it will be dumbing down UNIGEDS' data to GEDCOM's lower-common-denominator level. But I'm glad I did the import program first, because now when I write the export program, I'll know which tags and available GEDCOM structures to avoid.

ABOUT THE Treebard PROJECT

Occasional stabs in the dark on this project began in 2015. I went full-time on it in July 2018 and have continued full-time on it since then, with a few short vacations. By "full-time" I mean 60+ hours a week, usually 7 days a week.

Everything I've done in the way of creating genealogy software tools is free, public domain, portable, open-source, and unlicensed. This doesn't make me some kind of saint or ascetic. I accept donations but I'm so inept at marketing myself that I'm better off not doing so.

Treebard GPS is a showcase of primary genealogy database software functionalities, the so-called "front-end" or "graphical user interface" (GUI) that uses UNIGEDS to store its data. Treebard is not meant to be used for daily work, but rather to serve as a model for future development by anyone who wants to borrow from it. Borrowing from any of my work in the field of genealogy software comes with zero strings attached. But the name "Treebard GPS" is mine, all mine.

UNIGEDS stands for "Universal Genealogy Data Structure". UNIGEDS is a SQL database schema that's meant to replace GEDCOM by being adopted by all genieware vendors who value accuracy and convenience in the field of data transfer among the various genieware products.

Elucidom means "the condition of making something clear" in the way that "freedom" means "the condition of being free". *Elucidom* is a *gedcomoid*, a text file that, like GEDCOM, intends to help transfer genealogy data among different applications. *Elucidom* is a suggested early step in any GEDCOM import program. It's not geared toward UNIGEDS or Treebard, but should be usable for anyone who wants to write a GEDCOM import program.

gedMOM is a different *gedcomoid* based on the notion that SQL is the most successful, well-supported, well-established, and most mainstream means of storing data along with each datum's relationship to other data. Anyone

who studies a gedMOM translation of a GEDCOM file can come away with a general understanding of SQL's primary keys and foreign keys without looking at any SQL. Getting the cardinality right in data relationships is gedMOM's main criteria. gedMOM intends to be thoroughly unoriginal: a GEDCOM file designed as if it were meant to stuff genealogy data directly into a UNIGEDS database, with no hiccups.

gedMOM translates GEDCOM data into a gedcomoid that follows the rules of cardinality and can deliver GEDCOM data straightforwardly into a UNIGEDS database. Since gedMOM and UNIGEDS, like SQLite, are unlicensed, public domain, open source tools, vendors who'd rather keep their existing proprietary back-end data structure can create their own export programs to match their back-end structure instead of matching UNIGEDS. While Elucidom is a fantabulous first step in any GEDCOM import program since it is agnostic in regards to the ultimate target of the incoming data, gedMOM is the opposite of agnostic. It's geared specifically to importing data directly into UNIGEDS. Its current usage is a teaching tool for the GEDCOM-curious who want to see what GEDCOM would have looked like if its creators had designed it specifically to match the superpowers of SQL's relational database functionalities and rules.

Correct design of relational databases depends on understanding, conveying and storing the real-world cardinality or relationship-type between pairs of data. There are three types of data relationship, defined by simple cardinality rules: one-to-one, one-to-many, and many-to-many. By adhering to these rules, UNIGEDS intends to be capable of replacing GEDCOM, and thus GEDCOM along with its many frustrations can be phased out.

Any genealogy software vendor can upgrade their code base to use UNIGEDS (a SQLite database) as their storage facility for primary data, while retaining that genieware's own personality and marketing appeal through its user interface and secondary features, as is already being done. Most users don't know or care what back-end their genieware

vendor uses, so vendors have nothing to lose by all using the same back-end for genealogy's primary data. If UNIGEDS never catches on, upgrading one's back-end to a more capable SQL schema is still a worthwhile effort.

Treebard GPS, a showcase of genealogy software (genieware) functionalities, is the first genieware app to use UNIGEDS. I wrote my projects in Python, Tkinter and SQLite for ease of development, but they can be translated into the programming language and SQL RDBMS (relational database management system) of the genieware vendor's choice. But I think you should use SQLite because it's easier and it's perfectly adequate. ("Lite" is a misnomer. SQLite is light on extraneous features, not on primary SQL capabilities.)

As the sole creator and proprietor of Elucidom, gedMOM, UNIGEDS and Treebard GPS, I publicly guarantee, promise, and swear on a stack of census schedules that the use, adoption, adaptation, etc. of all these products is 100% free, open-source, unlicensed, and fully in the public domain, and not under my control in any way. I also do not seek to create, influence, or control any group, committee, team, gang, cartel, club, conspiracy, corporation, or company for the commercial use of this work, now or in the future, but any and all commercial use of this work is explicitly and unconditionally allowed, encouraged, and undisturbed by yours truly until the end of time.

To donate with no strings attached for the use of the work I've already done with no strings attached, you may, if you like, contact me at stumpednomore-at-gmail-dotcom. I'm not a professional programmer, not for hire, and not young enough to change my mind about that. But I don't mind being paid, by the willing and able, for the work I was somehow willing and able to do for free. But if you're not able or willing to give me a financial pat on the head, then just have at it. It's yours.

Donald Scott Robertson
a.k.a. Luther Rangely

a.k.a. Uncle Buddy
a.k.a. Professor U. d'Guru
Southern Mindanao

INTRODUCTION

The process of translating GEDCOM into something your genieware product's data structure can import needs a name of its own. My GUI or graphical-user-interface product (Treebard GPS) has a free, open-source, public domain SQLite database structure behind it called UNIGEDS. Since UNIGEDS is public domain (unlicensed), anyone can use it for anything, for example you can use it as the back-end for your commercial software instead of cobbling together your own homemade data structure. You don't have to donate or pay royalties for its use, but you are free to contact me when your conscience starts getting to you. Please add this step to your do list now, if you plan on doing anything about it, so you don't forget, because the balance of this extended diatribe is gonna help you decide whether the work I've done is worth its weight in magical baubles. This treatise is not about selling your ancestors back to you, so I might forget to mention donations again.

The import step between GEDCOM and UNIGEDS needs a name of its own. Part of vendors' mistake has been to wrongly frame our import programs as some sort of anonymous work which we must keep secret lest someone else figure out how to deal with GEDCOM and give us a run for our money. Who the heck wants to encourage our customers to leave our GUI for a better one by providing an example to a potential competitor of how to squeeze meaning out of a GEDCOM file? Well I do, because I'm confident that my Treebard is a better GUI for the average genealogist, and Treebard is free, so I have nothing to lose by sharing my journey of dealing harshly with that creature GEDCOM and its habits.

The point being, importing GEDCOM is at least as important as the GEDCOM itself, so that import step needs a name of its own. As slowly as GEDCOM evolves (or is it DEvolving?),

the real work of GEDCOM import is not the work that went into creating GEDCOM back in medieval times, but rather the real ongoing work that has to be done right now by anyone who expects to be taken seriously as a vendor of genieware. If you claim to have a genieware product but your product has no import or no export, your software is not finished.

The former GEDCOM import project "GEDKANDU" joined other all-but-finished-but-then-replaced former project names such as "DATABOY", "STEPMOM", "IDjot", "GEDFRAME", etc. There have been many project names since my key to *survival*--that's "success" spelled wrong--as a novice GEDCOM warrior has been to happily start over from scratch whenever the growing code base got too wobbly and contorted to serve as a foundation for GEDCOM's pie-in-the-sky promise: a family tree in a different genieware without having to re-input all the data over again one factoid at a time. GEDKANDU stands for "Genealogy Data Kruncher and Nobody Dares Under-estimate it". GEDKANDU was the last project that didn't pan out, and the current import project is just called `gedcom_import.py`.

GEDCOM is still currently to be temporarily used, tolerated and accomodated. But it is not to be indulged, orbited around, or catered to. We cannot continue to dumb down genealogy software because of GEDCOM's dodderingly low expectations of itself. It is to be treated as the distant cousin who came to visit, stayed too long, takes up too much space, and draws too much attention to itself without giving back what one might reasonably expect. We need to be polite but firm: one way or the other, GEDCOM is never gonna pull its weight, and therefore GEDCOM has to move out.

GEDCOM is like a dark comedy without the comedy.

Well, maybe you have to be polite when you do your networking in the genealogy community, but I don't; I have nothing for sale, no image to polish up, no reputation to bolster up with artificial professionalism. I don't have the energy to pretend that I'm a someone in this field.

It's just my hobby. Treebard University does not exist to sell your ancestors back to you. We're trying to help wanna-be vendors of genieware to stay out of the gutter of GEDCOM addiction.

The above list of project names is the most recent series of starts, following up on many earlier abandoned starts on the GEDCOM import process for UNIGEDS. gedMOM is the first start that was not a false start. The current effort which began over three months ago was boosted by this realization: you can export your data without UNIGEDS and without GEDCOM. To protect the integrity of the data, just make your own gedcomoid (gedMOM) that works the way GEDCOM *should* work.

UNIGEDS is different from GEDCOM in that the importing developer who is brave enough to be among the first to use UNIGEDS will have ready access to everything he needs: all the right primary keys already exist, data is nested only where it should be, there are no pseudo-subordinate data that should have been primary data. The elements of genealogy are presented as they should be, as elements or basics: irreducible parts of the data structure, not as subordinates buried in a pretend hierarchy from which they must be laboriously teased out into the open by the importing developer.

Unlike GEDCOM, gedMOM imports data to UNIGEDS so clearly that anyone who knows anything about SQL databases will understand how to import it immediately with no further instruction. At the same time, someone who knows nothing about SQL can easily understand how gedMOM works by reading through it and following the references, and in doing so, will learn how SQL works without seeing the actual SQL itself. While GEDCOM is easy to read by eye, gedMOM is even easier in one sense because it follows the principles of SQL exactly, but in another sense it's harder to read because the reader has to look up a lot of foreign key references using CTRL+F in a text editor.

Because gedMOM is basically a human-readable paraphrasing of a UNIGEDS database, i.e. gedMOM references work like a

database (without the referential integrity enforcement of an RDBMS program such as SQLite), gedMOM is a stepping-stone to UNIGEDS. When you look at gedMOM, you see SQL's structure.

We have to structure open-source, public domain UNIGEDS according to how the world actually works, and when we are wrong about the world that we're trying to accurately represent, we have to change the structure of UNIGEDS. This still happens occasionally, as these projects are not commercial endeavors, so I'm not in a rush to announce that I've attained perfection. In fact, I hope I never attain perfection. I'm old already, but just starting to get used to myself.

The eventual goal is that seeing the light of UNIGEDS, even if it has to be filtered through gedMOM, will tempt the most conscientious genieware developers to replace their proprietary back-ends with open-source, public-domain UNIGEDS. A coalition of common-sense could spring up wherein vendors who are in competition with each other will agree to use the same data structure as their product's primary back-end while reinforcing their marketing edge by perfecting their user interfaces. When three or more vendors adopt UNIGEDS or some certain something that is carefully crafted to turn out even better than my UNIGEDS, those vendors who won't join the coalition of common sense could be left behind in the dust of genieware past, because a good thing, once it grabs hold, can develop a momentum of its own.

As for vendors who are committed to using GEDCOM itself as a database, there's still time for them to mend their ways before common sense takes over the field of transferring genealogy data among applications, which it certainly will do someday. GEDCOM's final replacement may or may not be some gedcomoid. As for vendors who are using neither SQL nor GEDCOM as a database, the future is wide open for SQL, so why fight it? There are new database programs popping up all the time, created by professional programmers, so

clinging to some genieware back-end you invented for a team of one is maybe not as great as I'm making it sound.

A peek at gedMOM might show people who can read text files with their eyeballs, but are still afraid of SQL for some reason, what UNIGEDS is all about, compared to the punishment we're always giving ourselves with GEDCOM.

The real purpose of gedMOM is to make Mr. and Mrs. Genieware Vendor pine for a universal back-end to their product so that every BillyBob and his uncle can switch to that lovely Ma & Pa user interface that they've worked so hard to create. My own homemade interface is called Treebard GPS. Treebard is the first genieware to use UNIGEDS as the back-end, and one purpose of this treatise is to expose the mental hurdles I experienced while actually developing gedMOM and Elucidom, two different ways of translating GEDCOM to something beautiful. Since I have no trade secrets, someone might get some good out of one or more aspects of my work.

Remember when electrical plugs had not been standardized yet? Neither do I. Before this actually happened, I'll bet one or two manufacturers called their product "the standard". Our grandparents came to their senses on that problem before we were born. Computer genealogy will someday come to its senses too, if you and you and you really want it to. UNIGEDS stands for Universal Genealogy Data Structure. As you scan through a gedMOM file visually, there are so many values to look up, because of the many foreign keys (tags ending in "_FK") that it creates the *illusion* that gedMOM is complex. This is similar but opposite to the *illusion* you get from reading GEDCOM almost effortlessly by eye. The hierarchical line-numbered nesting used by GEDCOM to indicate relationships among data are wrong, upside-down, backwards or misleading so often as to be sometimes worse than useless. So the fact that GEDCOM is easy to read by eye is kinda irrelevant since computers are the ones that have to read GEDCOM, and they don't look forward to reading their next installment of the GEDCOM saga. The many foreign keys in a gedMOM file will force

anyone who studies it to look values up within the same text file to find out what is related to what, instead of having too many subrecords in the primary INDI record, for example, competing for the computer's attention. Just as importantly, you'll be able to tell *how* pairs of data are related (see **Cardinality**) based on where the pairs are linked to each other.

The foreign key values for data in a family tree are easy to look up in gedMOM; they're all in the file. I use a simple free code editor called *Notepad++* which has all the right features. The easiest way to find two gedMOM values (a primary key and a foreign key, or as GEDCOM calls them "identifier" and "pointer") that are close together is to double click an ID number to highlight it. With preferences set right in Notepad++, all other instances of that ID number will also be highlighted. The other way is to use the search function Ctrl+F to find the value associated with the ID. Here's an example of a gedMOM primary key (ID number) and its associated copy or foreign key reference.

```
PRSN 55
* *
PRSN_FK 55
```

The superficial complexity of a gedMOM file is what makes it easy to read by a computer program designed to create a SQL database from the text file. On the other hand, the seeming simplicity of a GEDCOM file is that it uses handy line numbers to visually indicate supposedly hierarchically related data. These numbers are what create the illusion. It's so easy to read that it looks like fun to create a program to read the file and make it do something.

Unfortunately, the hierarchy indicated by GEDCOM's assumptions is often incorrect and even nonsensical, maybe because GEDCOM was created when SQL and computer genealogy were both in their infancy. A text file will never enforce referential integrity or anything else; GEDCOM can be wrong forever and will never tell on itself. SQL is the opposite. To interface with GEDCOM we rely on a complicatedly written

manual of inconsistent and unenforcable rules to write GEDCOM "correctly" while knowing how deficient the design and implementation of GEDCOM really are at best, and vendors end up using custom tags because it's not our fault GEDCOM is so unresponsive to the needs of genealogy to describe the real world.

UNIGEDS, gedMOM, and Treebard GPS are demonstration models. They are free to use without permission. They are open source, completely free, and public domain. I don't waste much energy soliciting donations, but if someone wants to actually compensate me when they freely borrow some or all of my work, I'll figure out a way to accept a donation. I'm at stumpednomore-at-gmail-dotcom.

TAGS AND KEYS

gedMOM tags use the "YHWH vs. AE" principle: consonants are better syllable suggesters than vowels.

Primary tags in GEDCOM are marked with an asterisk*. In gedMOM, any tag that corresponds to a database table is primary, and the rest refer to database columns, some of which are foreign keys, and those end in _FK.

gedMOM TAG	GEDCOM TAG	GENEALOGY ELEMENT	NOTE
PRSN	INDI*	person	living or once-living being
MDIA	OBJE*	media	mostly computer files
RPST	REPO*	repository	where source was found
SORC	SOUR*	source	between source type and citation
CUPL	FAM*	couple	spouses, partners, parents, etc.
NOTE	NOTE*	note	details that don't fit elsewhere
CNTCT	SUBM*	contact person	submitter of the GEDCOM file
CTTN	PAGE	citation	where a source makes an assertion
ASRTN	TEXT	assertion	what a source says at a citation
GNDR	SEX	gender	'unknown' can be used for anything
LCTR	CALN	locator	leads directly to source citation

IRREGULAR FOREIGN KEYS

PRSN1_FK	HUSB	half of a couple	display on left
PRSN2_FK	WIFE	half of a couple	display on right

LUTHER'S 95 THESES

The *How-and-Why-to-Replace-GEDCOM* Rules and Rants

Here are my spontaneously generated, slightly organized, so-called "How-to-replace-GEDCOM rules" liberally peppered with complaints about GEDCOM's assumptions, attitudes and mistakes. If you don't like the taste of pepper, just skip to the APPENDIX, which I surgically removed, where you could have read the boring version of the same thing, if I hadn't removed it. This exciting, adrenaline-pumping, action-packed, superhero version is all that's left.

UNIGEDS is the replacement for GEDCOM since by replacing the current back-ends of all participating genieware programs, the UNIGEDS database can be shared directly. Knowing that such a common-sense step is not likely to be taken by all genieware vendors this week, I'll try to increase the potential appeal of my work by going to the trouble to write code that will:

Export the data in the gedMOM format, a gedcomoid (text file) which works with identifiers and pointers like GEDCOM, but has many more of them, thus divulging the exact schema of UNIGEDS to anyone interested enough to follow the links, without their having to look at any SQL.

Export the data in the GEDCOM 5.5.1 format with absolutely zero custom tags in spite of the pain it caused me to dumb down UNIGEDS' data to the GEDCOM format.

Import the data from GEDCOM 5.5.1 via a process called Elucidom which translates GEDCOM into a gedcomoid (text file) whose tags each have exactly one usage, as well

as other measures chosen to provide consistently-constructed gedcomoids to serve as foundations for the steps that follow it. The Elucidom file has no design connection to Treebard or UNIGEDS, so it can be used by not only Treebard, but by any genieware developer who wants to write a program to import from GEDCOM to any back-end structure.

Import data from Elucidom into a collection of Python dictionaries, and from there into a SQLite version of UNIGEDS which I would like the genealogy world to consider the demo version of a working model of a replacement for GEDCOM, even if more experienced programmers would like to rewrite it in a more efficient program than Python or a more complicated RDBMS than SQLite.

Open and use the UNIGEDS back-end in Treebard GPS, a graphical user interface written in Python and Tkinter, so that anyone interested in exploring any of my suggestions can see the end result. Treebard is a showcase of basic genealogy software which operates the primary import and export functionalities listed above. Treebard is meant to demonstrate GUI features I prefer over the bad *feng shui* currently available from genieware vendors. Treebard can be used to start and build family trees with or without import from GEDCOM or other UNIGEDS-based programs. Treebard is the first UNIGEDS-based app.

The practical everyday purpose of *Luther's 95 Theses* for me has been to give me a Complaints Department to go to whenever the self-imposed task of writing a GEDCOM import program becomes an gruelling task. It was during the past months of starting and starting over and starting over again on a GEDCOM import program that all 95 of the *How-to-replace-GEDCOM* rules were written. I call them "rules" because I want to have something to call them. But don't take anything/everything I say

literally as if it were meant to be the last word; that's not the intention of this book-length rant. The purpose of this book was not to write a book, but to help me to maintain some part of my sanity while trying to use what little I knew about designing a software project to translating GEDCOM--which was obviously created by neophyte computer geeks long, long ago--into usable genealogy data.

Without much further ado--or rather, saving the further ado for the long-winded but informative discussions below--here is Rule Number One:

GEDCOM is Easy to Understand--Unless You're a Computer

1. There's a reason why you can find jillions of explanations of relational database design (RDBMS, SQL) on the internet and hardly anything useful and simple about using GEDCOM as a developer. That simple reason is that GEDCOM is not simple nor very useful, and never will be either.

The fact that you can easily grok GEDCOM by eyeballing it is an *illusion* because understanding it with your brain is dozens of times easier than explaining how it works to a computer. Brains can take a hint, eyes can look back and forth among the related lines, but computers read lines of text one at a time.

GEDCOM does *not* need to be fixed, because it will never meet the needs of genealogy. It needs to be put out to pasture. It is antiquated technology from the primordial dark ages of consumer software design.

I offer up gedMOM, a different text file, not necessarily as a replacement for GEDCOM, but as a bridge to open-source, public-domain UNIGEDS, a real-world relational database that we can all use to back up and share our family trees created by competing genealogy interfaces. That's *if* the genieware vendors adopt UNIGEDS, or something even better, to replace their current proprietary back ends.

And if the vendors don't adopt a SQL structure as a common back end, then they can use gedMOM instead. In some important ways, it works like a SQL database, but like GEDCOM, it's a text file. Unlike GEDCOM, it's not confusing because there's only one way to do things, and... well, just keep reading.

Writing a GEDCOM Export Program is Easy

2. This book is about my experience trying to write a finishable GEDCOM import program. It never occurred to me until I was two days into the writing of a GEDCOM *export* program, why we are oh... so... *doomed*.

Here's why: writing a GEDCOM export program is about as hard as slicing butter. Because of this unfortunate fact, we could end up using GEDCOM--or is GEDCOM using us?--until hell freezes over and spits us back out.

May our ancestors forgive us for ruining computer genealogy before it could get started.

But why is it so easy to write a GEDCOM export program when it is so difficult to write a GEDCOM import program?

Simple. With the export project...

...you're starting from your own data structure, so you aren't lost in the wilderness of someone else's ideas of how things should be done;

...you don't use the tags you hate;

...you don't use the tags you think are stupid;

...you've just spent four months trying to write an import program so you know very well which tags you hate and which tags you think are stupid;

...you can make up your own tags (don't do this);

...you can resuscitate tags from previous versions of GEDCOM (don't do this either);

...dumbing down your hard-won data is easier than smartening up somebody else's data structure, and worst of all:

...GEDCOM is so rubbery, there are so many ways to do one thing, that you feel free as a bird and can write your .ged file in a number of different ways depending on 1) which tags and specs-constructs you hate the least, or 2) which tags and specs-constructs are easiest to comply with, and finally:

...closely related to the previous item, some of GEDCOM's primary records--NOTE, SOUR, OBJE--don't have to be treated as primary records, allowing you to skate through the project with no regard for the basic DON'Ts of computer programming such as "don't repeat yourself". Because GEDCOM said it's OK.

Don't do that either.

Writing a GEDCOM Import Program is a Real Education

3. Nothing in this document is meant to sound insulting to the creators of GEDCOM, to the GEDCOM replacement committees who have not managed to replace GEDCOM, to the genieware vendors who continue to use and abuse GEDCOM, or to the genieware users who have to do a lot of work over in spite of the odd notion that GEDCOM is a "standard" for data transfer. Unfortunately I have just enough experience trying to match GEDCOM up with some computer code that I can no longer discuss GEDCOM without falling into sarcasm. This is a personality defect on my part and I apologize for the inconvenience.

Trying to convince you that GEDCOM needs to be replaced yesterday is a small part of a hobby which I find enjoyable and personally as fulfilling as it is difficult, so much more difficult than doing genealogy itself.

I'm an amateur genealogist with many small trees under my belt. I usually study the ancestry of obscure old-time inventors, the more obscure the better, who wanted to do

impossible things with compressed air. I even know a little about my own family history, but I prefer studying families whose trees have not yet been published in great detail. To me, doing genealogy is a treasure hunt.

One reason it took five years to get around to writing a GEDCOM import program is that I don't import GEDCOM very much. I prefer to do my own work. For example, when I was a rockhound, and as usual my digging companion was more talented than I, he would sometimes feel guilty for going straight to the good stuff while I scratched around in all the wrong places, and since he felt guilty, he would try to give me one of the rocks he'd found. I didn't want him to feel bad, so I'd accept the token, but secretly it is a cold, hard fact that no gem found by someone else, no matter how shiny, was ever the equal of a rock I'd found myself.

4. Q: Aren't I expecting too much from people? How can I expect GEDCOM to be perfect? Don't we just need to force ourselves to bite the bullet and use the tools we have?

A: As for whether I should be writing this diatribe or not, I come here because if I'm ever to finish a first draft of a GEDCOM import program, it's a practical necessity for me to vent against an ugly, unbalanced, non-symmetrical, inconsistent, illogical, inaccurate, ambiguous, annoying, filthy, stinky, icky, deplorable, semi-useful compromise of a thing that could be made better, could have been made better 40 years ago, and now has been raising its stink so long that everybody is used to it and acts like it's some sort of perfume. Yes, I have a physical need to write this rant. Can't you tell? The question is, can you handle reading it???

5. This collection of complaints and suggestions is organized about as well as GEDCOM, and easier to follow than the GEDCOM specs. Isn't that supposed to be good enough?

6. Why should I be so annoyingly caustic, abrasive, and sarcastic? Well, I don't know for a fact that I *should* be

that way, but who wants to scale the Great Wall of Shoulds today? Not me.

I'll answer the question anyway, with a metaphor. A pack of canine creatures on a primitive planet feels they may be under attack. Maybe a single dog hears a cry of distress or whatever. He starts howling, and the whole pack starts howling together. The next pack of dogs (their enemies or competitors next door) have no reason of their own to get excited, but they hear the neighboring tribe and start howling too. And the next, and the next. Every dog for miles, including every dog's canine rivals, has now been warned that a dog-eating monster is in the area. By warning his enemies of their common enemy, each dog has transcended individuality and helped his rivals, and therefore his species, to stay alive.

In our times it is not fashionable to be the squeaky wheel in most areas of endeavor. Hobbies are not supposed to be contentious. Genealogy is no exception. The field doesn't attract that many loudmouths. But if I holler loud enough, someone else whose primal instincts haven't been completely squashed down inside him might also pick up the hue and cry and pass it along. When the genieware vendors, who have the most to lose by not playing politics, join the choir, it's gonna get nice and loud.

The vendors may or may not be friendly with each other, but in times of a common enemy, if someone else hollers loudest first, they can all pick up the call of the wild with less than the usual hesitation. Together the genieware vendors can declare independence from that common enemy of their customers which has been declaring itself a necessary evil, finally rejecting the droning claim that our being chained to that enemy is really a symbiotic (codependent?) relationship that none of us can live without.

I can't say it's a warm and fuzzy feeling to be a squeaky wheel, that sore thumb that sticks out, but that's who I've always been, so why not carry on being who I am anyway? Someone's gotta start telling the truth around here, and it

might as well be me. I can handle the embarrassment of being the one who failed to sing along with the pack.

7. UNIGEDS was created by an amateur genealogist (myself) who learned just enough computer programming to create his own genealogy software database and user interface features by working more than full time at it for over five years. Don't waste your energy tearing my work down or building it up. All my work is in the public domain. If I can do it, you can do it better. If I've done it wrong, my work will be ignored. I'm not a threat to any genieware vendor, past, present, or future, nor do I want to be. If I'm wrong, prove it by doing what I've tried to do, and doing it right. I'm just an old man with a hobby and a weird sense of humor. Try to be nice, if you have a bone to pick with me, and we can have a nice civilized conversation about it.

8. Last night I was almost ready to delete this document, throw out all my code, delete my website, forum, and video channels, and never think about writing genieware again.

Why?

GEDCOM burn-out.

So I cleaned some things out of the do list, reprioritized some things that were less essential, and started this morning on what I've really been wanting to do: translating gedkandu.py's output (Python dictionaries that already contain everything from the .ged file that's worth saving) into a gedcomoid I call "gedMOM".

A gedMOM or .mom text file is structured exactly like UNIGEDS so that the data will slide right into Treebard's SQL database like greased lightning. This has already been tested. It worked so well, months ago when I tested it, that the experience of testing it is what motivated months of full-time work, resulting in my first successful GEDCOM import program. By "successful", I mean that all the primary data worth saving has found its way into Python dictionaries.

As for the rest of the data, the import program is finishable, I made sure of that by starting over many times, so be my guest: make it better. I'd rather input data manually via the Treebard interface than spend more time working on GEDCOM's numerous edge-cases and weird ideas.

The exceptions log was going to be an interactive GUI but fortunately I was able to convince myself that it would be wrong to mix import and user input into one process. For now, in order to be able to breathe, I am just ignoring GEDCOM's worst features, except to send them to the exceptions log. I'm doing broad strokes now, getting it done, having fun. The horrifying details will still be there if I remotivate myself to handle them.

9. After the final do-over of my GEDCOM import file, which was essentially finished when I started over, these are my final instructions:

Stop parsing what the line means and just put each line, in its corrected form, where it goes in a translated file, such as another gedcomoid, but one whose structure matches your exact needs.

GEDCOM doesn't need to be imported. It can't be imported, because its structure doesn't match the data structure of your genieware. It only has to be translated. Don't try to get it perfect on the first read-through; you'd end up modeling the inconsistencies of GEDCOM and the code would become an unreadable, unmaintainable disaster. Don't fix what can't be fixed, just translate the good parts to a gedcomoid that matches the structure of your dreams, and import that.

What made me start over again, over and over, when I was all but finished? When there was literally one problem left to solve?

In programming, I find that if I get to the last straight-away in the marathon, and the straight-away ain't straight, then my ability to improve and maintain the code I've

somehow managed to cobble together is going to be nil. The right code base doesn't solve hard problems last; it solves hard problems first, and the rest should be relatively smooth sailing. I'm not saying this as an expert; I'm just guessing as someone who's done pretty much nothing else for the past five years. That don't make me a somebody, but it qualifies me to have an opinion.

The result of starting over when I'd pretty much all but finished: my second finishable import program bore little resemblance to the first, because I now knew what to expect, and unlike my experience while writing the first finishable import program, I now had an intuition for what order to do things in, and for what was really necessary and what was just fluff, or endless fiddling with edge cases.

I also was so disillusioned about GEDCOM's potential that I stopped trying to impress some imaginary critic and instead just did it my way. I had done enough research by now to become convinced that, due to the human tendency to embrace the status quo for no particular reason and to ignore new and better ideas for similarly illogical reasons, only a push equivalent to a major marketing effort would cause my work to displace GEDCOM. Such a push is not going to come from me. Others can finish what I've started if they want, if what I've done isn't good enough, deep enough, strong enough, etc.

Researching GEDCOM is an intimidating experience if you aren't a well-trained programmer, which I am not. I have no problem reading GEDCOM by eye or importing GEDCOM for my own needs, so I stripped some things out of my do list, including things I'd already proven I could do such as laboriously handling a few custom tags and the like. I boiled my project down to what it really is: a hobby that I'm doing for my own entertainment, and so I will have some genieware that suits my fancy. When I decided to forgo the Interactive Exceptions Report, the recent eruption of heartburn that had started to plague me just disappeared.

Custom tags? Exceptions? What--me worry? GEDCOM is not gonna ruin my hobby or my health.

The purpose of Treebard GPS and UNIGEDS is not to receive imported data. The purpose of Treebard/UNIGEDS is to be so easy to learn and fun to use, and to be so correctly detailed in its data structure, that users will actually enjoy inputting data that GEDCOM can't handle graciously. My reason for taking on GEDCOM, on the other hand, was a bit neurotic. I just wanted to look like I was up to the task of replacing GEDCOM, and didn't think anyone would take me seriously if I showed up to the GEDCOM party but refused to mingle with the crowd. Not that I have a clue how to mingle with crowds, but a fella's gotta look like he's trying?

Hell's bells, boys and girls, I just plain don't mingle. That is not what I do. I don't mix with oil or water, and that's gonna have to be OK, cuz it ain't a-gonna change. I handled GEDCOM to my own satisfaction, then I backed off so I could handle it right: keep the good stuff and ignore the rest. I can input by hand whatever GEDCOM can't import correctly, and I can enjoy doing it, because Treebard is not tedious to use.

Having no ambition to be a Somebody Among Genealogists, it became obvious that I have to import GEDCOM the way I'd import it if no one else was ever gonna look at my work. Who am I trying to impress?

Nobody. I'm just trying to have a good time.

10. This document was written as a venting post for frustrations incurred during the writing of my first finishable GEDCOM import program, gedkandu.py. After two months of full-time work on gedkandu.py and its several short-lived predecessors, I stopped short of finishing GEDKANDU because the problem I'd saved for last should have been handled first. To finish GEDKANDU, with its delusionally lofty aspirations, I would have had to rewrite some overly complex code and I had no interest in doing it.

GEDCOM, coupled with my mistake of trying to prove I could handle every single thing it threw at me, burned me out.

At that point I started over from scratch with a completely different approach, and wrote another nearly successful GEDCOM import program, `gedmom.py`, in a week. The `gedMOM` concept was not brand new, but the file `gedmom.py` was. This new import program was never intended to be a finished import program, and that was my ineffable stroke of soaring anti-genius: since it was not intended to be a complete import program, I was almost able to finish it.

GEDCOM itself is not a finished import program. The GEDCOM tool, riddled with problems as it is, sat idle down at GEDCOM Central for 20 years. That should have been someone's first clue that the project was not considered completable by its own creators. I'm no expert on the versions of GEDCOM, but it seems to me that when GEDCOM 7 finally came out, and tried to solve complex problems by piling on more complexity, sounding more pretentiously technical, and officially institutionalizing custom tags with more rules on the right way to use them, that should have told us that the new crew down at GEDCOM Central had gone down the blind alley of throwing good money after bad, instead of admitting defeat and asking for forgiveness for everyone's inconvenience and wasted time.

And still we consider GEDCOM a standard? This document lists many reasons why it is not a standard. You might call it a sub-standard, but that's an adjective, not a noun, so... substandard what?

It's not even a substandard *standard*. It's a sub-standard *tool* for transferring data between different genieware applications. Those who claim it's a standard just make the statement as dogma, but offer no reasons to back up such a statement. The well-meaning defenders of GEDCOM will keep it alive, but they are not immortal, and neither is their imperfect standard. A standard that doesn't pro-actively strive for perfection with a sense of urgency is a pretend standard. A standard in GEDCOM's condition--imprecise,

impractical, inconsistent, asymmetrical, unpredictable, unenforcable--is about as codelike as that Secret Spy Code Ring I ordered out of the back of a comic book in 1964.

My newly unperfect but finishable import program works well to import the basic elements of genealogy normally recognized by genieware, such as persons, places and events. It is not a complete GEDCOM import program because GEDCOM is not a complete tool for transferring data and I'm tired of pretending it is. The program happily imports data from the parts of GEDCOM that are reasonably sensible. But every single part of GEDCOM that makes me want to throw furniture is relegated to the exceptions log, which is also finishable.

I'll be jumping out of the GEDCOM stewpot forever, as soon as I work the bugs out of my finishable GEDCOM import and export programs. Writing Treebard GPS and UNIGEDS is actually *rewarding*, compared to exploring the intricacies of GEDCOM's shortcomings, because down here at Treebard University, our standards include precision, practicality, consistency, symmetricality, predictability, and enforcability. That latter is of utmost importance, because no text file will ever enforce anything, and a data transfer tool that enforces nothing, and can't be forced to enforce anything, will never be a standard, no matter what we call it, even if its other flaws somehow manage to repair themselves.

The missing standard down here at Treebard University is professionalism. We don't want professionalism as a standard, because it's a decoy, a polished-up facade. Part of the Treebard dogma is that genealogists can write their own software. So while we do make an attempt to emulate those standards listed above, polishing up some phony image of ourselves as "professional" is considered a big joke around here. Naturally this will turn some folks away with their noses up in the air, especially a few of those who were hoping to sell our ancestors back to us.

But there's nothing about my lack of interest in marketing myself that should scare anyone away. All my work is in the public domain, 100% unlicensed. You can take my work and translate it into your own project and make all kinds of money off of it, and I will never even email you to beg for a royalty. When code leaves here, it's yours, not mine. I did this for fun, and other than a few odd and somewhat unexpected donations, that's all I ever expect to get out of it. Except for the satisfaction of creating something that was missing from the world.

So my advice for those who want to use some part of my work, or all of it, to profit themselves: have fun!

SQL: Top of the Class Since GEDCOM was in Diapers

11. The right medium for data exchange is a universal database structure. SQL is a mature computer language for creating relational databases and SQLite is a mature RDBMS for creating SQL databases which is not particularly lite on usefulness but is relatively lite on dependencies like setting up a server connection.

A SQL database is a flexible, growable, shrinkable network of simple data pairs. A hierarchy such as GEDCOM's line numbers is none of these things. Insert a _CUSTom tag and the chain of meaning as intended by the hierarchy of line numbers is broken. Any tag that doesn't follow a symmetrical, predictable pattern breaks the chain of meaning, the hierarchy is broken if one chain is damaged. But it's just a text file, it doesn't know it's broken.

A SQL database tells you when you break the rules. GEDCOM creates its version of supposed flexibility by breaking itself, and the creators of GEDCOM tend to solve its problems by making it more complicated.

12. I hate to sound like a silly goose, but there's no other way to say it: gnats-eyelash specifications are right for nuclear power plants, not genealogy software. We must keep some perspective here on what is really going on. We have to understand what little we are able to understand

about living people and their psychological needs, including genealogy software vendors' addiction to GEDCOM.

This text-file gedcomoid approach to data transfer comes with no enforcement powers of its own. That's why someone is going to step forward with rules; why the rules keep changing: people are running this circus, not a programmed set of enforceable rules. A text file doesn't know what rules are, in regards to genealogy data, so specifications take the place of software with its built-in rules.

On the other hand, SQL is a computer language, not a text file. A relational database written in SQL doesn't need specifications, it just needs genieware creators to learn a little SQL (it's easy and it ain't rocket science). Learn SQL and you know how to create a genealogy database. That's because SQL and database relationships come with their own rules *and they are the same rules for genealogy data as for any other kind of data*. So all you have to do is follow the rules of the SQL language, which requires a little thought and experimentation.

There are pretty much three sets of rules you have to know about as you design the structure of a genealogy database: cardinality, normalization, and enforcement of referential integrity. These are all simple concepts with fancy names.

1) Cardinality: know which of the three relationship types is right for a pair of data. For example, person-to-name is a one-to-many relationship because one person can have many names but each name refers to only one person. (Two "John Smiths" have two different names; they're spelled the same but have different name IDs.) There are only three types of cardinality to learn, and the rules are simple, covered in another place here in this document.

2) Normalization: a limited set of simple rules exist which teach what not to do in database design. For example, requiring the same data to be entered twice in

the same database would "denormalize" that database. You can learn the key points of this topic in one hour watching YouTube videos.

3) Referential Integrity: each element of data has its own unique ID number (primary key) in a table. To refer to that piece of data, such as a name or a place or an event or a person, you use its ID number as a "foreign key" in a different table, or even in the same table for that matter. For example, the person ID from the Person table would be used as a foreign key in the Name table, while the name in that table has its own primary key which could be used somewhere if you needed to reference that name, for example, a note or an assertion could refer to a name using a name ID as a foreign key.

Unlike a text file, a SQL database has built-in enforcement. For example, a foreign key can't refer to a primary key that doesn't exist. Or you can define constraints on any column of data, for example maybe you'd want to say that a name record can't have a blank person ID (foreign key). The text file's contrasting total lack of enforcement and constraints adds up to simply this: GEDCOM is an incoherent, unenforceable disaster from the primeval dark ages of database management when no one knew whether SQL was actually gonna catch on. It did. But in 2024, to validate a GEDCOM file, you need more software, and every GEDCOM validator will give you different results. Because the GEDCOM specs also don't enforce anything; the specs are just more talk.

Cardinality Means Never Having to Say "Ooops"

13. Take a peek at these GEDCOM lines which are subordinate to an INDI primary line:

```
1 SOUR @S5@           (a source ID foreign key)
2 PAGE 25763/1960      (a citation string with no ID)
```

The GEDCOM strategy for sources and citations is to put the data in a jar, shake them up, pour them out and then read them like a fortune teller reads tea leaves. Also known as the "figure it out" strategy. The fact that the answer is usually in the specs doesn't help as much as you might think. The specs should be four lines long: "SQL. Cardinality. Normalization. Referential Integrity."

To make matters worse, genieware developers have built their products around the limitations of GEDCOM to an understandable degree, for reasons having nothing to do with the integrity of historical recordkeeping. Then they've created genieware that is anti-intuitive to learn and use, peppered with silly icons and grotesque abbreviations, crowded and jumbled and based on mis-assumptions vs. clean, simple and self-explanatory. Then along comes lazy little old me who won't look at the instructions, and the result is that all my old GEDCOM files show the sources linked directly to the individuals, because I used the program I used to use for years before it occurred to me that a source is generally linked to an event or a name or something concrete.

Not that persons aren't concrete, some more than others, but other than to incidentally imply that a person exists, the purpose of sources is generally to say something about events, attributes, names, and the like. Once I realized I'd created dozens of meticulously sourced trees with every single source linked only to individuals, I developed a distaste for using genieware written by other people. I'm not blaming the genieware developers for making software with a GUI that's too complicated and a database that's too simple, because my mama told me not to blame stuff on people. It might sound like I'm blaming folks, but in all honesty, did I not just spend 5-1/2 years creating my own genieware. By golly, I think I did.

After years of thought and hard work, I still haven't found any obvious connection between a source and a person.

Here's what I have found instead: A source says something, and in genealogy this something is an "assertion". But it's not really the source that should be linked to an assertion, it's *where* the source says it: the 1880 census, in general, says jillions of things; a citation within the 1880 census is what should be linked to an assertion.

Assertions are linked to events or names or whatever user-made conclusions the assertion relates to by placing an event ID foreign key or a name ID foreign key in the assertions table since one event or name can relate to many assertions, but each assertion should relate to only one event, name or other factoid. Events and attributes and names and such are conclusions on the part of the user. Assertions are what the source says. Assertions and conclusions are not the same thing, and UNIGEDS reflects this by keeping them separate and linking them as required to reflect real-world situations.

Citation foreign keys belong in the assertions table because one citation can make many assertions, but each assertion is made by only one citation. GEDCOM has a TEXT tag for assertions. It doesn't see much use, since genieware developers until now have barely bothered to care about recording exactly what the source said, the "TEXT" tag is vaguely named, and the GEDCOM standard has this tag linked to the source when it should be linked to a place within the source. TEXT should be subordinate to PAGE (citation) not SOUR (source).

There are also locators, which are coded numbers and other strings of text, like call numbers in a library and URLs on the internet, that take you directly to a source and/or citation even if you have no other information about the source or citation. UNIGEDS has a table for locators. Locator ID foreign keys are linked to repository ID foreign keys on a one-to-one basis in the `repositories\_links` table, in the same record where the locator can be linked to a source ID foreign key and/or citation ID foreign key.

The difference between locators and citations is that a citation tells you where the source makes its assertion, and the locator tells you where the source or citation is within the repository.

My former practice (before I thought to create my own genieware) of linking sources to persons was wrong. There is no obvious connection between sources and persons. The right way to link sources, citations, assertions, repositories, and locators to the events, attributes, names etc. that they need to be linked to, is not obvious either, but it can be figured out by thinking carefully about each relationship as to its cardinality: is the relationship between any two things a one-to-one relationship? A one-to-many relationship? Or a many-to-many relationship? There is no substitute for getting cardinality right when designing a database.

14. These are the rules for cardinality in SQL tables.

(1:1) Data with a one-to-one relationship go in the same row (same record) of the same table in the database. If they need to be in separate tables, with one of them referenced by a foreign key, then that foreign key field needs a UNIQUE constraint placed on it so it can't accidentally be used wrongly to reflect some other cardinality.

(1:M) Data with a one-to-many relationship each need their own table with their own primary key. The foreign key referencing one of these primary records has to go in the many-side table. Here's an example. One source has many citations, but looking at it the other way, one citation refers to only one source. You have to look at it both ways so you don't see just the one-to-one side and get the cardinality wrong. One source:many citations. Reverse it and you get one citation:one source. That's how 1:M relationships work, and how they are easy to get wrong. Look both ways. The many-side is citation, so the source ID foreign key goes in the citation table.

(M:M) Notes are a good example. A person can have many notes about him, and each note can be repeated many times. By giving notes primary status with their own table, each note has its own primary key which can be repeated as many times as you want as a foreign key in a M:M junction table. The primary key of a junction table doesn't see much use. The table mainly exists to accomodate many-to-many relationships. The same note ID can be linked to any numbers of other elements in the family tree by putting a foreign key for the other element on the same row of the junction table as the note ID.

15. The other all-important topic in database design (besides learning how to use basic SQL, which is easy for us since genealogy is not rocket science): normalization. A denormalized database is somewhere between a danger to itself and useless to anyone else. I feel that UNIGEDS is currently between 95-100% normalized. I won't describe this topic further but there's a good explanation of it on the internet here: https://www.youtube.com/watch?v=GFOaEYEc8_8

16. GEDCOM's Prime Crime was just revealed to me, and it is this: GEDCOM determines its own structure. Of course it has been groups of people, not GEDCOM itself, that have designed GEDCOM, but my purpose is not to insult anyone's effort. I will never suggest that GEDCOM is wrong or inadequate on purpose or for conspiratorial purposes. GEDCOM has had its place, as a placeholder, while we wait for a real data transfer tool to show up. So I put it the personification way, to be polite: GEDCOM's Prime Crime is to choose its design based on its own interests, based on nothing, based on anything, based on this and that. It is not a coherent design because it has no coherent basis.

Well, you might ask, what *should* it have been based on? I'm glad you asked. UNIGEDS is based on real relationships among the real elements of genealogy. What are the elements of genealogy? They're the things that would still exist whether genealogy existed or not. The persons, places,

events, source documents, etc. upon which genealogy is based *do not exist because of genealogy*.

It's time we recognize that elements of the real world, to be represented accurately in a software program, have to be represented according to their actual relationships to each other in the world, not based on any cute ideas we genealogists can come up with about these relationships. Cute ideas such as the hierarchical line numbers of GEDCOM.

Nothing in the world can be measured, analyzed or described in absolute terms. In absolute terms, the world is a great big mystery and nothing can really be explained in all its infinite complexity. But we have the ability to relate things to each other, and thereby we measure our experiences, our objects, and ourselves in relative terms, and that's how we can reduce complex things to networks of simply-understood relationships between pairs of data. For example, by understanding and accurately reflecting the relationship between a source and a citation as well as the relationship between a citation and an assertion, we come to realize that the relationship between a source and an assertion doesn't have to be described and should not be.

SQL helps us with this by enforcing referential integrity; we create table schemas for our data, which enforce rules.

It was revealed to me just now, by the muses of genealogy, that the Prime Crime of GEDCOM is to be beholden to the structure of the real world, and that came to me when I realized that gedMOM, in contrast, has no structure of its own. That's what makes it such an interesting step in simplifying the transition from GEDCOM's arbitrary whimsy to UNIGEDS' enforcable structure. GEDCOM can have all the specifications in the world, but as a text file it can't enforce anything, so it can never serve as a standard in computing anything.

gedMOM is a conceptual bridge between GEDCOM and UNIGEDS because, like GEDCOM, it's a text file, but unlike GEDCOM,

it doesn't make its own rules. It is absolutely beholden to the structure of SQL & UNIGEDS, in the same way that UNIGEDS is absolutely beholden to the way that parts of the real world are related to each other. Unlike users of a text-file transfer tool, SQL users can't easily pretend that a database is correctly structured if it is not. A broken database makes its brokenness apparent, while GEDCOM limps along decade after decade and we bust our butts to prolong its misery.

The pain we experience, permanently, to keep GEDCOM alive, could be replaced by the temporary pain of transition to a system of file transfer that actually works as intended.

Primary and Arbitrary are not the Same Thing

17. In GEDCOM, If the value on an EVEN (event) line is "Y" and there's no date or place, it indicates the event is known to have taken place but there is no further information. Using "N" is not allowed by the specs, e.g. `1 MARR N` to state that a person was not married, but its meaning is obvious so it could be handled anyway. In my experience, I've needed to enter "unmarried" under Marital Status many times, but have never had occasion to state that someone was married. If someone was married, we usually have evidence of this so it's obvious.

But in the absence of any marriage event, an assertion that the person was single is evidence of a real attribute, not just "absence of evidence that the person was married." Just another case of GEDCOM trying to not mind its own business by making arbitrary rules. The specs don't specify a way to say a person was not married. You could use the FACT tag:

```
1 FACT unmarried
2 TYPE marital status
```

And that works, but GEDCOM's lack of symmetry sticks out here. The MARR tag can only be MARR Y but not MARR N

because there's this imaginary element in GEDCOM, the family event. There are in fact couple events, and that's what marriage is. But the right way to record an event is with an event ID, not a bunch of subordinate tags straining at the analogy of subordination which GEDCOM stretches past the limit.

GEDCOM requires MARR tags to be subordinate to FAM records, and that's why you can't say `1 MARR N`: to say someone was unmarried in that way, you'd have to create a FAM record for a person who was not in a couple. The solution as usual is to stop fiddling around with text files pretending to be databases, and switch to a proper database structure which all genieware vendors can use in common for their primary genealogy elements.

Since you can do the FACT.TYPE tag as shown above, why am I complaining that you can't do 1 MARR N? Because unsymmetrical code leads to dashed hopes. Every time a pattern is arbitrarily broken instead of being applied as uniformly as possible, a programmer has to do more work, and the more code we write, the more mistakes we make. This problem would not exist if GEDCOM supplied every element of genealogy with an identifier (primary key) instead of foisting this strained-past-the-limit subordination analogy on us as if it were accurate.

18. Good golly, GEDCOM creators, how could you start out right, with identifiers and pointers, and then just arbitrarily stop cold after thus identifying only seven types of elements, and then do everything else in some random other way instead? Are places not an element of genealogy? Events? Assertions? Citations? Just as bad, and maybe worse: you create primary identifiers and then tell us we don't have to use them. GEDCOM's subordinate line numbering system + custom tags + lack of primary identifiers = the death of GEDCOM which nobody bothered to notice, because no one will admit that we are in the very early stages of developing computer genealogy.

Which means that certainly most of the computerized family trees that have been done up to now are gonna have to be

done over no matter how long we wait to get started on that unavoidable task by replacing the archaic assumptions of the 1980s with stable, mature, ingenious and fully functional technology like SQL.

Transcending GEDCOM's Pecking-Order Approach

19. While it was once difficult for me to write code to use the leading line numbers to find out what a previous line said, it's no longer so; I found an easy way to do it after wasting weeks on techniques that worked in the morning, but which I no longer could understand after lunch. The most entertaining of these was looping backwards.

Line numbers and useless characters should not be used to indicate levels in a hierarchy of subordinate data. The tag names themselves should indicate what record type the field is a part of. We'd have to read the lines anyway, to get the numbers, so we might as well instead read the level right out of the tag name. "0 @I1@ INDI" in gedMOM becomes `PRSN 1`; GEDCOM's "I" and "@" mean nothing and serve no purpose. Foreign key references are like this: `PRSN\_FK 5` instead of GEDCOM's `1 CHIL @I5@`. The foreign key lines are inside a primary record, as in GEDCOM, and they start with the text of their own primary table. There are only really three kinds of lines in gedMOM: primary key lines, foreign key lines, and key:value lines. Here's one of each in a name record:

* *

```
NAME 55
PRSN_FK 3
NAME_STRG Teddy Roosevelt
```

The record separator `\* \*` replaces the zero-line starting numbers of GEDCOM and does not mix with the content of the data. Like every other line in gedMOM, the separator has exactly two elements separated by a space, where as GEDCOM for no useful reason has lines with 2, 3, or 4 items. All of GEDCOM's inconsistencies have to be dealt with by more

code and more complex code. Programmers like a challenge, and that might be one of the things we're up against. We should like this to be as simple as possible, because that would be good for genealogy, good for all of us. Complexity leads to inappropriate decisions about structuring data. Complexity spawns more complexity.

Unlike GEDCOM, every line in Elucidom and gedMOM will have the same number of items. The developer trying to import gedMOM data won't have to worry about whether the line has two things in it or three, etc. Also unlike GEDCOM, the line parts (tag & value) will always be in the same order. In GEDCOM the primary (zero-starting) record lines are "number value tag" while subordinate lines (non-zero lines) are "number tag value"...

```
0 @I1@ INDI
1 SEX M
```

...which forces the parsing code to slog through conditional tests which should not be necessary but are, due to arbitrary meaningless design decisions that present data in unsymmetrical ways, apparently for the fun of it.

20. GEDCOM is sometimes guilty of nesting just for fun.

```
0 @I5@ INDI
...
1 CHAN
2 DATE 30 JUL 2023
3 TIME 22:07:59
```

GEDCOM seems to mistake nesting for a great way to communicate programming logic. The DATE and TIME values are siblings, for example in SQL they'd both be columns in the same record, not one subordinate to the other. And the useless CHAN line is just sitting there saying "I'm in charge of this hierarchy." The specs should have made the subrecord like this:

```
...
```

1 CHANGED 2023-07-30-22-07-59

I can't imagine any programmer disagreeing with me on this point. The hairsplitting sometimes never seems to end down at GEDCOM Central. This change date thing, which is periperal to genealogy since it's about the genealogist, should take its place at the back of the class instead of being a difficult nuisance.

Like all difficulties mentioned herein, it's not the difficulty itself that I mind. It's the way a simple thing has to be communicated as if it were not simple. What does the "COM" in GEDCOM really stand for? Communication? Or complication?

21. While GEDCOM wants us to pretend that all family tree data is hierarchical in nature, which requires us to jump through weird mental hoops, UNIGEDS correctly arranges data into omnidirectional networks built up from simple, easily analyzed relationships within pairs of data.

Instead of a huge INDI record bloated with supposedly subordinate elements that should have their own primary records, UNIGEDS has smaller records with relationships coded according to the type of relationship between the two elements being linked. This coding of relationship type is not code at all, but schema arrangement, for example putting two data in the same record of the same table if they have a one-to-one relationship.

An example of GEDCOM's inherent make-work, crazymaking confusion is the notion that a PAGE (citation) line should be subordinate to a SOUR (source) line. Well citations are smaller than sources right? So why not? Cardinality, that's why not. Sources and citations are two different things, and one is not subordinate to another in a way that is useful.

"Subordinate" is an abstraction with fairly little application in many parts of the family tree data problem. While a physical citation is nested inside a physical

source, this nesting might be irrelevant to a useful approach to recording the relationship between them. By consulting the rules of cardinality, we see that the relationship between a source and a citation is one-to-many: there are many citations possible in one source, but each citation can only refer to one source. So the relationship is accurately and usefully represented by putting the source ID foreign key in the citation table. This is anti-intuitive only if you start off by assuming that all data is nested in a hierarchy depending on little stuff being inside big stuff.

Once you realize that source IDs are in citation records, you have to ask, "How do I link a source to an event or a name?" This excellent question leads to something that genieware vendors have missed because they have tunnel vision with GEDCOM the disappointing light at the end of the tunnel. The assertion table is all about what the source says. This is where the citation link and the event or name link have to go. Cardinality dictates this, not "intuition". It's easy to mistake superficial conclusions for intuition, especially with a bogus light at the end of the wrong tunnel.

The line-starting hierarchy-coding numbers in GEDCOM are the first non-productive GEDCOM dependency to go. Compare the above analysis of the relationship between sources and citations with the following line of GEDCOM:

```
0 @I5@ INDI
1 BIRT
2 SOUR @S6@ [source]
3 PAGE E.D. 10-10, Sheet 7A [citation]
```

The pseudo-intuitive approach of the GEDCOM creators was to imagine a genealogist asking himself, "What is the source for this birth event?" (Answer: source 6) "OK then, where in source 6 did the information appear?" (Answer: E.D. 10-10, Sheet 7A). Just slapping the data down in the order it occurs to us, and calling this a hierarchy, and letting our

victims work out the difference between our assumptions and the real world, doesn't jibe with today's sophisticated database structures. GEDCOM is the square peg and the relational databases are the famed round hole.

Like my old gym coach used to say, "Hurt yourself in practice or you'll get hurt in the game." In order to replace GEDCOM, we're all gonna experience some pain so that the genealogists of the future including ourselves can fly free of GEDCOM's inadequacies forever. Doing some of our work over every time we import a GEDCOM file is not less pain than biting the bullet and replacing GEDCOM now. It's the opposite of that. The replacement of GEDCOM will literally ring in the Golden Age of Genealogy, and that is something worth experiencing discomfort for.

One Goal, One Method

22. In a gedcomoid custom-designed to not melt our brains for its own entertainment, there would be one way of doing one thing. For example, in gedMOM all events and attributes use an EVNT (event) tag with a subordinate EVNT_TYPE. Better yet, the usually null 3rd item after GEDCOM's EVEN tag could be used for the event type value. Types should be values, not tags, because users should be able to add their own types. Instead of this situation where there are multiple ways of doing events in GEDCOM...

```
1 EVEN self-filling wine glass***
2 TYPE Invention
2 DATE BEF 1934
2 PLAC Paris, France
1 OCCU dogwalker
2 DATE ABT 1923
```

...it could have been done with consistency a criteria:

```
1 EVEN invention
2 DETL self-filling wine glass
2 DATE BEF 1934
2 PLAC Paris, France
1 EVEN occupation
```

2 DETL dogwalker
2 DATE ABT 1923

*** The third item in a GEDCOM `EVEN` lines is allowed and encouraged by the specs, whether your vendor's import program realizes it or not.

23. I just complained that GEDCOM gives us two ways to do events and attributes. Now I will go on to complain that GEDCOM didn't feel that was confusing enough, so gifted us with a special bonus way for us lumpers to drive ourselves crazy trying to keep up with the splitters' little games. Here are two ways of doing the same thing:

1 OCCU shoemaker--orthopedic shoes & boots

1 FACT making orthopedic shoes
2 TYPE specialty

These methods as well as others that I could add to the pile are not how I would have done it. There should be one tag such as EVNT for all events and attributes. (While it's true that events and attributes are not the same thing, telling them apart is not GEDCOM's problem. That's up to the interface designer. This question is covered under a different item.)

If you have *some* tags for event and attribute types like OCCU, RESI, BIRT, then you should have *all* of them. Which is impossible, so the right approach is to let the interface designer and the user decide what types are needed. Instead of the "we gotta draw a line somewhere" approach, this problem begs for the "don't even start" approach.

24. Can anyone tell me why I thought the root word of "specification" was "specify"? You know, as in "specific". Is it an act of specification to give three different ways of doing the same thing, while actually requiring two of them?

INDI.FAMC.PEDI or INDI.ADOP.FAMC.

```
0 @I3@ INDI
1 BIRT
2 FAMC @F4@
1 FAMC @F4@
1 FAMC @F9@
2 PEDI Adopted
1 ADOP
2 FAMC @F9@
...
0 @F4@ FAM
1 CHIL @F3@
...
0 @F9@ FAM
1 CHIL @F3@
```

One can be absolutely certain, after careful study of the above example from the specs, that this fella was adopted by family #9 after having been born into family #4. It might not be true, but GEDCOM is sure convinced. I believe the correct description of this repetition is *ad nauseam*. If this repetition is actually needed in order for GEDCOM to work, it's just proof that GEDCOM doesn't work

25. A tag that can be used for a primary record should not be used for a subordinate record. For example, in GEDCOM, a NOTE tag or an OBJE tag can be either a primary key or a foreign key. This sort of unnecessary cleverness on the part of GEDCOM's creators is the stuff of nightmares for developers forced to write programs to use GEDCOM. Rule No. 442 in programming: don't cram essentially separate things together under one name if they're gonna have to be teased apart later in order to be used.

A real world example for those who still want to mistake cute-but-useless for programming design: you've been hired to cook a stew to serve at the Carrot Haters Convention. You chop up a bunch of carrots and toss them in the stew, courteously announcing before the meal that everyone will need to pick the carrots out of their stew, even providing little dishes for everyone to place their discarded carrots

in. You do this year after year. In response to the complaining, you provide different sizes of carrot chunks or maybe prettier little dishes for people to put their discarded carrots in. Your name is GEDCOM; apparently you can't be fired from your job.

This analogy also applies to the case where GEDCOM tags such as NAME can be applied to different things which are unrelated to each other, because some GEDCOM creator thought it would be clever to terrorize genieware developers with tags that had more than one reason to exist. As always, this is not an insurmountable problem. It would also be really easy to fix the dang stupid mistake in GEDCOM's design but who still thinks fixing GEDCOM is someone else's problem?

26. Did I mention that the real purpose of this diatribe is to preserve some hopefully resuscitatable seed of my sanity while I try to complete a GEDCOM import program?

I vaguely recall, some weeks ago, spending a few days on ASSO.RELA and finally sending it over to the exceptions log when in desperation I learned by doing research that this tag is well-known to be defective, not to mention ill-mannered.

Now that I've spent three days trying to handle ASSO.RELA in the exceptions-report writing project, I have this to say about handling GEDCOM in general. This is not my opinion; it was revealed to me, so there will be no argument.

It's obvious that GEDCOM is inconsistent with my data structure UNIGEDS, and it's obvious that this is a source of my troubles. I cannot argue against the notion that if I were to structure my program in the same way that GEDCOM is structured, the data would be several times easier to force into my database. Of course, if that were the right attitude, then by all means, I could just pretend that GEDCOM is a database and store all my data in GEDCOM itself. But that's not the topic at hand. What I have to say goes a level or two deeper than that.

No, I am not at all bitter about having to work hard to match GEDCOM's stored data with my storage system's own idiosyncracies, in spite of the fact that I heartily embrace the notion that I have built a correct system that is correct because it accurately reflects the real-world relationships of real-world genealogy elements to each other. The discrepancy between GEDCOM and UNIGEDS is not my gripe.

What I'm here to rail against is the discrepancies between GEDCOM and itself. GEDCOM is unsymmetrical, unpredictable, and untameable because it is inconsistent with itself. Handling GEDCOM's data is like pushing a square peg into a round hole with a limp noodle, and once this is accomplished, in order to handle the next shape-shifting tag design, you have to turn around and force the square peg, now firmly wedged into a round hole, back out of the hole, with the same limp noodle, which is by now ruined.

That pretty well covers it.

27. Page 76 of the GEDCOM 5.5.1 specs show you how to use a SOUR (source) tag as a subordinate tag inside an INDI (individual or person) record, and then instead of using some sort of citation tag, the specs go on to concatenate the citation to the source. They call this a "SOURCE_CITATION" and now that they've shown us how to do it, they say that doing it is not encouraged. That's like giving a poisonous snake to a dog and asking the dog to avoid the snake. Most dogs in this situation will die trying to kill the snake.

Analogies aside, let me repeat what I read on one brave GEDCOM savant's blog: 'There is no such thing as a "SOURCE_CITATION".' Amen, brother. A source is a big thing such as "1880 census" that contains a small thing--a citation--such as "line 19 of page 395". The GEDCOM specs' example of how to do something that we are not encouraged to do might be the product of "too many chefs spoil the stew" plus "one of the chefs threatened to quit if he couldn't include his pet SOURCE_CITATION toy." But I should

not guess what might have been on someone's mind when they rubber-banded GEDCOM together.

In any case, UNIGEDS is the work of one person and hopefully as a result does not suffer from ambivalence in the way that GEDCOM does, giving dev-users two ways to do things--the bad way and the worse way--and pleading with them to do it the bad way. When GEDCOM says a SOUR tag can be either a primary record or subordinate to a primary record, my response must be that GEDCOM is already fatally lacking in primary types, resulting in bloated primary records with all kinds of supposedly subordinate data crammed into them. I cannot, in good conscience, perpetuate the "SOURCE_CITATION" delusion, so it will be relegated to an exceptions log. Sorry for the inconvenience, but fixing GEDCOM is not my goal and I have to stay focused.

28. I hate to be a tattletale, but NOTE can be used so many ways in GEDCOM that it is likely to make my head spin if I try to deal with all of them. This may be why Genbox, for example, didn't take advantage of GEDCOM's ability to make notes a primary element. Every single line in GEDCOM is supposed to have 2 or 3 parts to it: num, tag and optional value. (For value to be optional is wrong, but that's dealt with in another rule.)

In a primary note record, we are expected to add the note text as a fourth element of the line. It is these arbitrary and unnecessary inconsistencies that make GEDCOM processing code so complicated. We have to rely more and more and more on the excessive use of conditional statements which are considerably more complicated than this makes it look:

```
if len(line) == 2:
    make_a_note_record_like_this()
elif len(line) == 3:
    make_a_note_record_like_that()
elif len(line) == 4:
    make_a_note_record_some_other_way()
```

Since lines don't contain the same number of items, every line has to be checked for length, or you can do as I did

and give every line an extra null item so line-length checking doesn't have to be done. And two or three ways to record a note in GEDCOM? Why? Why? Why? Every added stage of checking conditions makes the code harder to finish, and makes the final code harder to read, understand, maintain, extend, or improve.

Experienced GEDCOM import program writers will be objecting, "Yeah, but all you do is leave the fourth item blank and start the note value in a subordinate CONC line." Yep, this works great, so you don't have to test for 4-item lines. But no... in reality, we do not write the GEDCOM files we are trying to import. What if some other writer of GEDCOM actually follows the specs by putting the note text in the zero line as a 4th item? We actually do have to check for a 4th line item (or provide a blank 4th item, or some other sub-optimal hack), at least if the line is a primary note line.

Sure, not many programs use primary notes. But--and this is a big but--all genieware *should* use primary notes. You give a note an ID so it can be used over and over. You write more code, you do more thinking, you solve more problems, and when you're done, your genieware is easier to use than Brand X because notes that already exist and apply to more than one element (which are generally copy/pasted from element to element) don't have to to be copy/pasted anymore.

This dramatically reduces the time it takes to read an import file. What all this means to me, as I sit here a few hours or days away from being able to test the first testably-close-to-being-usable GEDCOM import parser I've ever written, is that I now realize the need to start over again. I must do what I've long known I should do, but could not convince myself of it. As one of the very first steps in parsing a GEDCOM file, the code has to literally rewrite the GEDCOM the way it should have been written to begin with. Anything else leads to complex, unreadable code.

Two things stand out here: every line in the import file has to contain the same number of items. And there has to be one way to do any particular thing.

29. I'm not picking on Gramps here. Although I seem to be allergic to Gramps' interface, I'm pretty sure that Gramps is trying harder than any genieware vendor on earth to get their inner data structure right. I'm not seeing the following travesty in Gramps' exported GEDCOM, but I've seen it in GEDCOM being produced today by a popular genieware vendor as well as a popular vendor of the past:

```
0 @I13@ INDI
...
1 BIRT
2 FAMC @F3@
...
1 FAMC @F3@
```

The above gobbledegook is not at all the fault of the vendors who write this strain of GEDCOM, including myself, since the specs state: "Biological parents can be shown by a FAMC pointer subordinate to the birth event(optional)."

But I am seeing Gramps' following curious attempt to record an adoption, and I'm still not picking on Gramps; this is pretty close to specs, but I don't know why the custom tags had to be added to say what the other tags already said. Note the repetition of two of the same tags in the same subrecord, three different ways of saying the person was adopted, and possibly Gramps' believing in the necessity of using custom tags to get the point across. It's no wonder there's not much genieware out there that handles alternative parents such as adoptive parents, foster parents, and guardians. The problem is not that GEDCOM is broken and can't be fixed; the problem is that anyone ever got the idea that it was a standard.

```
0 @I0625@ INDI
1 ADOP Y
2 FAMC @F0017@
3 ADOP HUSB
```

```
1 FAMC @F0017@
2 _FREL Adopted
2 _MREL birth
```

At the risk of being repetitive--not that it should matter by now, I guess--using ADOP twice to say two different things increases the complexity of the conditional tests that have to be performed in order to figure out what these lines are trying to say. The worst part is that using any given tag in different ways is so unnecessary.

I've read somewhere among the prolific writings of Mr. Tamura Jones, a GEDCOM savant whose writings are always worth a second reading, that we should not have so very many tags. Making blunt pronouncements that aren't backed up by reasons is part of his personality, he can't seem to help it. I can't imagine why he would say we don't want a lot of tags. I believe the opposite. In a computer program, if there are 1000 values to track, and you try doing it with 500 variables... well, what do I know, but it sounds scary.

How about human language? Here's a real sentence in the Visayan language: "Naa na na." Those are three different words, each pronounced differently. It means, "That's already there." In human language, we can consult context to clear up ambiguous language, or ask someone to speak up, repeat themselves, explain better, or take off their mask so we can try to read their facial expressions. But when computers communicate with each other, ambiguity is a mistake, not to mention a big no-no, and I imagine it should be considered a sign of some developer's not trying hard enough to express simple things simply.

Just to ice the cake, here's another specs-suggested way to make the patron devil of repetition-in-code jump up and down in ecstasy:

```
0 @I3@ INDI
1 FAMC @F9@
2 PEDI foster
1 ADOP
```

2 FAMC @F9@

In a belabored attempt to be fair, I'll admit that the specs might here be trying to show two ways to do the same thing, not actually requiring that both be done. However, why should there be two ways to do the same thing? Here's a simple, unambiguous way to do it all, not that we have time to do simple things, when we're so busy doing things the hard way:

```
0 @I3@ INDI
1 ADOP_COUPLE @F9@
2 ADOP_PARENT @I2@
2 ADOP_TYPE foster
```

30. This one is lower down on the list because it took so long to get even a tenuous grasp on those GEDCOM tags which should not be tags because they're event types. Actually it was a horrifying study of the RESI tag which generated this item.

You could get a masters degree in ways to use the RESI tag, a tag which should not have existed to begin with. It breaks several of the more important rules. RESI always means "residence", but that's the only thing it does right. It can be used as an event tag (although it's really an event type). In this case, it has a null third line element so we can futz around with subordinate lines to try and extract its value. RESI can also be used as an attribute, in which case its value is the third element on the line. As stated above, every line should have the same number of elements and each tag should have exactly one way of being used.

Types are values, not tags. The EVEN tag already exists, and it should always be used for every event, with another line (since this is GEDCOM) for the event type. Another feature that GEDCOM flaunts is this gigantic error in splitting hairs between what are events and what are attributes. To make matters worse, besides splitting a perfectly good hair, the two branches then are supposed to behave differently, for arbitrary reasons, which makes

splitters worry when someone doesn't know that events and attributes are not the same thing.

To make matters even worse, along comes RESI tag which can be used as either an event or an attribute. The addition of the FACT tag for attributes, so that the EVEN tag for events won't feel lonely, compounds the complexity. Let me say this about events and attributes: they are perfect candidates for lumping together into one happy category whose members could then be treated identically by simple code. But no, that's not how the standard-makers down at GEDCOM Central saw it.

There's Only One Version of GEDCOM

31. GEDCOM v. 5.5.1 is the version of GEDCOM that is most commonly taken seriously, and therefore the only one that should be supported during GEDCOM's coming death throes.

I've been taking quick peeks at GEDCOM v. 7.0.5 during a few fleeting moments of masochistic fascination, and it appears that the v. 7 specs delve into how custom tags should be properly used. They call the custom tag an "extension" to make it sound smartly technical, but is anyone fooled? Does anyone by now not realize deep in their heart of hearts that custom tags are a nuisance of infinite proportions? I hope not.

Caving in to custom tags by trying to regulate them is a symptom of being unwilling to admit that GEDCOM is broken and can't be fixed, that its basic premises are inadequate to the task of representing the world that genealogy must be able to represent completely and accurately. How long are we gonna keep walking on eggshells about this GEDCOM unstandard, just because we genealogists love the Mormons' fantabulous collections of data? I used to love going to the big family reunions that my mom's Mormon 6th cousin used to organize. I have never met a Mormon that I didn't like.

Now about GEDCOM... Someone's gotta speak up and I do mean yesterday. There's always a slight chance that if some loudmouth smarty-pants like me manages to break the ice for

the proper gentlefolk of genealogy, a proper ousting could ensue and that worn-out old tool GEDCOM could finally get itself a new home in a museum.

When GEDCOM Actually Gets Replaced

32. The difficulty in writing an import program for GEDCOM is directly proportional to the degree of discrepancy between the target app's data structure and the data structure of GEDCOM. This fact is stifling the whole genieware industry by encouraging vendors and wanna-be vendors to avoid creating a better data structure than the one which matches GEDCOM's assumptions and mis-assumptions.

The obvious solution is for every genieware vendor to store data in the same data structure. A few clever vendors have instead gone to the trouble of figuring out how to use GEDCOM itself as a database. When GEDCOM inevitably gets tossed out, these databaseless vendors won't have anything extra to throw out, just their whole back end. I was gonna say they'd have it the worst, but not really. Everyone will feel the pain about the same.

33. I suspect that if there are 45 wanna-be future vendors of genieware out there, and 44 of them are following the development of UNIGEDS because it's in the public domain and they can use it any way they want, 43 of those are likely to use UNIGEDS instead of developing their own custom data structure. If any of this fantasy comes true, most existing genieware vendors are gonna be left behind in the dust. The obvious solution is for existing genieware vendors to follow the development of UNIGEDS too, so they can keep up with the Joneses when they find themselves not the Joneses anymore.

34. I hate to say it, but the most realistic scenario for genuinely replacing GEDCOM has genieware vendor A developing a new product that uses UNIGEDS as its back-end, vendor B developing a new product that uses UNIGEDS, vendor C... you get the picture.

The existing products could be abandoned and left right where they're at, and the new products could never touch GEDCOM. Believe it or not, we are resilient, hard working genealogists, and we would get through this transition period. Over a period of time, every one of those vendors would have time to develop a means for importing their old product's trees to their new products. Of course version 1 of these new products should be free for anyone who owns the old products whose end-of-life should all be announced immediately.

With every vendor now using the same database structure, nothing like GEDCOM would ever be needed again.

My GUI Features: Not GEDCOM's Problem

35. ABBR is a short title for a source. UNIGEDS doesn't encourage abbreviations, but some users like to record the longest-possible title for a source, and that has encouraged some vendors to get a second, shorter title as well from the user, for use in tight spaces. Instead of this dual-name system wherein the user has to think of two names for everything, the user could input readably succinct name for a source as its name, and record the longer, more formal name for a source as a note. Example:

Put this in a note linked to the source:

Twelfth Census of the United States--Schedule No. 1--
Population (1900 United States Federal Census)

And input this as the source's title:

1900 U.S. Census

Or however you want to do it. Treebard uses autofill fields for source names, so the user will only have to type the name once.

36. AGNC refers to the organization responsible for a source. Like PUBL and AUTH, this is an attempt on the part

of the splitters responsible for GEDCOM to drive lumpers up a wall. The citation is already covered by PAGE. Every vendor can support "illegal" CONC and CONT tags subordinate to PAGE without GEDCOM's permission. Problem solved. GEDCOM v. 5.5.5 allows the use of CONC with any text field.

So what is the problem exactly? The problem is that this is a good example of the "just don't start (what you can never finish)" principle, a principle that GEDCOM has thoroughly ignored down through the ages. There are exactly 150 bazillion kajillion ways to write a citation, so it's up to the user how to do it, and it always will be. This is because anyone who's ever tried to write a genieware interface for input of citations has either offered the user an arbitrary number of vaguely-named fields that looks intimidating (who likes filling out long forms?), or else if it were done right, the citation text would be completely up to the user and he could type anything he wanted in a single field.

Also, of course, this field should autofill so he can re-use identical citations over and over; not something any vendor has tried as far as I know; Treebard's creator might be the first to bother making the citation a re-usable element with a primary ID. It wasn't that hard. It was a hassle, but once I strictly pinned down the cardinality of the actual relationships among the various data--source, citation, assertion--the code wasn't hard to write and the user interface practically teaches itself.

If you find a GEDCOM 5.5.1 file that actually uses the tags like AGNC, PUBL, and AUTH, send it to me. I need a new dart board. I have yet to deal with AGNC because handling a tag that no one uses doesn't sound like that much fun. And is it really GEDCOM's problem to tell us what parts should be in a citation? Citations should never be split into parts, because there's no way to define a citation universally for all users in all situations, so just don't start down that road.

37. Let me say this about a tag I've decided not to handle: INDI.EVEN.SOUR.DATE.DATE. The example below is from Gramps'

sample tree GEDCOM export file. The SOUR tag is subordinate to any event-type tag or to a NAME tag.

```
0 @I0044@ INDI
1 NAME Louis /Garner/
...
2 SOUR @S0002@
3 PAGE Page 11 2/3.
...
3 DATA
4 DATE 5 MAR 1990
4 TEXT On every third blue moon, Lewis...
```

Here we see a citation that was recorded in 1990, but Louis was born in 1855. The 1990 date could be meaningful to someone, but it is not a date assertion, it's the date the assertion was made in a citation about an event from 1855.

This level of detail is not unheard-of in genieware. Family Historian finds it convenient to include a field to record this data (though the purpose of the field where the date goes is not clear in the interface.) Since FH uses GEDCOM as its only data storage facility, FH can fall into the trap of designing an interface to serve the interests of GEDCOM instead of the interests of genealogy and genealogists.

Gramps also includes the use of this tag in its export .ged, as shown above. Gramps uses a real database, so I assume this field is on their interface somewhere, if I could only but find it. Just kidding, I'm opening Gramps right now to see where this field might be. I'll count the minutes till I learn the truth... starting now... OK, well never mind, it took 5 minutes to learn that the input for this place is either well hidden or doesn't exist.

The real issue here is that GEDCOM is trying to help us design our user interface. GEDCOM should stick to transferring primary historical data, which is plenty complex enough, and if a genieware designer wants to add too much detail, let them do it on their own time.

Here's some GEDCOM that no one uses. Why not?

```
0 @I1@ INDI
1 NAME Thomas Jacob /Black/
1 ALIA @I2@
```

The specs state that the purpose of this tag is to say that person ID 1 and person ID 2 might refer to the same person. Someone's bright idea didn't get picked up by the genieware vendors. Why not?

Instead of doing this, the vendors use the ALIA tag anti-specs-wise to link alias names, AKAs. This is supposed to be done by means of multiple NAME tags. But the tag is not supposed to have anything to do with names (I won't mention that the tag should therefore not have been spelled "ALIA".)

Here's why someone's bright idea never gets used. 1) It's none of GEDCOM's business, and 2) no genieware vendor (to my limited knowledge) has a way of marking two people as maybe the same person, to be potentially merged. Why not? Unless your GUI shows two people at once (which is confusing, so maybe a bad idea), the marks would be meaningless, and ignored.

There's a right way to do this, and it's still none of GEDCOM's business: a Projects feature or a To Do feature could have an item of research: "Figure out whether Janet Similar #42 and Janet F. Similar #189 are the same person." GEDCOM wastes our time with tags that are not well thought-out, because we are detail-oriented, so we fret over goofy edge cases that we should just ignore. If they weren't calling their specs document a "standard", such mistakes would just be questionable suggestions. But if you're gonna call your document a standard, it should stick to primary data and let each vendor do what he wants with secondary features.

Why do I keep saying this is none of GEDCOM's business? GEDCOM should be about basic genealogical data, and since

they can't get the basics right, they have no business trying to write our GUI features for us.

38. The creators of GEDCOM lived in an ancient epoch when there were not a whole bunch of genealogy database software programs competing with each other. Back then, there were only a few such programs, none destined to last as long as GEDCOM, and the most popular such program was created by the spiritual brothers of the creators of GEDCOM, and only shortly before GEDCOM itself was created.

The creators of the first GEDCOM version, and all their successors, have been peculiarly myopic in one area: they have not been able to tell the difference between what the GUI designer needs to decide for himself, and what the GEDCOM designer needs to present as a standard of data storage. Unbeknownst to GEDCOM's creators, some things are none of their concern.

A lot of careful thought has been easily avoidable over the years since the developers of genieware felt they only had to follow the so-called standard (which GEDCOM claims to be), instead of thinking original thoughts about how to correctly and accurately portray the world with a data structure. Entire interfaces were built around just simply satisfying the arbitrary decisions made for them by GEDCOM rules.

Take names and ID numbers such as SSN, for example. UNIGEDS stores all person labels the same way. There's no reason to not lump them all together in one category. The reasons to lump them all together as person identifiers outweigh any rationalization for splitting them into separate categories and then treating them pretty much the same after all that trouble.

What Users Don't Know... Won't Hurt Them

39. In the process of trying to figure out how to get GEDCOM-preserved data to become useful somehow, something else was revealed to me. It's the First Law of Dealing with

GEDCOM, and you may recognize its official version: "What they don't know won't hurt them."

Genieware developers over the 40 years of GEDCOM's existence have, oddly enough, not been particularly concerned with helping their customers leave for better genieware. For some reason, getting rid of their customers has not been their top priority. This is certainly not a conspiracy, but it might be categorized as a human tendency, especially when money is involved. You know, that fickle human desire to strive to keep the wolf from the door. I mean, if you really did want to get rid of all your customers fast, using GEDCOM wrongly might not be the quickest way to get that done. Who's even gonna notice if you don't have the time, ability or patience to leap gracefully over every single one of GEDCOM's hurdles?

Of course there are some over-the-top hard-core data buffs, the kind of folks who'd be busy memorizing the phone book or counting toothpicks if not for this wonderful genealogy hobby that requires us to leave no stone unturned, who might be going over the 60,000 individuals and their hundreds of thousands of exciting events and attributes after a tree has been created by importing a GEDCOM file, and picking out all the mistakes and inconsistencies. But as for regular ordinary folks like you and I, we look at the slop trees that GEDCOM churns out and shrug it off as, "Oh well, it's just GEDCOM, what did I expect?" and spend the next month or year redoing much of our work in the new program.

Or like I did: find a new hobby (writing my own genieware). I could annoy you with any number of examples of GEDCOM-gone-wrong when used out in the real world by actual genieware developers, but here's the main point: vendors are creating something for the love of creating something, and they hope their customers will also love what they are creating, because they get inspired by the thought that someone might appreciate their work and maybe even pay for it. They have their own goals and motivations which don't involve GEDCOM; not voluntarily anyway. When they bump up

against GEDCOM's self-perpetuating web of inadequacies, sometimes the tendency is to Draw the Line Somewhere, ask themselves who the heck's gonna notice anyway, and just not deal with every single problem that one can expect to encounter when using an antique tool.

One Task per Tag

40. All Elucidom tags are uniquely named according to their relevant superior (parent) tag. A REPO_NAME tag is only used for repository names. A SUBM_NAME is only used for the submitter name. A big reason for this unique-tag-naming rule is that it will reduce the amount of conditional testing that the code needs to slog through in subsequent steps where code complication can go from zero to 60 in ten skips of a heartbeat:

```
if tag == "NAME":
    if supertag == "INDI":
        link_name_to_person()
    elif supertag == "SUBM":
        link_name_to_contact()
    elif supertag == "REPO":
        link_name_to_repository()
```

41. Because of GEDCOM's unnecessary habit of using a single tag such as "TYPE" or "NAME" for multiple different purposes, if you plan to write a program to import GEDCOM files to your data structure, one of the first steps should be to translate every single tag to one that has one single usage. Don't do this later, while doing other things; the result is a tangled, confused mess that you don't notice till you get close to the goal. This is just my opinion, but getting close isn't good enough, and the omni-directional tangle caused by ambiguous tags is what caused me to start over at least once. It seems I had ignored that little voice inside that said, "Translate the GEDCOM to something that's easy to read, and do it before you write any code to make sense of the values and their relationships to each other."

Woe to me, I thought it would be an overreaction to translate every tag from the git-go, and I didn't listen. Fortunately I enjoy a good mystery. And I like starting over, because then I get to re-experience the thrill of that first part of a coding project... you know, the part where everything seems to fall into place, and you know what to do, and it couldn't be easier. It makes you feel so dang intelligent. Unfortunately, GEDCOM's greatest talent is making programmers feel stupid.

Elucidom Means *the State of Making Clear*

42. Everyone knows by now that I think all the genieware vendors should adopt the same SQL data structure as the back-end or data storage structure for their app's front-end or user interface. Two main things could conceivably be wrong with that notion, which I call "UNIGEDS":

- 1) There could end up being a lack of interest--no participation--for any number of good or bad reasons.
- 2) My idea could be wrong, or just slightly off, for some reason I have yet to discover or haven't been made aware of.

Therefore I've also introduced a partial solution for wanna-be genieware vendors who hope to keep using GEDCOM but find themselves tearing their hair out trying to write a GEDCOM import program. "Elucidom" isn't an acronym, it means what it says: *the condition of making things clear*. Elucidom is a simple and easy first stage in the GEDCOM import program writing proces in which the developer first identifies which GEDCOM tags have more than one meaning or usage. These tags are translated into non-ambiguous tags with one meaning and one usage. The result is Elucidom, a gedcomoid with only a few more lines than GEDCOM, and many lines identical to the original .ged file, since some of GEDCOM's lines are fairly useful as is. Elucidom was one of my first ideas, and one of the last things I tried, but it should have been the first thing I tried. After two years

of dithering and delay, writing the code for Elucidom took six whole hours.

UNIGEDS means "Universal Genealogy Data Structure" for family tree transfer from app to app, written in SQL and used in common by all genieware vendors. I haven't studied other suggested replacements for GEDCOM; they can't seem to compete with our current irascible captor, GEDCOM itself. I assume this is because of the warm, fuzzy feeling one gets from looking at a text file or "gedcomoid" and relatively quickly coming to an understanding of what it intends to convey. It even looks like a fun puzzle to try and solve, say for example if you know a little Python.

The gedcomoid approach does have its appeal, akin to crossword puzzles or Sudoku since you can look at it and see the treasure hunt hiding within; it makes you want to figure it out. On the other hand, more advanced technology is apt to scare off anyone but a professional programmer.

Every part of this document can be read as either...

- ...an unending grind of worthless complaining, or:
- ...in support of replacing GEDCOM with a database, or:
- ...in support of replacing GEDCOM with a gedcomoid

whose traits could be compiled as a summary of suggestions stated herein.

Elucidom and gedMOM are not nicknames for each other, they are two different gedcomoids. The gedMOM type of text-file transfer, which has already been tested and certified capable by yours truly, is designed to match the structure of UNIGEDS. On the other hand, Elucidom is meant to be generic, adaptable to a variety of data structures, but lacking the mistakes and omissions of GEDCOM, as well as GEDCOM's tendency to try and solve its problems by complicating things even more.

One way to prevent the bloating-solution approach is to declare the tool finished-as-is and immutable. One way to encourage this would be to limit the scope of this transfer tool to being a transfer tool.

For example, one or more of the valueless GEDCOM lines like this...

2 MAP <null value>

...was created to assist one or more vendors in compiling lists of things++. This is not a correct goal for a data transfer tool. Its only reason to exist should be to transfer data.

[++This was in regards to 5.5.1's new, improved, more complicated OBJE.FILE.FORM.MEDI tag. From the specs:

Note: some systems may have output the following 5.5 structure. The new context above was introduced in order to allow a grouping of related multimedia files to a particular context.

To which I respond:

Dear GEDCOM: Allowing or forbidding things is not your concern. Your job is to structure a transfer of data in a way that matches the structure of relationships of data-to-data in the real world, so that your involvement in the transfer process will be inert. Thereby may the data be transferred without being changed.]

The Only Good GEDCOM Parser is a Custom GEDCOM Parser

43. If you hope you can google "GEDCOM parser" and find a program that will grab a GEDCOM text file, read it, and change it to the format that will input directly to your app's back-end, well then you're like I was, several incarnations of UNIGEDS ago.

UNIGEDS is my back-end, my data storage structure. It's unique, it might have some things in common with your genieware's data storage structure, but maybe not much, maybe not enough to shake a big stick at. Sure, a GEDCOM parser, like the Python GEDCOM parsers I sorta tried to decipher not so many long months ago, will translate a

GEDCOM into some other format. But that end format is not going to import directly to anything.

Here's what you have to do instead of wasting your energy trying to force-fit a magic bullet approach. You know your target data structure. You know GEDCOM's structure. In the process of trying-by-doing to translate GEDCOM to your target data structure, you'll learn more about both, and if you're dense like me, you'll start over plenty of times or else you'll never finish. It's not possible to analyze GEDCOM first and then build something symmetrical, beautiful, elegant, and clean.

Sure you need those sorts of goals, to a reasonable degree, but the part about "analyzing GEDCOM" is unrealistic beyond a certain point, because there's nothing particularly logical to analyze. Why waste time analyzing chaos? Maybe quantum physics would help. GEDCOM is a willy-nilly text file cobbled together with rubber bands and bubble gum into a purported specification over the course of 4 decades by nice, proper, well-meaning, competent folks doing their very best, who were nevertheless lying to themselves about the ultimate usefulness of the GEDCOM strategy.

What we have done by using GEDCOM and continuing to use it for 40 years is to work ourselves at great cost of our own labors into a dead-end, a blind alley. We have climbed halfway up Mt. Everest because we weren't listening to those who said it was not a very good idea. Now in order to survive, we must literally give up, cut our losses, and turn around, climbing back down Mt. Everest in an emergency retreat, pounded by a blizzard of threatening proportions, with no more self-delusions to bolster our unreasonable efforts, with just the survival instinct that we ignored while tunneling into this infinite rabbit hole. But the main point is that to translate from GEDCOM to your unique data structure, you have to do it directly, without any 3rd-party intermediary software, or you'll just be translating to your unique data structure anyway, from some mostly irrelevant parser's idea of what someone (not you)

might need *if* a generic approach could be taken to get from GEDCOM to a real database.

Of course the best, and in the long run easiest strategy would be for every genieware vendor to use the same data structure, and that's why UNIGEDS is open-source, free, public domain, and written in SQLite which is also open-source, free, and public domain (completely unlicensed).

Custom Tags: For Workaholics Only

44. There's an understandable but anti-productive tendency for GEDCOM users to try and improve GEDCOM with custom tags. Understandable because they've been hollering unheard for truly useful GEDCOM specs since the dawn of time, and anti-productive because if vendor A is taking his effort to gedcomize his data that-a-way, it's pretty much guaranteed that vendor B is gonna be taking his similarly-motivated effort in some completely other direction. The result is an explosion of efforts, by which I mean to say we are coming apart, we are moving away from each other, further with each such effort.

One unfortunate result of the industry exploding away from the official specs is a proliferation of custom tags, with each vendor using his custom tag to do the same thing in a different way. That's not one wrong thing, it's at least two: 1) using custom tags, and 2) using their respective custom tags differently to do the same thing.

Here is an example, with the GEDCOM lines from Ahnenblatt v. 3 and Family Historian v. 7. This pair of efforts is trying to solve the problem of GEDCOM's having relegated the place element to subordinate status instead of giving it an identifying primary key all its own. Such a thing is needed so that each unique place can be referenced with an ID number many times, instead of the place name and the place's other attributes being repeated over and over.

There are other serious problems with how genieware vendors are using places, which are discussed elsewhere in this

treatise. These vendors do correctly recognize the need for a primary place record so that a specific place can have its coordinates recorded with the correct cardinality; *place-to-coordinates* is a one-to-one relationship.

Here's what's being attempted. The GEDCOM files quoted are from the respective programs' sample trees:

Ahnenblatt:

```
0 @L7@ _LOC
1 NAME Bremen
1 MAP
2 LATI N53.079278
2 LONG E8.801667
...
1 MARR
2 PLAC Bremen
3 _LOC @L7@
3 MAP
4 LATI N53.079278
4 LONG E8.801667
```

Family Historian:

```
0 @P50@ _PLAC Farmborough, Somerset, England
1 MAP
2 LATI N51.342362
2 LONG W2.4875262
...
1 RESI
2 PLAC Farmborough, Somerset, England
```

Ahnenblatt correctly uses a pointer `\_LOC @L7@` like a foreign key, but then wrongly repeats the place name and coordinates, which were already stored in the primary record.

Family Historian wrongly uses no pointer to the primary key or identifier `0 @P50@ \_PLAC Farmborough, Somerset, England`. FH also repeats the place name that was already

recorded in the primary record. FH correctly does not repeat the coordinates, so what makes them think they should repeat the place name? FH also wrongly puts the place name on the zero-line of the primary record. Because GEDCOM itself allows a fourth item on one (only one) of their primary record types--`NOTE`--we have to check every line for a fourth item, and then FH can go ahead and add a fourth item to their custom tag since we have to check for a fourth item anyway, so why not?

When the leader (GEDCOM) is a screwball, it sets a low bar and the users of such a standard--not just Ahnenblatt and Family Historian--get to do anything they want. Importing a riot of custom tags becomes someone else's problem.

If the repetition of data in the GEDCOM was found to be necessary for some reason by these exporters of GEDCOM, I suspect that the real underlying reason why such a circumstance might exist would have something to do with the fact that all or most vendors haven't bothered to define places as what they actually are. "Rifle, Garfield County, Colorado, United States of America" is not a place. It is four places nested inside each other. Vendors haven't bothered with this, because finding a simple way to present the facts of life to the software user takes some design work, some trial and error, some doing it over, till you get it right. Who wants to bother with all that when your software product is already in version 17?

45. I'm only gonna say this once: relegate all custom tags (tags which start with an underscore, e.g. `\_DETAIL`) to the exceptions log. I can't really say that GEDCOM is useful without at least a few custom tags. But I can say that GEDCOM is therefore not useful, so I will allow this rant to remain, in spite of the fact that I've handled plenty of custom tags and then ripped the code out of my module for every single one of these tags, because handling custom tags gives the impression that GEDCOM is The Tool, when it is in fact The Antitool.

If you use a big old monkey wrench to pound a nail into a wall, it don't make the monkey wrench a goldarn hammer.

If GEDCOM had the ability to do its job, then no custom tags would be needed. I removed every drop of the code I'd written to handle custom tags because once you head down that road, you've unleashed a flood of "Oh, by the way, I don't suppose you'd mind doing this also..." while watching your code turn to garbage as your random act of kindness that only manages to make GEDCOM look good, when it's just a stinky old monkey wrench.

We have to stop playing GEDCOM's game, wherein the authors of the GEDCOM specs say, "Please don't use custom tags," and then provide us with a broken system wherein custom tags are necessary because there's no intention to fix the broken parts. Using custom tags only perpetuates the common myth that GEDCOM is usable and/or fixable. Exceptions logs are the correct place for data saved by GEDCOM-used-wrong or for data that GEDCOM couldn't handle because it wasn't up to the task.

Genieware users should contact their vendors and insist that GEDCOM has to be replaced with a modern tool such as a universal relational database structure like UNIGEDS. This new tool should not be based on GEDCOM, because GEDCOM is a thing of the past, it is obsolete. Principle 274 in elementary programming is, "To write bad code, make sure you don't start out fresh, but instead base your new code on your old broken code that barely worked and was unmaintainable, unreadable, and unextendable. In this way your past mistakes will be built into your new work. Good luck."

In the end, I didn't even handle the well-meaning attempt of a variety of vendors to non-redundantly import latitude and longitude coordinates for places by using custom place tags such as `_PLAC`, `_PLACE`, and `_LOC` in zero lines. I had already written and tested the code because I felt compelled to demonstrate that I could've handled custom

tags, if I weren't so sure that it's immoral, so that when I am accused of being too lazy to handle custom tags in my code, at least I can give myself a reassuring pat on the head, knowing that the I already did handle them, and managed to make myself stop.

What I am too lazy to do is this: I am too lazy to keep my mouth shut when GEDCOM's perpetrators try to bang nails into my trees with their rusty old monkey wrenches. As Wilbur Fred Rhubottom, my piano repair teacher, once said every day, "Use the right tool for the right job." GEDCOM is the right tool for the recycling bin. GEDCOM's system of subordinate line numbers is so inflexibly inept that the very notion of allowing dev-users to imagine their own tags into existence is obviously an act of desperation by a GEDCOM creation committee that can't figure out any other way to not mention that they had failed to create a data transfer tool.

For Wanna-Be Authors of GEDCOM Import Programs

46. Here's one suggested order of operations when importing GEDCOM. I don't know any way to do everything on one read-through, and in my experience, it isn't reading the file that is slow, but inputting the data to the database, so my import program reads the file several times, with limited goals each time.

Read all primary tags first, persons (INDI) before couples (FAM), so that you have a reference to persons to put into the couples collection.

Do places from within events since you'll need a reference to the current place to store in the events record.

Do events from within assertions since you'll need a reference to the current event to store in the assertions record

Do sources, citations (PAGE) and assertions (TEXT) all at once. The assertion record stores a reference to the citation, so do citations from within assertions. The

citation record stores a reference to the source, so do sources from within assertions. Here are some instructions I wrote for myself when creating the final version of `gedcom_import.py`, which stores values in Python dictionaries:

EVEN, FACT, BIRT, etc: set `self.current_event_id` by incrementing `self.max_event_id`.

SOUR pointer: set `self.current_source_id`.

PAGE: create a citation ID for the current (new) PAGE value or get the ID you'd already created for the current (duplicate) PAGE value, and save it as `self.current_citation_id`.

TEXT: create an assertion ID (every assertion gets one, even if identical text to another assertion). This allows evidence for a conclusion to build up in favor of assertions repeated from source to source. Because UNIGEDS requires an assertion record to link citations to events, a blank assertion had to be created for PAGE lines (citations) where the GEDCOM had no linked TEXT. The Treebard user will then have a cardinality-correct database where he can fill in the real assertions via the interface at any time by looking up the citation to see what the source actually says.

Currently I'm recording TEXT values as either a name assertion, or regarding events/attributes, a date assertion or a particulars assertion. The user can edit that if it's something else.

Names can also be linked to sources, so the code I wrote works about the same for names.

The final dictionary might look something like the one below. There are good reasons for its complexity; citation-to-assertion is a one-to many relationship, so the assertions value in the dict has to be a list, with each assertion appended to it. Similarly, source-to-citation is a one-to-many relationship, so the

citations value in the dict has to be a list, with each citation appended to it. The `events` dictionary in my import program is global, but could be `self.events` if you prefer. I put my final values in global dictionaries so that various classes could add values to it. The Python `global` keyword is not needed to add values to a global dictionary, I guess because this is not considered a modification of the dictionary. Don't ask me, I just work here.

```
events = {
    self.current_event_id: {
        "EVNT_TYPE_FK": event_type_id,
        "CUPL_FK": couple_id,
        "PLACE_NEST_FK": string_of_place_ids,
        "EVNT_AGE1": husb_age, "EVNT_AGE2": wife_age,
        "ENVNT_AGE": person_age,
        "EVNT_DATE": date, "EVNT_DETL" particulars}}
    self.sources = {
        source_id: {
            "SORC_TITL": source_title,
            "SORC_PUBN": publication,
            "SORC_ATHR": author,
            "citations": [
                {citation_id: {
                    "CTTN_STRG": citation_text,
                    "assertions": [
                        {assertion_id: {
                            "ASRTN_STRG": assertion_text,
                            "ASRTN_TYPE": "particulars"}}}}]]}}}
```

The ability to know the event ID when recording assertions is experimental, based on deleting the DATA tag line, which changes this GEDCOM, which is legal but missing a link from PAGE to TEXT...

```
1 BIRT
2 SOUR @S89@
3 PAGE chapter 17, page 110, line 42
3 DATA
4 TEXT Born in India en route to midwife clinic.
```

... into this, which could be better by specifying an assertion type (name, date, place, particulars, age, or role) but at least it makes sense:

```
1 BIRT
2 SOUR @S89@
3 PAGE chapter 17, page 110, line 42
4 TEXT Born in India en route to midwife clinic.
```

This is experimental because the exporter of a GEDCOM could legally do this...

```
1 BIRT
2 SOUR @S89@
3 DATA
4 TEXT Born in India en route to midwife clinic.
3 PAGE chapter 17, page 110, line 42
```

...in which case, deleting the DATA line will be worse than useless. I think the experiment is worth the risk because:

- 1) Hardly any vendor outputs TEXT lines anyway.
- 2) Those few vendors who output TEXT lines seem to be doing it in the "right" order even though the specs don't care.
- 3) If the DATA lines aren't in the right order for this gambit to work, the TEXT line won't be linked to the right line. More code needs to be written to handle this case if it occurs, for example if the TEXT line isn't linked to a DATA line, it could be sent to the exceptions log.

Genieware vendors apparently have enough experience with GEDCOM to know that positioning the DATA line below the PAGE line makes it possible to link a TEXT line to the current event, as the specs should have done it. Unwritten Rule: *GEDCOM export programs have to write PAGE lines above DATA.TEXT lines, even though the specs don't specify it, in order for the TEXT line to become correctly subordinate to the PAGE line by the importing program's informally informed but not specs-prescribed act of deleting the meaningless and superfluous DATA line which the specs incorrectly link to the source. Import programs that delete the DATA line also have to catch any exceptions that this*

might incur if the GEDCOM exporter had not followed this unwritten rule.

47. Having started over on a GEDCOM import program probably two dozen times, I feel that I'm uniquely qualified to have opinions about GEDCOM. But not all my opinions are negative. I don't *only* play the critic; I've also done the work. My import program is nearly finished for the ~25th(?) time, and here are my notes for how I will proceed with the detail work, now that I've got some idea of what the broad strokes must entail in order for me to create a finishable import app. Which I've theoretically done.

Theoretically, all that's left is to fill in a small, medium or large procedure to handle each GEDCOM tag, depending on the tag. The procedures have already been written in various unfinished versions, but what stopped me from finishing the last version was that I had tried to glue together the broad strokes, the framework of the machine, from various previous versions, but then I had tried to fill in the details before I understood how the broad strokes were working together, if at all. The current framework for the app was created more carefully, the broad strokes were put together simply and are well understood, so here are some of the key concepts I plan to employ in finishing this version, if possible.

If not reasonably possible, I'll probably start over again.

The INDI record is the beast, because it contains many sub-records. It occurred to me recently that an inspection of the GEDCOM 5.5.1 specs reveals that there are a finite number of sub-record types that can occur within an INDI record. Each of these sub-records is signalled by the occurrence of one of these lines (not counting lines I intend to ignore; my app is open source so more features can be added if you care about the data I intend to ignore):

1 NAME ...

```
1 EVEN/FACT/BIRT/DEAT/RESI/etc. ...*
1 NOTE ...
1 SEX M/F/U
1 FAMC ...
1 FAMS ...
1 OBJE ...
1 SOUR ...
```

Below each main sub-record beginning line (a line that starts with a "1"), there can be lines that start with a "2", "3", "4" etc. These lines fall within the current sub-record.

The most complex sub-record is the event sub-record. The event has to be assigned an ID, and several elements within the sub-record need to be assigned IDs. By "ID" I mean a primary key as SQL calls it, which is basically the identifier that GEDCOM would have assigned to the element itself if GEDCOM were designed to match the requirements of SQL.

Entering the event sub-record, assign a current event ID by incrementing the maximum existing event ID. This ID is valid until the end of the sub-record is detected by the next occurrence of a line starting with a "1".

Within the event sub-record, its possible sub-sub-records are signalled by any of these lines:

```
2 TYPE ...
2 PLAC ...
2 AGE ...
2 DATE ...
2 CAUS ...
2 NOTE ...
2 SOUR ...
2 OBJE ...
2 FAMC ...
```

The multi-faceted ineptness of the GEDCOM PLAC specification requires a variety of hurdles to be jumped in order to get GEDCOM places into a UNIGEDS database. A place

ID has to be created for each single place within the GEDCOM nested place string. A place name ID has to be created for each place name used in the nested place string. And in case an app like Treebard GPS comes along which plans to provide an autofill field for nested place strings, a nested place string ID has to be created.

To do all this without introducing erroneous assumptions--such as wrongly split places that are the same place but called differently at different times, or wrongly lumped-together places that have the same name but are not the same place--is a hurdle that can only be jumped by either 1) writing so-called smart software to guess wrongly most of the time, or 2) stop the import process, open a dialog, and ask the user to sift through all the single place names. This has to be made easy for the user, or the user will cancel the import process. This was one of the broad strokes, and to make it work without messing up the code, it had to be done separately. Mixing this sort of complexity into the final detail work has stopped more than one attempt to write an import program. It should be done separately, and I ended up writing a separate Python class with its own Tkinter GUI to get it done.

The FAMC bag of worms is one I prefer to not discuss today, as it is particularly complex. I'll try to describe the right way to deal with FAMC/FAMS/HUSB/WIFE/CHIL soon, when I'm sitting in the pot of gold at the end of the rainbow with a successful procedure to show off.

One of the broad strokes is to read through the whole .ged file and record all the primary identifiers first, before starting over and reading the non-zero lines. As a text file, GEDCOM's left hand doesn't know what its right hand is doing, so I have never seen any simple solution other than to save the primary IDs first. So, for example, the family ID will already be saved when you come across a reference to it in a later read-through.

Another broad stroke I like to do first is to handle concatenations, keyed to the line number where the concatenated text began. Because CONC and CONT lines work like nothing else in GEDCOM, it lightens up the code-reading load considerably to not mix their handling in with the rest of the code.

As for handling the sub-records of the event sub-record, it's a complex hurdle to jump because it contains so many elements that need an ID, and the relationship of these elements to each other--their cardinality--has to be understood in order to store them in a way that the data will just slide into UNIGEDS effortlessly without jumping more hurdles.

But the main point of this whole item is simple: within a sub-record, assign an ID where needed and then (Python-wise), save that ID as an instance variable such as `self.current_event_id`, `self.current_citation_id`, `self.current_source_id`, etc. Why? That's the whole point: so you will have access to that value in the next line, and the next, until that sub-record is ended by the beginning of a new sub-record.

If I try to go into much more detail here, I'll use my fresh morning energy to blather and dither and froth at the mouth instead of writing the code itself.

It's going well, here at Treebard University, but I had all the windows removed in order to minimize damage when I throw furniture through them.

Tags You Should Use Wrong

48. In general, the best way to treat GEDCOM is by the book. For example, I don't think it helps the cause--which is preserving *genealogy*, not preserving *GEDCOM*--to resort to custom tags just because the specs say you can. But what I'm about to suggest is contrary to that by-the-book approach.

The CONC and CONT tags are allowed by the specs to only be used with certain tags, not including the PAGE tag as well as other tags whose values could easily exceed 255 characters. The only way to proceed is to allow incoming GEDCOM files to use CONC/CONT tags subordinately to *any* other tag except identifiers (zero-lines; primary keys) and pointers (foreign key lines). Legacy does this with the PAGE (citation) tag, for example, and it's the only way to do justice to the user's data.

I don't know where GEDCOM comes off thinking that a citation should be less than 255 characters long; it's not GEDCOM's decision. In GEDCOM's defense, an apologist might say that GEDCOM was just making a suggestion. Of course the specs were just making a suggestion. Nobody ever *planned* to make GEDCOM perfect, right now, from the very start, and that's why the specs are not specs at all. You know what's worse than GEDCOM? *Versions* of GEDCOM. Save us from a "standard" that was created to be changed.

The tool we need is not like a piece of software, to be versioned and reversioned. It needs to actually be right and complete the first time it's used, and that's what everybody is depending on it do be. As impossible as it is to reach this goal 100%, that still must be the goal: to treat the data transfer tool as an absolute, not as software to be reversioned later when we change our mind about how data are related to each other in the real world.

What we don't need is a bigger and bigger goal. I didn't say "write the most technical, complicated, bloated set of rules imaginable". When I say *make it right*, I mean make it *simple, obvious, and primary*. Leave the gray areas for the vendor to decide.

Don't Group Tags in Categories--Usually

49. Now that my new hobby is to start over on my GEDCOM import process every single day (because it's so much fun, why else?) I have a suggestion. OK, so it's a rule. No, it's a principle. Whatever. Let's call it "GEDCOM's dirty little secret". This is for developers and other wanna-be

writers of GEDCOM reading programs. I'll let you in on this secret but don't spread it around.

Here goes: the secret to GEDCOM success is to know and realize that most GEDCOM tags need to be treated as if they were really special and precious, by being given their very own function for handling only that tag. Most tags have to be treated special like this: If you find a tag "BOBO", then you need a function "do_bobo()". This clears out a lot of the well-meaning but tanglesome conditional code that would result if you were trying to be clever enough to group tags together into categories of tags so they could be teased apart again later.

There are tags which are even more special, which need to be quarantined like deadly viruses. And there are exceptions to this "don't categorize tags" rule such as event and attribute types which should not have been tags at all, which should be kept together in a collection like ("RESI", "OCCU", "BIRT", "DEAT", etc.) since they all get very similar treatment and can't rightfully be used as tags when they are actually types.

GEDCOM is Not About Relationships

50. Just when you think you've thought of everything, here come notes. Unlike the superficial design standards of most genieware, each note should have a unique ID so we can link any note to any number of elements and any number of elements to each note.

Cardinality-wise, that's a many-to-many relationship, which is not handled correctly by putting a foreign key in a primary table. Being a correct representation of the facts of life, getMOM must include something that GEDCOM had possibly never heard of: a place for many-to-many relationships to be recorded, not by some trick or work-around, but by being correctly represented as a many-to-many relationship. In UNIGEDS there are just such junction tables such as `notes\_links` and `media\_links`, where most of the columns are foreign keys. The plurals in the table

names clue the database user that these are many-to-many tables.

For each pairing of a note ID and another element ID, there will also be a note topic string and a topic order integer in the many-to-many record in UNIGEDS. Since users should not have to copy and paste identical notes to multiple elements, and we can't expect users to memorize or look up ID numbers, the topic or title given to each note help the user select the note he wants from a list of suggestions that pops up when he starts typing in the Treebard interface.

Families that Stay Together... are Weird

51. Sorry to bust your bubble, but there is no family event in family history. The closest we can come, while not compromising (fudging, lying, smudging, wishing) in regards to accurate record-keeping is couple events and marital events. Couple events include marital events, but not all couple events are marital events, such as "first kiss".

As for the mythical "family event" that was mentioned briefly in the GEDCOM specifications and subsequently used by genieware vendors down through the ages to confuse their customers, the GEDCOM specs in regards to real family events are actually couple events such as MARR (marriage) or DIV (divorce).

Then there are the other GEDCOM family events, the ones that are not actually family events in the real world, just in GEDCOM. RESI (residence) is one example of a non-couple, "whole family event" and this one is easily dispensed with by anyone who's tracked a family through the years.

Say there are 6 kids, although there's likely to be two or three times that many, depending on how many wives wore out and had to be replaced. In one document you'll have the two oldest kids, in another one, say a census ten years later, you'll have the three middle kids, then in a later document there's the one or two youngest kids. No problem, perfectly normal for the times, it's just that you can't define a

family unit as all them youngsters, all six or twelve or twenty-seven of them, and then pretend they had a "family event" of residing on some farm in New York State in 1915.

Sure there are family units, as pertains to the relationships. But including "residence" as a family event to try and make it look like there is such a thing as a family event, well it's downright dishonest of them boys down at GEDCOM Central, not to mention making everybody's life more difficult by affording us the golden opportunity to try like heck to track such an unwieldy, shape-shifting beast. There just plain and simple is no such thing as a family event in genealogical recordkeeping. It's not accurate enough to maintain the illusion that there is. There are couple events though. So we changed the GEDCOM FAM tag to the gedMOM CUPL tag.

52: Among the events which GEDCOM considers to be either family events or individual events are CENS (census), RESI (residence) and EVEN (event). As I keep saying, GEDCOM should not use any tag for more than one thing. An event type is either a couple event or it isn't. The EVEN tag is what should be used for all events, and there should be no other event or attribute tags such as MARR or RESI or OCCU. This is because the genieware user must be free to create his own event types. And since we can't guess at what will be added to the list of event types, we should not start guessing by making a limited set of tags for what we think we know. Types should be left open, used as values not keys, as in UNIGEDS where a table of event types gives each event type a unique ID.

CENS is not an event at all, it's a source. GEDCOM apparently thinks once a family unit is defined (which is really a couple and their children, i.e. a collection of individuals), then family events must apply to each individual in the family unit. This can seldom be assumed accurately, and since it's one of GEDCOM's worst mistakes, the import process should either send the supposed census event to an exceptions log (which would be the right thing to do) or else change the CENS tag to a RESI tag, which

would be the wrong thing to do because we honestly have to doubt very much that every child of that couple is really on the census every time the census taker comes around.

Children move away, unfortunately some of them die, and every once in a while one or more gets adopted out, runs away from home, or gets kidnapped or even abducted by a UFO, the leprecauns, or Bigfoot. In good conscience, UNIGEDS cannot support the notion of family events at all, so the census event will have to be manually re-entered as individual events for those family members who are actually on the census for that year.

The RESI tag is of the same ilk. Families don't live somewhere; individuals live somewhere in groups of varying membership. Sorry for the inconvenience, but we have to stop covering GEDCOM's tracks by pretending it is genealogy. Genealogy has to represent the real world, not some antiquated snapshot of it. I had been planning to create a user interface to ask the user about each GEDCOM family event by presenting the user with all the couple's children and asking which of the partners and children were actually mentioned by the source for that particular event. But this would not only interrupt the import process, it would force the user to look things up and do input when his intention had been to do import. So the only input I now ask the user to perform during the import process is to separate Paris, France from Paris, Texas and please tell me that USA and United States of America are the same place.

53. GEDCOM communicates by hints and insinuations, not by direct knowledge. GEDCOM whispers into one ear what your other ear needs to know. GEDCOM's motto is "Stuff it Anywhere", and the corollary of this motto is, "Figure it out." Instead of the .ged file telling us what we need to know, the specs tell us how we might be able to figure out what the .ged is trying to say, if the .ged has been written according to specs, but there's no guarantee of that, since .ged is just text and can't perform checks to enforce the specs' rules.

The rules are therefore treated as mere suggestions by vendors who understandably dislike and distrust GEDCOM after writing a GEDCOM import program, because the design of GEDCOM does not value the hard work and precious time of the developer who has to figure out what to do with data that has been stuffed anywhere. So when it comes to writing a GEDCOM export program, the developer has grown cynical, knowing that his app's carefully collected data will not be communicated directly by GEDCOM, but rather hinted at, contorted, dumbed down, and averaged out.

The reason for all this? Each line of GEDCOM is the center of its own universe, with few lines having a reference to other lines' values. The worst part is that the creators of GEDCOM knew how to grant access among lines--with identifiers and pointers--and arbitrarily stopped passing out identifiers after the magic number of 7 primary identifiers had been reached.

When a tool like SQL has long existed for the purpose of storing and transferring related data, the continued existence of GEDCOM in 2024 is an insult to the users who pay for genealogy software. They are being taken advantage of by genealogy insiders who do not tell them what they don't want to know. In true tech vernacular, all GEDCOM experts know that *GEDCOM sucks*. But that's not what we're told.

Instead, we get, "GEDCOM isn't perfect, but follow its rules exactly." Which no one will ever do, because people who like to write computer code are likely to consider themselves superior to the task of following nonsensical rules that obviously lead down a blind alley. With no one willing, understandably enough, to follow the rules exactly, we have everyone following the rules to his own disposition.

Here's an example of a nonsensical GEDCOM construct, the example which served as the fodder for the current rant.

```
0 @F1@ FAM
```

1 RESI shoe shop with apartment upstairs
2 DATE 1930
2 PLAC 144 East Main Street, Little Spring...
3 MAP
4 LATI 34.126476772597925
4 LONG -93.07273371608865
3 FORM address
2 HUSB
3 AGE 37
2 WIFE
3 AGE 25

Apparently out of what I might politely characterize as mental laziness, GEDCOM pretends that a residence event is a family event, as if this is any way to communicate who was living at a certain place at a certain time. All genealogists know, and this is not controversial, that the collection of actual family members found living at a certain place at a certain time varies wildly and unpredictably from from time to time in the real world. In fact, one of the reasons I started working on a genieware of my own was that recording the same date, time, place, and source for every family member who is found in the source for that event is tedious with existing software. It will always be tedious, but Treebard makes it less tedious.

In order to import the data from FAM.RESI, a GEDCOM importer with my aspirations of accuracy would have to create separate residence events for each partner in the couple @F1@, upon encountering the RESI tag. The user might have to delete one of these events, since there's no guarantee that both partners actually lived at that place at that time. The user will have to create separate residence events for each of the couple's children found living at that place at that time.

The worst part is that the usual method of knowing what the current event happens to be is useless in this scenario, so a whole new procedure has to be invented for just this construct. This is because two events are made at once, one for each partner, and then later, the ages of the partners

have to be recorded but only the last event ID that was created is now current.

This is certainly not impossible to get around, it's just another coding problem to solve with a little more complicated code. It's just an example of another reason why GEDCOM has to be replaced by a consistent system like SQL which functions according to a *single set of rules*.

Events & Attributes: Not One Thing; One Category

54. Trying to use the EVEN and FACT tags to delineate between events (hiring, retiring, won employee of the month) and attributes (career, position) is unnecessary, unproductive, ambiguous, and a downright pit of quicksand.

While it's true that events tend to come with a specific date and attributes tend to come with no date, or an optional or less important date, or a span of time, these are only tendencies and as such they cannot be relied upon to draw a reliable line between events and attributes.

A genealogist should not need a PhD in Hairsplitting in order to use genealogy software. To keep from churning up the mud, the obvious solution is to treat them identically in the database, and not try to separate them. Then the GUI creator can define his own criteria for separating them, or not, instead of an arbitrary line being drawn by some self-appointed standards committee. While it is important that a standards committee *not* mind its own business--if it's creating standards for nuclear power plants--this is a hobby, for crikey sakes, so the correct challenge for GEDCOM's replacement is to never draw a line arbitrarily, make no unnecessary rules, and take no position on delineations such as "event vs. attribute" which are inherently muddy.

Evidence: I just looked at all the sample tree GEDCOMs I've exported from the various programs I can use, and I couldn't find one usage of the FACT tag. How are these vendors recording attributes? A quick peek assures me that they aren't, unless there's a pre-fabricated tag such as

DSCR provided by the specs. If all events and attributes were chosen by the vendor (who could also allow the user to create more), then vendors would use their imagination (and copy from each other, like a good old blues song) instead of limiting their GUI's selection of attributes to what GEDCOM arbitrarily specifies.

Here's an example of muddy best left muddy. A RESI tag, according to the specs, can be treated as an event or as an attribute. This topic is muddy because an attribute implies a span of time: "He lived in Boston from 1845 to 1862". But many sources don't give a span of dates: "He lived in Boston on April 9, 1850". What is a GEDCOM to do? Nothing! Call them all "events" in the code and stop worrying about it. It is not GEDCOM's job to make decisions for us that should be made by the vendor and/or the user.

Here's a great example: the attribute "description" is clearly not an event but an attribute. At the age of 1, my dad had blond hair. At the age of 30, my dad had black hair. At the age of 90, my dad had gray hair. This attribute not only needs a date, but the date span that sometimes belongs to an attribute doesn't belong to this attribute. We would want the date that the description was made.

There's no universally correct way to separate events from attributes, in spite of the fact that they're two different things, and combining them simplifies everything while doing no harm.

While it's true that events and attributes are not the same thing exactly, this doesn't mean they shouldn't be lumped together. In UNIGEDS they are lumped together, not because they are the same thing, but because they should be lumped together. Trying to keep them separate promotes application bloat, will not be understood nor appreciated in the same way by different genieware creators, and serves no practical purpose. Family Historian lumps them together with its "Fact" element but then needlessly analyzes imported data and complains if an attribute is called an event.

For an example of pointlessly harping on the theme "events and attributes are not the same", see "Attributes do not have Age" in the 5.5.5 specs.

55. As an example of an event that is part attribute, residence does work with a date, like an event, "She was living in New York on April 7, 1877" and it also works with a span or range, like an attribute: "She lived in New York from 1875 to 1878" or "She lived in New York between 1857 and 1878". Splitting hairs as to whether a thing is an event or an attribute is anti-productive, a makework annoyance.

GEDCOM specs allow RESI to be used as an attribute or an event, which is an attempt at accuracy, but the problem is that the two categories should not be split at all, at least not by GEDCOM. It adds nothing for GEDCOM to try and standardize philosophy or to get involved in arguments about semantics; enforcing one's opinion about how the world is arranged is the business of the genieware designer, who needs useful generalities that are always true from an import file vs. irrelevant opinions built into the import file design.

Because UNIGEDS doesn't split events from attributes based on how events & attributes use or don't use dates, or on any other criteria, Treebard can do what it does to separate them or not, while some other genieware that uses UNIGEDS can do what it wants also, with neither genieware constrained by UNIGEDS. GEDCOM makes this harder, not easier, by its half-hearted attempt to separate the events and attributes... sometimes... while in fact throwing in the towel and making the developer decide anyway, but only after jumping through unnecessary GEDCOM hoops.

I'm hoping that this is not just a philosophical discussion or an argument over semantics. It boils down to the practical matter of providing genealogy software that tells a story vs. semi-useful database software that sorta works for genealogy if you put blinders on. There is too much

splitting of categories in genieware design, and because of it, no story is told about the subject of research. A shambles of data dumped into a bucket does not amount to the story of a human being. The GUI should facilitate the telling of a story through a picture made of words, and if the genieware user has to click from here to there and then over there and then try to remember what he was trying to accomplish, while jotting things down on a scrap of paper (remember paper?) because the data is scattered all throughout the program's various views, then a story has not been told. The user has been given some busywork to do and gets sore from clicking the mouse.

My notion of an interface that tells a story vs. a computer program that creates busywork for a wanna-be genealogist to do is to show the whole picture of a person's life in one view. Not a view broken up into a dozen scrunched-together scrolling and resizing opportunities, but an actual story. So the events vs. attributes argument is not about whether events and attributes are different things. It's about whether they should be shown in different views, or whether the user should have to resize columns to read them, or whether information should be truncated and abbreviated. The answer is no, no, no. The interface should tell a story by providing an attractive summary, not an interface that looks like the mother of all tax forms.

A Name is a Name

56. Person names in UNIGEDS are one string. A separate line is used to indicate sort order instead of the antiquated "Jeanne Marie /Middlebroth/" system of using slashes to indicate surname, as if everyone had a surname. (I suspect this system is an artifact of the age of the typewriter.) The first/middle/last name system is not usable from culture to culture and is unnecessary, as well as difficult to write code for. In gedMOM, sort order would be indicated thus, in the American style or the Dutch style, respectively:

```
NAME_STRG Dick Van Dyke
NAME_SORT Van Dyke, Dick
```

...or...

```
NAME_STRG Dick Van Dyke
NAME_SORT Dyke, Dick Van
```

The single-string name with a separate field for sorting order is important not only because the first- + middle- + last-name system is even more ethnocentric than using the typical American sorting order as a default. It's not about offending Icelanders. It's about Icelanders not being able to use the tool at all without ugly compromises to their cultural ways.

Due to my habit of jumping in with both feet and learning on the fly instead of studying a problem to death, it was late in the game before I learned that the NAME line such as `1 NAME John C. /Jones/` is expected to be supplemented redundantly with these lines:

```
2 GIVN John C.
2 SURN Jones
```

Some exporters of GEDCOM actually follow this silly rule, as if our time on this earth were not more precious than some silly rule. I can see using tags for nicknames, name prefixes and name suffixes, but GIVN and SURN should never be needed. If I'm wrong then OK, use them if you have to, but if you don't have to, then why should you? GEDCOM should be about genealogy, not about GEDCOM. When you're just a text file and therefore can't enforce your rules, you should not make any rule that isn't necessary. "Because I say so" is not good enough.

57. Breaking the full name up into parts and naming the parts... doesn't work. That's how we end up with weird categories like "forename" (first + middle) or "middle name(s)". But those aren't the real problems.

There are two real problems. 1) many cultures don't follow the first-middle-last routine for naming people, and 2)

writing computer code to deal with the first-middle-last custom is a waste of time because it's unnecessary.

A person has a full name, that's how the person is known. Save the full name as one string of characters. Ask the user which order the name goes in when alphabetized. Here's everything you need to know about my birth name, according to this system: "Donald Scott Robertson", "Robertson, Donald Scott". End of task.

If you think I'm making this up, well, no, I actually researched this. Ask a programmer. StackOverflow is your friend. The question has been asked many times and the consensus among programmers is overwhelming: do not break up names. I'm not talking about simplistic examples of SQL structure where online SQL gurus love to create pretend tables of names with a column for first name, a column for last name, etc. I'm talking about this: get on StackOverflow or any other forum where you can ask specific practical questions of experienced programmers, and ask their opinion. Actually this has already been done so many times, you can just google it.

The underlying concept here is that, in order to represent data accurately, if the structure of a certain type of data is subject to an open-ended number of systems, then the complexity of accurately representing that data is also open-ended. An open-ended degree of complexity is not a good thing to get caught up in. That's my fancy way of saying, "I wouldn't mind the work, the headache, the torment, the pain, except that the obvious goal of someday getting it just right will never be reached, and therefore I must conclude that representing the maximum possible complexity of the data is, as a practical matter, as impossible as it is unnecessary."

There's a simple solution to a problem of this nature, and that solution is: Simplicity Itself. When complexity blossoms into more complexity, when a project becomes a self-generating fractal of pain, the solution is to turn around and start over with the simplest representation of the data that is still correct and accurate. Leave

compulsive analysis--which means "to break stuff up into smaller and smaller categories for the fun of it"--to the software user, that's why the good Lord gave us thinking caps. It is not necessary for every element of genealogy to be broken up into smaller and smaller pieces. That's one of GEDCOM's mistakes. Tags that break names up into pieces are wrong.

Because GEDCOM is incapable of doing the task it was created to do, it will try to solve its problems by becoming more and more complicated. The specs say it right out loud: "Based on the dynamic nature or unknown compositions of naming conventions, it is difficult to provide more detailed name piece structure to handle every case." Which doesn't keep them from doing it halfway anyhow and then shrugging it off instead of backtracking and getting behind a simple system instead.

That's like Aunt Madge hollering on the way into the kitchen, "I really don't need this second package of chocolate-covered marshmallow Easter bunnies that I'm about to eat," as she makes a beeline for that selfsame package of bunny rabbits. We can keep pretending, like Aunt Madge pretends, that everyone in the world has a first, middle, and last name. Who's gonna notice? How many Icelanders are buying your software? Breaking names into parts is an impractical compulsion. A habit, not a practical necessity, not done in pursuit of any actual goal.

I also disagree with the detailed specs on addresses. Not only do the specs expect us to include GEDCOM lines for both the old way and the new way of doing addresses. The new way (which is the only way, according to the 5.5.5 specs) exists for mechanically addressing envelopes and such. This techy stuff is not genealogy, it is not my problem, so I've left addresses out of my import program for now. When I'm doing genealogy, if I find an address for a person in the tree, I save it as a place, and it works in the nested place structure the same as any other place.

Age is Simple, Let's Keep it That Way

58. The specs give us this technical-looking specification on AGE:

```
[ < | > | <NULL>]
[ YYy MMm DDDd | YYy | MMm | DDDd |
  YYy MMm | YYy DDDd | MMm DDDd |
  CHILD | INFANT | STILLBORN ]
]
```

Where:

> = greater than indicated age

< = less than indicated age

y = a label indicating years

m = a label indicating months

d = a label indicating days

YY = number of full years

MM = number of months

DDD = number of days

CHILD = age < 8 years

INFANT = age < 1 year

STILLBORN = died just prior, at, or near birth, 0 years

The right way for specs to handle age is this: "Age is text, any text the user and/or vendor desires."

It's not always GEDCOM's business to try and define standard ways for data to be written. The purpose of GEDCOM is to transfer data. Age is an open-ended string, GEDCOM has nothing to say about it.

Age is genealogy's least accurate data. Many uneducated people, like my ancestors for example, didn't know exactly how old they were. For this reason, age in genieware should be recorded as it was written in the source. It should not be calculated and filled in. It should not be standardized. If the source didn't mention age, it should be left blank

by the user. Applying standardized formatting to data that has a high probability of being inaccurate... kinda silly, dontcha think? Do record it, but don't change it.

These latter remarks are just my opinion. Am I going to try to impose my opinion as if it were a standard? No, I'm gonna build my opinions into my genieware product. My database product, UNIGEDS, doesn't care what you input as age. Any text, or blank, is OK.

There's Only One SUBM Record

59. SUBM is the least-used of GEDCOM's seven primary tags, so with no meaningful examples to look at, somewhere along the line I guessed that the purpose of this tag was to link a contact person or "submitter" of data to the data he submitted. It seemed like a reasonable guess since that's what the specs seem to indicate.

With no examples available of how any vendor actually uses this tag to accomplish anything, I went ahead and used it to populate UNIGEDS' contact table, manually adding some SUBM tags to the GEDCOM I was using as a test file while writing my import program. I spent many days working with the resulting structures, knowing that the tag is actually barely used, and knowing that my time was probably being wasted.

The ASSO.RELA comments that follow this one indicate a high level of frustration as this work went on, first while writing the import code itself, and later while actually trying to handle the structures while writing the exceptions log code. Near the end of this struggle, in desperation I tried looking into a document (GEDCOM 5.5.5 specs) a project spearheaded not by FamilySearch but by Tamura Jones, who is a highly opinionated and very knowledgeable GEDCOM savant, a deeply experienced genealogist, and a computer science professional. Here I learned that, from Mr. Jones' point of view, there should be exactly one SUBM tag in each GEDCOM file.

I guess that single SUBM tag would be for the person who submitted the GEDCOM, right? A pointer referencing the single SUBM record would probably be in the HEAD record, right? since the submitter of the GEDCOM is not necessarily a person in the family tree. As I explain elsewhere in this document, I completely ignore the HEAD file in my import program, partly since GEDCOM is a text file, not a computer program, so it cannot enforce the rules it makes, and partly since I have to limit my scope as a team of one. I've never had to worry about character sets, maybe because I only use Windows, but I'm not instructing anyone to ignore the HEAD record. Treebard is a showcase of primary genealogy functionalities, not a technical how-to on hardware issues.

60. I will not automatically import contact information to UNIGEDS, because there's a privacy issue at stake in the area of contacts. Much as I wish we could have a completely open exchange of information with no holds barred, spammers and stalkers feel the same way, for different reasons. For the sake of our contacts' privacy it might be wise to curb our enthusiasm when passing around contact information belonging to other people. In this way, genealogists are encouraged to do their own research vs. copying other peoples' trees. If researchers want to correspond with someone, let them get contact information from a source that the contact persons have made available themselves.

And while I'm on this topic, would someone tell the webmaster down at find-a-dead-person.com that posting people's email addresses as clickable links on your website has been out of fashion for many years, because it supports spamming.

REFN, AFN, RIN: The Prior Database's IDs are Obsolete

61. I'm probably about to make some unfair and uninformed remarks because I'm tired of handling GEDCOM tags instead of working on my own genieware project. You've been warned. I'll get back to this topic when I'm in the mood to deal with it properly, but here goes tonight's version.

While trying to research how vendors actually use the number tags such as REFN, I find that in spite of REFN appearing over and over in the specs, it is not about genealogy. It's about structuring the data in a way that's not going to be needed anymore, since the data is being exported out of the framework that used those numbers. The vendors whose sample tree exports I've seen are not providing examples of REFN. Forum posts on the topic of this tag are incredibly geeky but do not discuss how these numbers relate to information of a genealogical nature, why the numbers are important, why someone would want to keep them around. It's all about toying with the data, not about someone's great-great-grandfather's uncle working on a stage coach that got attacked by Indians on its way over the pass. It's not about how Aunt Ruth fostered hundreds of children in her home and lived to the age of 98. It's not about whether great-grandfather Constantine was from France or Switzerland.

User-defined reference numbers might date back to the days of the typewriter. These reference numbers seem to be a carryover of the original database system: the shoebox full of numbered index cards. I've seen genealogy books wherein each person was assigned a number. This is what SQL databases do, also. But exported reference numbers belong to the prior database, which may be a book with numbered people, a box full of index cards, a computer program, whatever.

The data is being exported, it's leaving the home of those numbers. Genealogy is about people, places, events, dates, hopefully true stories, that sort of thing. The new database has its own numbering systems. Trying to preserve the data-numbering system of a database that is no longer being used might appeal to someone even geekier than I. But it does not appeal to me. I'm not going to attempt handling these number tags right now because I need to research the need for it before I'll know what to do and why I'm bothering to do it. I need to get on with this project, work on the exceptions log, and get my blossoming Python

dictionary converted into a UNIGEDS database that I can open in Treebard GPS.

UNIGEDS might get around to handling prior data structures' numbering systems someday, but at this time I'm concerned with handling the elements of genealogy, so I'll be leaving the elements of yesterday's data-numbering systems on the far back burner, for now. I consider this a secondary feature, not really UNIGEDS material.

Much Ado about One Thing

62. There should be no bidirectional (i.e. redundant) linking of data in anything that even pretends to be a computer program. If such a thing is actually needed, then the thing that needs it is not, in fact, a computer program.

GEDCOM requires spouses and children to be linked with INDI pointers in the FAM record as well as FAM pointers in the INDI record, which has led many aspiring genieware developers to join ready-response teams specializing in containing nuclear meltdowns instead of trying to write GEDCOM import programs, because it's more sensible and seems like a more worthwhile way to waste one's precious time on this earth.

Recording any data twice is against everything that database normalization stands for. In computer code in general, repeated code leads to nightmarish complications. INDI.FAMC and INDI.BIRT.FAMC/INDI.ADOPT.FAMC can appear in the same record while CHIL appears in the FAM record and says the same thing. I'm waiting for someone to explain to me why this redundancy is allowed, much less required. "Don't repeat yourself" (DRY) is one of the most often-repeated notions in programming Q & A forums such as StackOverflow.

About those redundant FAM pointers. The GEDCOM 7 specs spell out the requirement to use bidirectional links somewhat more calmly, but still without a reason given. The 5.5.1 specs themselves **require (in bold print, so they must**

be serious) that FAMS and FAMC must be used in the INDI record in spite of the fact that CHIL, HUSB, WIFE are also supposed to be used in the FAM record. The reason given by the specs is not so much an explanation as a hint. It seems that if we want a real explanation, we have to figure it out ourselves. In fact, that should be the motto of this whole so-called standard: "Figure it out yourself."

As for the hint given by the specs in lieu of a reason, the redundant linking of couples and children is supposed to be on account of "pedigree navigation", whatever that is. Sounds technical, but I'm going to have to guess what that means based on my own experience.

GEDCOM is not a computer program comprised of binary data where the computer looks for needed values based on computerish things like binary-ness or digitality, whatever that is. GEDCOM is a text file. It has to be read linearly, one character at a time. This might explain why you generally see FAMS near the bottom of an INDI record, as well as why there is a FAMS tag for spouse instead of a FAMW for wife and a FAMH for husband. Does this arrangement secretly depend on the SEX tag's commonly appearing ahead of the FAMS tag in the same INDI record? I hope the creators of GEDCOM would not rely on secret gambits like this, but it's hard to say since the annoying requirement that double-definition of values be used deserves to be explained to the same extent that it is annoying and to the same extent that it is required. Instead of being hinted at.

So you're tootling along, saving GEDCOM data one line at a time, and you come to a pointer (foreign key) that refers to an identifier that hasn't been defined yet. What I'm trying to say is that the requirement for double-input of families and children might be a gambit to try and make the text file look like it knows what's going on. It's hard to say exactly, because these specs don't elucidate where needed, any more than they specify specifics specifically. With GEDCOM, it's all a big guessing game with the bad habit of blossoming into a bigger guessing game.

63. I thought I had FAMS/FAMC/CHIL/HUSB/WIFE licked, until I got to ADOP/PEDI. These tags look innocent enough, apart from having something to do with FAMS/FAMC/etc., which is in itself a red flag. It's gonna take me a few days to get through this. I can't just barely get it working and then run like hell and hope it works, because there's too much riding on it. It can't depend too much (or at all) on the SEX tag's value, because the gender of every person in a tree is not always known and/or recorded. I'm coming to the conclusion that the requirement to link INDI and FAM records bidirectionally is good advice, based on the fact that GEDCOM is some pretty deep quicksand and non-compliance can be a horrifying experience, even if the specs don't give reasons for their pronouncements.

For a long time I didn't want a family unit in UNIGEDS. The family is a compounding of the more basic person elements. I was able to glean family relationships from events. But until I recognized *family* (really *couple*) as an element of genealogy, I'd had to constantly fiddle around with a `kin type` element which cost me a lot of time and effort and some convoluted code I was happy to get rid of when I finally caved in and created a `couple` table in UNIGEDS. It's a simple many-to-many junction table in which foreign keys for person ID can appear as many times as necessary in a left column and/or a right column, constituting couples made up of individuals.

Gender has to be irrelevant, not because we seem to no longer be living in a world of exactly two genders, but for a simple, practical reason: there's no guarantee we will know the gender of the partners in a couple. The wife could be named Frank and the husband could be named Evelyn. Stranger things have happened.

The problem with the FAM record, and possibly the source of my confusion, is that it is not as primary an element as the other elements. A FAM record is comprised of INDI pointers. This turns out to be a problem since the GEDCOM is read one line at a time, and we don't have the FAM data

yet while reading the INDI data for the first time. Unless, as we are commanded to do, we plant it there on purpose in order to be properly bidirectional.

What finally convinced me to add a couple element to UNIGEDS, by demonstrating an actual need for a couple element, was realizing that unlike GEDCOM and all genieware, Treebard was up to that point unable to record an eventless couple relationship. In order to detect a couple, Treebard had to find couple events and/or children. The user couldn't just pronounce two persons to be partners, and this was a shortcoming. This is the point at which I had to rewrite Treebard's family display table, but I was sort of happy to do it in the name of adding an expected functionality to Treebard.

The import strategy I'm using is to read through the whole file more than once. Sifting through the data later will make more sense with the primary elements already saved as keys in Python dictionaries.

The more difficult parts of the GEDCOM are easier to tackle in complete isolation from other parts of the GEDCOM. And going through the whole file to get a few things filed properly and out of the way will mean that later on, another reading of related data will find a basis of well-organized structures for further additions to that group of data to be pigeon-holed in.

So, although I thought I already had this department licked until I got to ADOP and PEDI, it now seems as if the FAMC and FAMS are used in enough ways that it would be worth the trouble to rethink this part of the code, to store FAMC and FAMS data as much as possible as early as possible so that in the 2nd or 3rd read-through, the FAM (couples and their children) structure will already be in place and the details will have an obvious place to go.

I have no idea whether other GEDCOM warriors feel the need for more than a single read-through of the GEDCOM file. Since it's a linear text file whose parts can't communicate

with each other, I've never been able to imagine any other way to do it.

64. For various reasons already mentioned above, the FAM tag is an exception to the expectations set up by the other primary tags. But this is GEDCOM, it is to be expected, because everything is an exception. You get used to looking up, then you have to look down. There is no consistency. There are two ways to do the same thing, but each requires a different strategy.

There's no logical reason to do things two ways. Are we trying to please everybody or are we trying to transfer some data? Are we trying to set a standard, or cater to different thinking styles by a variety of members of the GEDCOM creation committees? The INDI.FAMC (child) tag gives the same information as the FAM.CHIL tag, its bi-directional partner. But the FAMS tag falls short of providing the same information as its partner the WIFE/HUSB tag. If the FAMS tag is actually needed for some reason, then it should be FAMW/FAMH.

Hey now, while I'm complaining about things I already complained about--because when I woke up this morning GEDCOM had not been a bad dream, it was still here, and I still have to deal with it, and this document is still what keeps me sane--I have to mention this cute little bug in the ointment:

```
0 @I560@ INDI
1 NAME Trevor /Tewksbury/
1 ADOP
2 FAMC @F321@
3 ADOP WIFE
```

"Bug" doesn't really do it justice. More like a plague of locusts on a biblical scale. That second ADOP tag has already cost me a day or two, although to be honest if I got more sleep two nights ago, I might have been able to think more clearly yesterday. The right way to accomplish what INDI.ADOP.FAMC.ADOP = WIFE tries to convey is to make a different family record that includes the wife and not

the husband who didn't adopt the child. I say this because there is no practical reasoning behind the existence of a *family* element. The element that is helpful is a *couple* element. When GEDCOM treats the family element as a family element, it is lying to us. There's no such thing as a family element because its membership changes so much over time that we can't say who the members are without putting on blinders and shutting out the annoying facts.

The right way to indicate adoption events is not by pretending there is a solid, definable, simple, physical thing--the family unit--and then adding complication every time the pure nuclear family arrangement experiences a hiccup in the real world. Because the real world consists of what happens to us "when we're busy making other plans" as John Lennon observed, GEDCOM should work like SQL, which does not store data hierarchically, but as a network of related pairs.

Here are some better ways to represent data, and I'll give the same example in both a nested Python dictionary and gedMOM. The nesting goes pretty deep on the dictionary. That is hierarchical, but in a way that is useful because it can accurately reflect real relationships among data. The gedMOM below it, which shows identical data, is structured to exactly mirror UNIGEDS' SQL relationships.

If GEDCOM woke up one day to find that the Blue Fairy had magically transformed it in its sleep to mirror the way SQL works, then the data would slide effortlessly into a SQL database and the structure of the gedcomoid would tell us how to construct the database tables. The purpose of gedMOM is to demonstrate how SQL databases work to anyone who studies it, without looking at any SQL. I've already tested a manually created gedMOM file, and it was converted to a UNIGEDS database so easily and perfectly that the experience inspired this whole effort to write GEDCOM import and export programs for UNIGEDS. I'd postponed that decision because I'd been afraid it would cost me a year of

my life, but it looks like it will only cost me half that much.

In the gedMOM example below, the troublesome FAM data of GEDCOM is tamed by representing all the elements of genealogy as elements, not as subordinates to a sometimes-imaginary hierarchy. The annoying INDI.ADOP.FAMC.ADOP web of duplicity is simply represented by creating the right number of couples, instead of saying that Jack and Jill are always a couple no matter what happens. Since adoption is essentially a couple event, but a child can be adopted by only one member of the couple, then a new couple_id is needed for those adoption events which do not involve both members of an existing couple. Why lie and say that couple 14 adopted person 92, then add a footnote to mention that, to be honest, only an individual actually did any such thing? That's not how computer data should be communicated.

nested Python dictionary:

```
self.couples = {
    3: {
        1: {
            "children": [
                {"child_id": 93, "event_type_id": 48},
                {"child_id": 38, "event_type_id": 1}],
        2: {
            "children": []}},
    8: {
        3: {
            "children": [
                {"child_id": 65, "event_type_id": 83},
                {"child_id": 89, "event_type_id": 1}],
        4: {
            "children": [
                {"child_id": 65, "event_type_id": 95},
                {"child_id": 89, "event_type_id": 1}]}}}
```

gedMOM text file:

```
PRSN 1
* *
```

PRSN 2
* *
PRSN 3
* *
PRSN 4
* *
PRSN 93
* *
PRSN 38
* *
PRSN 65
* *
PRSN 89
* *
EVNT 74
PRSN_FK 93
EVNT_TYPE_FK 48
CUPL_FK 95
* *
EVNT 82
PRSN_FK 38
EVNT_TYPE_FK 1
CUPL_FK 95
* *
EVNT 59
PRSN_FK 65
EVNT_TYPE_FK 83
CUPL_FK 22
* *
EVNT 78
PRSN_FK 89
EVNT_TYPE_FK 1
CUPL_FK 8
* *
EVNT 29
PRSN_FK 65
EVNT_TYPE_FK 95
CUPL_FK 33
* *
CUPL 3
PRSN1_FK 1

```

PRSN2_FK 2
* *
CUPL 8
PRSN1_FK 3
PRSN2_FK 4
* *
CUPL 95
PRSN1_FK 1
PRSN2_FK null
* *
CUPL 22
PRSN1_FK 3
PRSN2_FK null
* *
CUPL 33
PRSN1_FK null
PRSN2_FK 4
* *
EVNT_TYPE 1
EVNT_TYPE_STRG birth
* *
EVNT_TYPE 48
EVNT_TYPE_STRG adoption
* *
EVNT_TYPE 83
EVNT_TYPE_STRG guardianship
* *
EVNT_TYPE 95
EVNT_TYPE_STRG fosterage

```

65. This document was created spontaneously as a venting post (a rant) whose main purpose was to keep me semi-sane by giving me a Complaints Dept. to go to while trying to write a GEDCOM import program. Nothing in this document is intended to be the last word on anything, and not every mistake has been removed from it since it's sort of a journal, though not organized by the date anything was written. I removed the most wrong, the most ridiculous, and the most confusing, but tried to leave that spark of spontaneity, or sarcasm, or whatever you want to call it.

For example, just the other day I did finally manage to import information conveyed by the `1 FAMC...` lines which indicate who parents are but are not linked to a birth event. Because GEDCOM accomplishes its goals in a number of different ways, in order to write a GEDCOM import program that is finishable, I've had to step outside my *one set of rules* comfort zone.

No doubt professional programmers have to do this every day, due to the many interacting dependencies which I've managed to avoid for the most part by using Python/SQLite/Tkinter which is all packaged together for Windows users. I do this for a reason, not just laziness or the fact that I'm too old and impatient to deal with dependencies: as a team of one, I've chosen to make my work accessible to other novice- and hobby-level programmers and other teams of one. While you could say I'm spoiled by my hoverparent Python, you could also say that I'm making genieware programming accessible to genealogists who are not programmers but who suspect that they could write their own genieware.

A terrible and therefore seldom-used structure INDI.ADOP.FAMC.ADOP exists for recording adoptions. By not ignoring this usage of FAMC (when it is both subordinate and superior to ADOP tags), I can laboriously glean information about an adoption. However, this is also a bi-directional tag since the CHIL tag is still used in the FAM record for the adopted child, so I don't want to give the impression that I'm not horrified by any injunction by the specs to record the same data twice in two different places.

I'm going along with GEDCOM to demonstrate my sincerity, not to support its existence. I'm out to replace GEDCOM with a proper database structure that can serve as a universal genealogy data structure, such as UNIGEDS. I'm not concerned with whether the world accepts my UNIGEDS or adopts something else instead, for better or worse. My task is to create a prototype for a GEDCOM replacement, and then I'll retire from this preoccupation and use my creation in

my own genealogy hobby, possibly documenting how this goes by way of blogging and vlogging, but possibly never being heard from again.

Despite the intensity of my involvement with the creation of an example of how GEDCOM could be replaced, it's not my problem and that's not why I'm doing it. I'm mainly doing it because one day I decided that no genealogy software worth using actually exists, and therefore I set out to create my own genieware for my own use. This project exists because I've accumulated thousands of documents in my genealogy hobby and I have no place to file the data. I enjoy filing genealogical data in a database so much that I refuse to do it till I can do it right.

66. This rule is so obvious I almost forgot to write it down. The DRY principle ("don't repeat yourself") is one of the backbones of programming. It was named in 1999 but no doubt born much earlier. As principle #23 of programming, it surely must have existed by 1984 when GEDCOM rose from the primordial mud of prehistoric computing. In programming, it's rare that one is commanded to repeat a block of code, only occasionally tolerated for specific and limited purposes.

But repeating a value definition is never actually prescribed as far as I know, because anyone who's spent ten minutes on StackOverflow would be horrified if you tried to suggest it. It's all good and well when you first write the code, you can even tie a string around your finger as a reminder that when you change that definition, you have to change it in two places. But you will forget, and the resulting bug will take hours to find, if you're lucky.

Saying the same things twice in regards to the FAM tag is exactly what GEDCOM expects, and the result is quicksand. No programmer could take the GEDCOM specification seriously as to what we are expected to do with FAM/FAMS/FAMC/CHIL/HUSB/WIFE. If I'm wrong I will eat my grannie's family bible. In SQL programming specifically, inserting the same data to a database in more than one place is an infraction of the principle of normalization.

67. ``INDI.FAMC``: I could assume it's a birth event. But if there's no birth event in the GEDCOM, the event has to be created, which would be all right, since Treebard auto-creates a birth event for every person added to the tree anyway. However, the eventless FAMC could become `INDI.FAMC.PEDI` in the next line, in which case PEDI's value might indicate that the event was not a birth. For a tag that's rarely used and which provides nothing that can't be done another way, PEDI requires too much hurdle-jumping as well as requiring extra-sensory perception since we don't know what to do with FAMC when we don't know if PEDI is coming next or not.

These obstacles could be overcome for a well-designed and frequently-used tag, but PEDI is not that tag. I have a sample GEDCOM from Gramps which uses ``PEDI`` 1374 times, and ``PEDI birth`` the same 1374 times, corresponding exactly to 1374 ``BIRT`` events. This tag can't be ignored since it can indicate that a child is in a non-birth family, but it requires special handling. The person whose INDI record it occurs in has to be found in the events dictionary. Then it has to be determined whether or not that person already has a birth event. If not, a birth event can be auto-created for that person.

But the fact that this might be a fosterage event, and not a birth at all, is not the only problem. There's also no guarantee that the person's birth event, if any, occurs *before* the `INDI.FAMC` tag. I know, vendors tend to insert BIRT lines early in the INDI record and INDI.FAMC lines at the end of the INDI record. That's not much consolation, since these are conventions, not requirement.

Also, the FAMC can be used in two different ways not counting the fact that ADOP can both precede and follow FAMC. One result of this mish-mash of unsymmetrical pseudo-functionality is a lot of code for a tag--PEDI--that is rightly held in contempt by vendors who, understandably enough, won't use it.

But that's not all. GEDCOM requires the nonsensical INDI.FAMC to be used redundantly in the same record where the fairly useful INDI.BIRT.FAMC or INDI.ADOP.FAMC is optional. A slight indication of why this is required is named in the specs but then left undefined: "pedigree navigation". This could be a case of GEDCOM's trying to be more than a data transfer tool by minding the genieware creator's business for him.

Not only that, but the same information conveyed by FAMC in the INDI record is also conveyed by CHIL in the FAM record. This reminds me of a young high school Spanish teacher I had who was so unsure of herself that she completely changed her teaching strategy every week. Most of the students in the class learned how to say "Hello" in Spanish by the end of the semester. Most of them.

I know that there's a built-in redundancy in genealogy family elements. The birth event, for the person who is born (or the adoption/fosterage/guardianship event for the person who is adopted, fostered, or guardianized) requires the existence of an offspring/adopted-a-child/fostered-a-child/guardianized-a-child event for each of the participating foster/adoptive parents or guardians. This is not GEDCOM's problem. This is solved by the genieware developer doing what needs to be done with the primary birth/adoption/etc. data of the person born or adopted.

In the case of Treebard, which uses UNIGEDS to store its data, which never stores any data redundantly, when Treebard encounters in UNIGEDS a birth event for a person, it auto-creates an offspring pseudo-event for each parent, which has no separate event ID since the child was only born once, and the pseudo-event is used only for display purposes. When Treebard encounters in UNIGEDS an adoption or other alt-parentage event, it auto-creates an alternative parentage pseudo-event for each participating parent. GEDCOM does developers a disservice by arrogating their responsibilities, not only giving them nonsensical tag structures to deal with or ignore with the usual

misgivings, but those vendors whose interfaces are based on GEDCOM's structure are limiting the abilities of their app to the confused mess of omnidirectional glop that is GEDCOM.

For a simple-looking tag with a non-ambiguous meaning, PEDI is one of GEDCOM's worst features. The worst of the worst is that its supposed functionality is easily covered as shown at the bottom of this chapter. First, here are three child-of-family subrecords from the specs:

```
0 @I3@ INDI
1 BIRT
2 FAMC @F4@
```

The birth subrecord above is correct and useful, and a correct adoption subrecord can be made *symmetrically*:

```
0 @I3@ INDI
1 ADOP
2 FAMC @F9@
```

68. The PEDI subrecord below is correct per specs but seems useless to me based on how I designed UNIGEDS and/or Treebard because it's not linked to an event. This reflects a frequent error in GEDCOM's family-first orientation, an attitude which is often excused as a religious orientation, but is really an ordinary mistake. We can't walk on eggshells forever just because the creators of GEDCOM had a religious mindset. The delusion is that families are more primary than individuals.

This sample from the specs denotes an adoption where there is no adoption event. I'm not just picking at GEDCOM for not understanding reality as I do; it's a practical matter that transferring GEDCOM data to an event-based genieware becomes unnaturally complicated when an event that doesn't actually exist in the GEDCOM is assumed by some non-standard GEDCOM tag doing its thing in some alternative way for the utter heck of it:

```
0 @I3@ INDI
```

```
1 FAMC @9@
2 PEDI Adopted
```

The rationalization for using PEDI (which fortunately no one actually does, I hope), is that instead of denoting adoption, the tag's value could denote fosterage. But the way I interpret the specs, there's already a better way to do this:

```
0 @I3@ INDI
1 ADOP fostered | adopted | ward | {other}
2 FAMC @F9@
```

Unfortunately, when it comes to denoting a birth family in the INDI record, this is all that's actually required:

```
0 @I3@ INDI
1 BIRT
1 FAMC @F4@
```

But in fact that's more helpful than it really has to be, according to the specs, which do not prevent a data export from just doing this...

```
0 @I3@ INDI
1 FAMC @F4@
```

...or even this:

```
0 @I3@ INDI
1 FAMC @F4@
1 BIRT
```

This aggravated nonsense wants us to assume that the linked family is the birth family, but the undeniable nuts-and-bolts of the actual situation is that software, due to the fact that it can only do what we tell it to do, is not capable of making assumptions. GEDCOM, therefore, by virtue of its very nature and the assumptions that it makes, is not software, is therefore not a software tool, and is therefore not a transfer tool for genealogy data. It is only a placeholder for a real data transfer tool.

Here at Treebard University, we're not waiting for that tool to fall from the sky and land at our feet, and we're not waiting for a self-appointed committee of GEDCOM devotees to agree on GEDCOM's replacement. We're making the needed software tool all by ourselves and planning on using it every day once it is ready. Whether anyone else sees the value in it or not, well, my hobby is my hobby and your hobby is your hobby, and I'm not here to tell you how to do your hobby.

As for how I'm planning to use the god-awful PEDI tag in a GEDCOM import process, I'm planning to ignore Gramps' 1374 superfluous mentions that the child born also happened to be born. If the PEDI tag is used for adoption or fosterage, I'll mention in the exceptions log that an adoption or fosterage event for person X and couple Y needs to be made manually in Treebard once the import process is complete.

That's not the absolute best I can do, but it's more than GEDCOM deserves. You see, we are not supposed to do GEDCOM better than GEDCOM is. We are supposed to follow its rules exactly, which no one does, because that's all GEDCOM is: rules. Unenforceable rules in a world where the perfect enforcement software for recording and manipulating data relationships has existed even longer than GEDCOM, and that tool is SQL.

A Place is Really Three Things

69. I keep saying that genieware has no use for the jurisdiction category such as "county", "parish", "nation", "township" etc. But vendors use it so it needs to be imported and exported. I added a place_type table to UNIGEDS to take care of it, but Treebard won't use it for anything important. If you think it's important, well that's why we have many different geniewares. And that's why we need import and export.

70. Places have to be stored in three ways:

- 1) single places (primary key for "Dallas" and same primary key for "Dallas City"),

2) nested places (ordered list of place ID foreign keys for "Dallas, Texas, USA"), and

3) place names (single name strings with primary keys of their own paired with place ID foreign keys, so that "Paris, France" and "Paris, Texas" are stored and used separately).

71. Detecting duplicate place names and asking for user clarification as to whether the two places are the same... it's a task which most genieware either ignores or barely pays lip service to. While in the process of personally not also ignoring it myself, it just occurred to me why it's not only unnecessary and undesirable for genieware to come pre-loaded with places; it's absolutely wrong in the primal sense of the word "wrong": there's nothing to argue about.

In order to detect duplicate place names, handle nested places in the correct degree of detail, allow single places to have more than one parent (enclosing place), etc., a list of places as long as the earth is round would mean that in order to check for duplicates, a very large amount of data would have to be sifted through for each place name that is checked.

This sort of checking is a part of the import procedure for UNIGEDS to accept place data from GEDCOM, because GEDCOM gives us compound places with no unique identifiers at any level, so there are cases where the user will have to decide if two places are the same or not. In order to keep from having to ask the user too many questions during the import process, we have to write some of the sort of code that gives me nightmares. But we don't mind a little nightmare now and then. It's purposeful dumbing-down and putting on blinders that really bugs me.

In order to offer the unnecessary and undesirable feature of a genieware's coming pre-loaded with places, the creator of the genieware has every motivation to do as little as possible in terms of handling the real-world details of real places. You don't want to write even one line of code that has to interact with every place on Google Maps.

Meanwhile, most family trees actually use a very small percentage of the places that come built into the genieware, so as usual it's the genieware user that gets the double whammy: a bloated program full of irrelevant junk that handles relevant places pretty badly, just so everything we do from sun-up to sun-down can be linked at the hip to Google.

72. Regarding SUBM.ADDR, GEDCOM gives two ways to record addresses. The old way and the new way. It's permitted to use both, but if you use only one, it has to be the old way, which is "required". The reason given is "backward compatibility" which the creators of GEDCOM somehow mistake for my problem. The old way is SUBM.ADDR.CONT which is a nice, simple way to record a multi-line address. The new way has tags for each address line, amounting to a key:value system. The stated purpose of the new way is for indexing. I guess this means that the new way tries to be helpful for vendors who want to sort addresses by country or street address or city or zip code, etc. I can't argue with any of that, but in UNIGEDS, a contact address will be a block of text, like GEDCOM's old way, because if a vendor wants to be able to index addresses they can input their addresses as key:value pairs like this (and this is how GEDCOM should have fixed the problem instead of creating a new way to be used on top of the old way):

```
1 ADDR 1640 Stetson Dr
2 CONT (address:) Apt 2A
2 CONT (city:) Phelps
2 CONT (state:) Texas
2 CONT (zip code:) 11111
2 CONT (country:) USA
```

The advantage here is that the vendor and/or user can decide what the keys should be, and this would be nice for places that want to be a "province" instead of a "state", for example. For GEDCOM to try and decide what tags to use as keys here assumes that it's possible to define address labels that will mean something 100 years from now when

UNIGEDS plans to still be the universal genealogy database structure.

It seems to me that the ADDR tag in an INDI record is like the street addresses we can collect from city directories. These resources often provide evidence that wasn't available elsewhere. A street address is essentially a place name, and census records often provide them too. But if GEDCOM wanted to provide an address tag in the INDI tag, they should have provided a unique tag. An address is just another nest in a nested place, so it doesn't need its own tag. Here is the UNIGEDS way, as represented by gedMOM:

```
PLACE 57
PLACE_LTLD 22345723458 N
PLACE_LNGTD 3898G453459 W
* *
PLACE 45
* *
PLACE 92
* *
PLACE 33
* *
PLACE_NAME 923
PLACE_NAME_STRG 1455 East Jaguar Place
PLACE_FK 57
* *
PLACE_NAME 745
PLACE_NAME_STRG Dallas
PLACE_FK 45
* *
PLACE_NAME 334
PLACE_NAME_STRG Texas
PLACE_FK 92
* *
PLACE_NAME 221
PLACE_NAME_STRG United States of America
PLACE_FK 33
* *
PLACE_NEST 4573
PLACE_FK 923-745-334-221
```

In Treebark GPS, the first genieware on earth to use UNIGEDS as its back end, the nested place element is crucial so that autofill widgets can provide entire complex place strings which therefore only have to be typed once. The creator of UNIGEDS recognized early on in the creation process that each single place could have multiple enclosing places, or parent places in the nesting chain. Researchers whose hunt for the facts depends on knowing what a place was called and when it had that name and what its boundary was at that time, these researchers are out of luck with existing genieware. Brand X might care if a single place is a county or a country or a municipality, but Brand X doesn't care that Paris, Texas is not in France.

With these considerations in mind, I had to redo UNIGEDS' place structure several times before I got it right. If you know what normalization is in a database, you can see by the above gedMOM sample that this structure is not denormalized. In spite of the complexity, nothing is recorded twice.

After beating my head against the wall trying to decide what to do with GEDCOM's having split addresses from places as if they were not the same thing, this is what I decided. In the import process, rather than trying to guess at what might be the exporting vendor's intentions, it's probably best to add all ADDR to contacts and add all PLAC to places and not worry about it. Also I must remember to complain that in GEDCOM, the PLAC and the ADDR *are not linked to each other*; they are the same level. Here's an address structure subordinate to a burial. This is from vendor Legacy's sample tree GEDCOM export file. Notice the degree of repetition.

```
1 BURI
2 PLAC Minneapolis, Hennepin, Minnesota, United States
2 ADDR Layman's Cemetery
3 CONT 2945 Cedar Ave S.
3 CONT Minneapolis, MN 55407
```

3 ADR1 2945 Cedar Ave S.
3 CITY Minneapolis
3 STAE MN
3 POST 55407

73. Don't translate ADRS to CNTCT_ADRS, REPO_ADRS etc., since an address can be linked to not only a contact, but also a repository or an event. GEDCOM seems aware that its PLAC tag as provided by the specs is so inadequate as to not lend itself to admitting a street address or a rural route address as part of a nested place. While UNIGEDS assumes that Great-Great-Grandfather Solomon didn't have a fax number or a zip code, it's likely that he did have a street address or rural route address, so UNIGEDS' `contact` table could be used for this purpose.

But preferably a street address would just be the smallest nest in a nested place chain, so that's how Treebard does it. The problem is that GEDCOM's creators, being splitters and not lumpers, have split the address from the place as if a street address was not just another nest in a series of nested places, so we need a place to store event addresses imported by GEDCOM.

UNIGEDS is designed to provide complete, long nestings as a single string of smaller enclosed places, while retaining the place ID of each of the individual places. The way to do this did not come to me on a silver platter, it was kind of a long, dusty road to getting this figured out to work in a simple way. It was necessary because UNIGEDS' main reason for providing a long string of nested places is so that genieware vendors can provide autofill fields in their user interface.

So if the subject of research, Herbert Van Dorf, lived at Rural Route 3, Van Dorf Township, Van Dorf County, New York, United States of America, the user neither has to type, paste, nor abbreviate this compound place name or "nesting"; chances are, it will all fill in after he types one or two words, and in some cases just a few letters. I felt that an autofill input field is essential to the

process of raising the bar for how genealogy software treats places.

Since GEDCOM conveys street addresses which are linked to genealogy events with ADDR instead of PLAC--another example of using a single tag for unrelated purposes--event addresses could be stored in UNIGEDS' contact table which was not designed for that purpose. But I have to draw a line somewhere, so instead, the ADRS tag when subordinate to an event should be sent to the exceptions log where the user will be asked to link the street address to the correct genealogical place, and if there is other information besides just an address line, the user can choose among 1) discarding the other info such as irrelevant zip codes etc., 2) storing the data in a note, or 3) storing it in Treebard's contacts feature. The latter choice would not make sense for addresses and other contact information that's out-of-date, which is why GEDCOM should not use the same tags and structures for submitters and repositories--which are presumably current contact data--and historical addresses that we would have no reason to use as contact information.

Having said that, it seems like it would be possible to glean the correct place from the GEDCOM event subrecord since the PLAC and ADRS tags would be siblings, subordinate to the same event tag. The problem is that the address line(s) might not be worded the right way, might be too long, or something else might be wrong with just plopping it into a long string of nested places, so I think it would be best to send addresses linked to events to the exceptions log.

In addition, GEDCOM like this...

```
2 ADDR Bush Family Cemetery
3 CONT 3944 Ouichita Ave
3 CONT Maddox, AR 83838
3 ADR1 3944 Ouichita Ave
3 CITY Maddox
3 STAE AR
3 POST 83838
```

...which is "correct" as prescribed by the specs, is cancerous. "Backward compatibility" is the excuse given, but in that case, GEDCOM mistakes itself for software. It is not software, whether it has versions or not, and it should not. It's a text file. A version of GEDCOM is an entity unto itself, and it should not make reference to other versions. It has cost me long minutes of dense, foreboding pondering as to what my response should be to this goofy guff.

It requires logic code to determine whether both methods of recording an address are used, more code to somehow link them together, and... anyway, my solution is to send them both to the exceptions log if I ever get around to it, and move on to the sensible task of inputting addresses for repositories. Events should be linked to places, and an address is part of a place. If the user wants to record contact information for a cemetery, then the cemetery should be recorded as a repository, and then the address will go where it belongs. In the absence of things being done my way, I'll have to make some suggestions to the user from within the exceptions log. Code will still have to be written to gather all the related lines together in a meaningful way, as it would never do to show the GEDCOM lines to the user, who might have a conniption fit, and I wouldn't blame them if they did.

74. The way that places have been handled in genieware up till now has been an insult to history, geography and genealogy. It's as if to assert that we are jist little ole pea-pickin' genealogers, and geographology is not our little ole problem.

The ease with which an entire database full of Texans can be destroyed by one innocent error on the part of the user--who is likely neither a geographer nor a computer nerd--is mind-blowing. It should be hard, not easy, for a genieware user to mistakenly link "Paris, Texas" to an event that occurred in "Paris, France".

The GUI has to handle both the inconvenient fact that place names are duplicated in the real world as well as the heretofore ignored fact that places are elements of genealogy. As elements, each single place needs a unique ID number, each nested place needs a unique ID number, and each place name (including each separate place named Paris) needs its own unique ID number.

There are dozens of places on earth named "Paris" and there's a town in France named "France". There's a town in Maine named "Maine". Dallas, Texas was once Dallas, Republic of Texas. The Dallas, Texas of today is currently straddling four counties. Because of these slight inconveniences--which are only "slight" inconveniences till we start tracking these places with a computer--most genieware GUIs sweep the details under the rug and hope the user doesn't notice. The user assumes he is working with a modern, sophisticated computer database program, but he hasn't looked under the rug.

The user interface has to handle the complexities of place identity, which is impossible if the data structure is where the dirty details have been swept under the rug. In the case of GEDCOM, the details never got in or out of the data structure because GEDCOM swept them under the rug before anyone else got a chance to.

Sorting this out after a mistake has propagated itself throughout the tree is the wrong approach. Expecting the user to go back and fix it is not very nice. It's one of the reasons I stopped using genieware and started writing it. I suspect that maybe Gramps and possibly one or two other vendors have taken a stab at storing place names accurately. The others are possibly using GEDCOM's devil-may-care approach as their excuse: "if I bother to treat places with the correct detail, in order to export it with GEDCOM, I'd just have to dumb it back down again." That's what I had to do in order to export places with GEDCOM.

75. I suspect that one reason places have been treated simplistically in genieware is that developers tend to be too detail-oriented for their own good, but then go after

the wrong set of details: every place on earth. Why on God's green planet should genieware try to store everything it can find on Google Maps? Or anything of the sort? UNIGEDS comes fully packed with one place: a place called "unknown". I don't even remember why I added that one place, it was a long time ago.

If we have to track places accurately, and we also have to come pre-loaded with every place on earth, well there's no hope. Why should users input their own places? Because every document lists a place name differently, and it is the task of the researcher to work through the discrepancies, decide what to do about it, and store the data in a way that suits himself. If I were to make a list of countries every ten years, the list would change every ten years. Tracking every place on earth is about as useful as sitting around memorizing the phone book.

76. Using extra commas to indicate missing jurisdictional levels is a throwback to the days of the typewriter. With typewriters, we had no choice but to do ugly things such as using ALL CAPS FOR SURNAMES; that was an attempt to be codelike on the part of well-meaning detail nerds. Welcome to the 21st century; this is ugly, useless, unnecessary, and requires more code than not doing it: "Telluride, , , USA" in the case where we don't know that Telluride is in San Miguel County, Colorado. There's no way to know the right number of commas because there's no fixed number of jurisdictional levels anywhere on earth, and making some arbitrary decision about which levels count and which levels don't is not our decision to make.

What if there's a township level between city and county? What if there's a village or district or borough or named neighborhood nested inside the city level? What if there are no counties in Louisiana? What if the county level doesn't exist in all parts of Alaska? What if a user prefers to ignore everything but city, state, and country? That should be up to him. Tracking jurisdictional levels in any code-dependent way in genieware is a can of worms. Because it can't be done accurately for every place, it

should not be done at all. It's possible to record jurisdictional levels--as the trivia that it is, as place types unrelated to each other--without making any code dependent on these levels. The extra-comma place string will certainly be abandoned sometime before the next millenium, so we might as well stop doing it right now.

Where I live, we don't have states, we have provinces. We also have cities, municipalities, baranggay, and purok. Now where ya gonna put yer commas, and how many of them? Programmers need to know where to draw the line on detail, which details are elements of genealogy, which details are not reducible to accurate categorization, and which details are trivia.

GEDCOM Can't Handle Adjunct Roles

77. ASSO.RELA is the only 5.5.1 specs-prescribed way to do roles, but not the only way that roles are being done. This is because ASSO.RELA is the least useful. This is because it isn't linked to an event but to an individual. So you could say Jane had a biographer Sally but there's no way to link to Sally's event "wrote a biography" or Jane's event "someone wrote her biography." Therefore, ASSO.RELA should not be used for roles such as "flower girl". Flower girl... when? where? In line with GEDCOM's way of giving two or more tasks to a single tag, so developers can have more fun writing code that tries to read minds, ASSO.RELA can also be used for relationships (ASSO.RELA could also denote that Sally was also Jane's granddaughter) and the _WITN._ROLE could be used for roles.

On second thought, that was way too calm. I just wasted two days on this ridiculous tag and here's the real rant on ASSO.RELA:

`ASSO.RELA` refers to a submitter's relationship to the person in the INDI record if the ASSO line pointer is a SUBM pointer:

```
0 @I1@ INDI
1 NAME Joe /Jacob/
```

```
1 ASSO @SUBM1@
2 RELA great grandson
```

That whole notion goes down the tubes if you have only one submitter (see the details in GEDCOM v. 5.5.5 specs where the notion of more than one submitter is put in its place.)

`INDI.ASSO.RELA` refers to any person in the tree who has a primary ID and is referenced by a pointer in the ASSO line and the role type in the RELA line:

```
0 @I1@ INDI
1 NAME Fred/Jones/
1 ASSO @I42@
2 RELA Godfather
```

Participation in an event seems to be *implied* in the next examples from Kessler but the ASSO and the BIRT are siblings so it's just *implied*. Ignore the implication and link it to the person as per specs, i.e. ALL THREE OF THESE USAGES are the same, and the FK can either be a `person_id` (INDI) or a `contact_id` (SUBM).

```
1 BIRT
2 DATE 3 AUG 1780
1 ASSO @I4@
2 RELA Saw birth
1 ASSO @I3@
2 RELA Godmother
```

ASSO.RELA is not useful for linking events to roles, it is only useful for relationships, which are linked person to person. Roles need to be linked to events. The implied link to events in the last example above can't be relied on, since the ASSO.RELA's following its sibling BIRT event could be a coincidence.

Once again, we are made to guess and figure things out because of the GEDCOM specs' insistence on giving two or more meanings to many single tags. Are they still doing this? I'm afraid to look into that.

A glance at the generally unaccepted GEDCOM 7 specs will be all it takes to assure any sane programmer who is not a certified apologist for GEDCOM that GEDCOM's creators are not solving problems by simplifying anything, but rather by making things more complicated. I have no doubt that they're doing their best. But doing their best to preserve GEDCOM's role, undeserved as it is, as the so-called standard of file transfer in genealogy. Instead of doing their best to fess up: GEDCOM will never cut the mustard because GEDCOM *can't* ever cut the mustard. Complicating GEDCOM's impending demise with more versions of increasing complexity is no solution. It could even be a case of *winning by intimidation*: make it sound so technical that no one will question it because they're afraid of sounding ignorant.

I'd like to point out that genealogy is not rocket science. The time has come to adopt a universal relational database structure such as UNIGEDS for all genealogy programs, and put an end to this text-file-transfer charade. If a little old hick from small-town Southern Colorado like me can figure out how to use SQLite and have fun doing it, then so can all the vendors who for no particular reason have avoided SQL.

Based on not being able to find any GEDCOM exporters actually using ASSO.RELA, and on other problems mentioned above--especially the obvious fact that an adjunct role in an event has to be linked to an event--UNIGEDS import can realistically only support the ASSO tag by sending any occurrence of this tag to the exceptions log.

This quote from Tamura Jones is not about the ASSO tag but about a custom tag usage based on the ASSO tag, which is worse. I like the quote, although I disagree with using custom tags for anything.

"Witnesses must be associated with the event they witnessed. This _ASSO record approach fails to associate witnesses with an event and because of that,

cannot even tell you which marriage someone witnessed. It is best to use some vendor-defined _WITN record on the event itself."

--Tamura Jones, The FamilySearch GEDCOM 5.5.1
Specification Annotated Edition, p. 64

78. There is so much wrong with the so-called "ASSOCIATION_STRUCTURE"--which is a long, unilluminating synonym for "ASSO.RELA & Friends"--that trying to encapsulate the problem tempts me to fall into using the kind of adjectives that don't really belong in civilized discourse. I cannot mount an all-out attack on this tag because it would be pointless and it would make me look like the bad guy, or the pitiful complainer who won't bite the bullet and deal with the reality of the situation. But I must bring up and/or re-bring-up one or two points, because the reality of the world and its related factoids is exactly what GEDCOM fails to take into account.

ASSO is one of GEDCOM's most ambiguous and/or confusing tags. It's supposed to work for either relationships or roles, but since the term "role" is short for someone who plays a role in an event, it actually doesn't work for roles because there's no event link. ASSO is supposed to give a link to either an INDI record or a SUBM record, to record a relationship between two people, OR a role played by a person in an event.

There's no excuse for using ASSO for both, but it's a duplex tag x 2 (a doo-doo-plex tag?), because its supposed functionality extends in two different directions twice. As if SUBM could somehow morph into INDI, now the contact who provides some family data can also be added as if he were a person in the tree because he is, but he doesn't have to be added to the tree, even though he is a person in the tree, because he's a contact. I believe that's a pretty good analysis of the situation, and please don't injure yourself trying to figure out what all that means.

If GEDCOM was trying to avoid confusion by avoiding repetition--i.e. not entering the same person as both a

contact and a family member--then that attempt failed. This is the sort of flexibility that destroys the weak. ASSO tries to do so much with so little, but in its effort to mimic flexibility, it only creates confusion, dooming itself to the Purgatory of Unusable Tags. It is a pointer to either of two record types. For starters. Which makes it an unpointer that translates to, "Go figure it out."

If you suggested to a SQL programmer that SQL could be improved (made more flexible perhaps?) by allowing a foreign key to refer to *either* the Individual table or the Submitter table, that poor SQL fella could find himself involuntarily rolling on the floor laughing. What GEDCOM gets away with... while people still call it a "standard"... you can't make this stuff up.

Keys and Values are not Reversible

79. As I compiled a few examples of GEDCOM lines that were supposed to denote romanization of person and place names, along with subordinate TYPE lines to denote what type of romanization and phonetic rendering systems were being used, I finally saw through the queasy feeling and realized this was just another case of GEDCOM not knowing the difference between keys and values. A simple for instance of what keys and values are:

KEY	VALUE
boy	Alex
game	tiddly-winks
game	Monopoly
boy	Jed

Since GEDCOM, in its earnest effort to record all the important factoids that genealogists care about, has not come to terms with what keys and values are, a GEDCOM file sometimes specifies that we do something that amounts to this:

KEY	VALUE
boy	Alex

Jed	boy
Monopoly	game
game	tiddly-winks

A category is a key. An example of a member of that category is a value. Not able to keep this straight, GEDCOM is, amongst other things, a system of misplaced keys and values. (Sometimes in a proper *key:value* system, a value can be a key for another value; then we have nested sets of keys and values. That's not what I'm talking about.) GEDCOM makes tags (keys) out of values and avoids giving primary status to elemental keys:

```
0 @8@ INDI
1 NAME Jane Dalton
1 EVEN
2 TYPE coronation
1 OCCU politician
1 TITL Queen of San Francisco
1 RESI
1 PLAC 東京
2 ROMN Tokyo
3 TYPE Pinyin
2 FONE [to:kjo:]
3 TYPE IPA
```

Why the null values? Because EVEN (a key) has no value (GEDCOM doesn't give events an ID) and RESI is a type of event which really belongs in the value column. Why is TYPE used for three different things? Short keys are cute, but using the same variable name for more than one thing is not done in programming. During my GEDCOM-import-program writing project, facing the prospect of dealing with romanization and phoneticisation of names and place names other than English, I finally realized that the queasy feeling I was getting was the one you get when you are standing at the edge of a bottomless rabbit hole and leaning forward. I felt a gnawing hesitation to proceed. Only when I sat down to do it anyway did I realize why.

I've learned by designing real databases that types are special. They are primary, you might say primal, or more primary than primary. For example, names are primary elements of genealogy. But name types are more primary than names. You can't reference a name ID in a name type table; it has to be the other way around. Types tables reference no other tables. You don't put a file folder in a document, you put a document in a file folder. An event table references an event_type_id, not the other way around.

Without experience in designing databases, this sort of thing wasn't obvious to me 5+ years ago when I first started trying to design SQL databases. I still remember the complete blank I drew when faced with trying to design a database table in which to record events and their traits. Nothing about it was obvious.

But it boils down to this: a name table records name_type_id as a foreign key: a reference to a primary name_type record where each name_type has a unique ID that can be used in other tables. Data doesn't come in GEDCOM'S unnatural, linear, force-fit hierarchies, it comes in networks of related pairs.

UNIGEDS Needs a Transcription Element

80. I had left ROMN and FONE partly because I thought they'd be easy and partly because there was nothing like romanization or phonetic fields in UNIGEDS.

My first impulse was to create a place_name_type table with a field for the string. The only problem with this is that is was wrong. This sort of thing happens all the time in database design. You start with what comes to mind, build it and try to overlook its inadequacies, but you can't. In what way does this model fail? Does it provide too little detail or too much? Does it require blinders to make it seem to fit? This one seems to do so.

The problem with PLACE_NAME_TYPE isn't that it's barely needed--some trees might use it a lot--but that it's wrongly named. The problem is that this type should equally

cover person names and place names, as the ROMN and FONE tags do in GEDCOM. The solution was that UNIGEDS needed a transcription table and a transcription type. Upon considering whether "place_name_type" might be all wrong, it also came to mind that a transcription of a name is not a type of name at all, but something else: a transcription.

"東京" and "Tokyo" are not two different names for the same place. They're two different ways to reduce the same name to a writing system. "東京" is the name itself and "Tokyo" is a transcription.

Here's an example of expanding an overly-complex GEDCOM INDI record into a set of precise gedMOM records, using a real historical person:

GEDCOM:

```
0 @I2001@ INDI
1 SEX F
1 NAME 青山みつこ
2 ROMN Mitsuko Aoyama
3 TYPE romaji
1 NAME Mitsuko Thekla Maria
2 TYPE married
1 TITL Countess of Coudenhove-Kalergi
1 BIRT
2 DATE 7 JUL 1874
2 PLAC 東京
3 ROMN Tokyo
3 FONE [ to:kjo:]
1 DEAT
2 DATE 27 AUG 1941
1 IMMI
2 NOTE one of the first Japanese people to...
```

gedMOM:

```
PRSN 2001
PRSN_GNDR female
* *
NAME 1609
```

PRSN_FK 2001
 NAME_TYPE_FK 27
 NAME_STRG 青山みつこ
 * *
 NAME 1611
 PRSN_FK 2001
 NAME_TYPE_FK 81
 NAME_STRG Mitsuko Thekla Maria
 * *
 NAME 1612
 PRSN_FK 2001
 NAME_TYPE_FK 88
 NAME_STRG Countess of Coudenhove-Kalergi
 * *
 EVNT 30
 EVNT_TYPE_FK 1
 PRSN_FK 2001
 PLACE_FK 95
 EVNT_DATE 7 JUL 1874
 * *
 EVNT 26
 EVNT_TYPE_FK 4
 PRSN_FK 2001
 EVNT_DATE 27 AUG 1941
 * *
 EVNT 34
 EVNT_TYPE_FK 14
 PRSN_FK 2001
 * *
 TRANSCRIPTION 35
 NAME_FK 1609
 TRANSCRIPTION_TYPE_FK 54
 TRANSCRIPTION_STRG Mitsuko Aoyama
 * *
 TRANSCRIPTION 112
 TRANSCRIPTION_TYPE_FK 54
 PLACE_NAME_FK 95
 TRANSCRIPTION_STRG Tokyo
 * *
 TRANSCRIPTION 38
 PLACE_NAME_FK 95

TRANSCRIPTION_TYPE_FK 73
TRANSCRIPTION_STRG [to:kjo:]
* *
PLACE 95
* *
PLACE_NAME 43
PLACE_FK 95
PLACE_NAME_STRG 東京
* *
NOTE 763
NOTE_STRG one of the first Japanese people to ...
* *
NOTES_LINKS 819
NOTE_FK 763
EVNT_FK 34
NOTE_TITL Countess Mitsuko's Immigration
* *
NAME_TYPE 27
NAME_TYPE_STRG birth
* *
NAME_TYPE 88
NAME_TYPE_STRG title
* *
NAME_TYPE 81
NAME_TYPE_STRG married
* *
TRANSCRIPTION_TYPE 54
TRANSCRIPTION_TYPE_STRG romanization (romaji)
* *
TRANSCRIPTION_TYPE 73
TRANSCRIPTION_TYPE_STRG phonetic (API)
* *
EVNT_TYPE 1
EVNT_TYPE_STRG birth
* *
EVNT_TYPE 4
EVNT_TYPE_STRG death
* *
EVNT_TYPE 14
EVNT_TYPE_STRG immigration

By basing a gedcomoid on the structure of SQL, I've arrived at a better data structure for GEDCOM's ROMN and FONE tags. Until I stopped calling this a place name type, it didn't occur to me that a transcription table would be needed. It's needed because of the cardinality of the *name:transcription* data pair. Each name can have many transcriptions, but each transcription refers to only one name. The cardinality of *name:transcription* is one-to-many, so the name ID or place name ID foreign key goes in the many side of the relationship, the transcription table.

Until I got to the heart of the matter, which is cardinality--i.e. what kind of relationship exists in the real world between a name and its transcription--I had no idea a transcription table was needed. A transcription isn't treated like a name because it isn't one.

Assertions, Sources and Citations

81. GEDCOM's TEXT tag is GEDCOM's neglected assertion element.

In GEDCOM, the TEXT tag is made subordinate to the wrong thing, the SOUR tag. Yes, I'm always saying, "An assertion is what the source says," but to be more accurate, I should be saying that an assertion is what the source says at the citation within the source. TEXT should be subordinate to PAGE, and the meaningless, valueless DATA tag should not exist.

Assertions have been forgotten by vendors, because they are so omnipresent that we take them for granted. Assertions are what make sourcing worth doing, because if the source had nothing to say, there would be no genealogy. So you might say that assertions are the heart of genealogy. That doesn't mean we give them credit.

As usual, we don't know how to link anything to anything in a real database till we define the cardinality of the relevant pairs of data involved. A citation, in one line of a census page or one paragraph in a book, can yield many

assertions, but with TEXT subordinate to SOUR, we are left with such a vague glop of undifferentiated assertions that assertions have been assumed to be an abstraction not worth looking at. There's nothing wrong with assertions, without which genealogy would not exist. What's wrong is to lump them all together as "source text and then linking blocks of lumped-together assertions to the source instead of linking them to what they're actually related to.

Here's a TEXT tag value in a real GEDCOM file. This is from a major genieware vendor whose name I won't mention.

```
0 @I224330@ INDI
1 NAME Matilda O. /Ross/
2 SOUR @S41634@
3 PAGE Year: 1910; Census Place: Trenton, Alachua, Florida;
Roll: T624_156; Page: 7B; Enumeration
3 DATA
4 TEXT Birth date:  abt 1858Birth place:  South
CarolinaResidence date:  1910Residence place:  Trenton,
Alachua, Florida
```

Since GEDCOM links TEXT to SOUR, the vendor is not inspired to create a useful interface for delineating individual assertions to link to names or events. The sample above contains not an assertion, but a clump of them:

- a date assertion about a birth
- a place assertion about a birth
- a date assertion about a residence
- a place assertion about a residence

This ball of string is not leading anywhere. It's due to a feature being built around GEDCOM's structure instead of the way the data works in the real world. In spite of there being four separate assertions about two separate events, the clump of assertions is actually linked to a name, not to an event.

What exactly are vendors missing? What is an assertion really supposed to be linked to? We'll get to that. It's actually a bit of a surprise, which I didn't discover till I got down on all fours and sniffed out the real cardinality of the real-world data.

Assertions are not some kind of abstraction. They are what we use to draw conclusions. Contradictory assertions made by various citations are what we compare to each other, whether our genieware collects them or displays them or ignores them. There's no other way to come to a conclusion in genealogy, other than to look at the assertion: what the source says.

Assertions, sources, and citations are primary elements of genealogy and the only way to treat them correctly is to analyze their relationships to each other correctly. The notion of subordinate tags in GEDCOM doesn't work, especially with only seven primary elements recognized and everything else stuffed just anywhere, apparently based on physical size and/or nesting, e.g. "Text is found inside a source so it's subordinate... blah, blah, blah..."

The direct relationship of assertion to source is as follows: zip. None.

The relationship of citation to assertion is one-to-many, because each citation can yield multiple assertions, but each assertion relates to only one citation. There can be similar and identical assertions coming from different citations, but they are separate assertions with unique IDs. This is important. In this way, each assertion gets a vote as to what conclusion might be drawn about a name, an event, an attribute, or whatever element we're gathering evidence about. If six citations assert that Bob and Mary were first cousins, and only one citation asserts they were just neighbors, that will influence our conclusion about their relationship. Whether your genieware recognizes the existence of assertions or not, it is assertions which you

use every time you draw a conclusion in your genealogy, unless you're one of those genealogists who just make stuff up. There's plenty of those genealogists, but they're beyond our scope.

Assertions must be linked to citations, and assertion is the "many" side of their one-to-many relationship, so the foreign key for the citation ID goes in the assertion table. This way, many assertions can be linked to any one citation. Unfortunately, GEDCOM's arbitrary definition of TEXT as a subordinate of SOUR (source) means that I thought I'd have to put TEXT into an exceptions log with suggestions as to how the user should re-enter the data in a program such as Treebard GPS which honors the importance of the assertion and treats it correctly.

I ended up cheating instead, by deleting the meaningless DATA line, which in most cases will cause the TEXT to link to the PAGE (citation) line. I don't like doing this, but I'm doing it experimentally till I find it not working somewhere, and then I will find a way to add the problem to an exceptions report for those cases where my cheat doesn't work.

82: From page 40 of GEDCOM 5.5.1 specs:

"Actual text from the source that was used in making assertions, for example a date phrase as actually recorded in the source, or significant notes written by the recorder, or an applicable sentence from a letter. This is stored in the '.SOUR.DATA.TEXT' tag context."

Since Genbox (an app which I used to generate my sample GEDCOM for the import program development effort) provided no GUI functionality to link an assertion to a citation, I pretended otherwise and hand-edited the assertions I'd entered (lumped together and linked to the source) to match the GEDCOM specs so a proper import of assertions could be tested. Genbox put out 5.5 GEDCOM, not 5.5.1, but the specs are the same in this area for both versions.

DATA is the perfectly vague name for a meaningless tag with no value that does nothing. A GEDCOM replacement post somewhere online shrugged off assertions as just too "abstract" to be bothered with. That attitude of summary dismissal might be in part due to the fact that GEDCOM has relegated this all-important element to subordinate status nested under a valueless tag, a tag which is not only meaningless and unnecessary, but also barricades TEXT from the PAGE tag it should be linked to. Then the assertion is placed in another ho-hum, non-descript tag line, the TEXT line. The DATA tag has no reason to exist and the TEXT tag should have been called ASSERTION or ASRTN.

Also both the DATA and TEXT tags are each given more than one job to do in the GEDCOM specs. So vendors are going to deal with assertions only begrudgingly, having had no help at all from the GEDCOM rules to make assertions look interesting, useful, or relevant. Gramps includes one TEXT line in their sample GEDCOM, and it's a joke about some guy wearing a dress on every third blue moon.

```
0 @I0044@ INDI
1 NAME Louis /Garner/
2 SOUR @S0002@
3 PAGE Page 11 2/3.
3 DATA
4 DATE 5 MAR 1990
4 TEXT On every third blue moon, Lewis Anderson Garner
would dress in a purpl
5 CONC e dress and claim that his name was Louis Garner.
```

Gramps' syntax is correct, which beats Genbox' missing DATA:TEXT syntax that reflected Genbox' GUI's lack of any way to link assertions to citations. Gramps' use of the DATE tag as a subordinate part of the assertion (to record when the assertion was made by the source) is also correct according to the specs, but in both cases of nominal correctness, it's the specs themselves that are poorly designed. Any given tag should be used for one thing and one thing only in order to make the import/export process smoother and faster for both the end user and the dev-user.

Here's how GEDCOM could have specified this assertion:

```
0 @I0044@ INDI
1 NAME Louis /Garner/
2 SOUR @S0002@
3 PAGE Page 11 2/3.
4 ASRTN On every third blue moon, Lewis...
5 CONC e dress and claim that his name was...
5 ASRTN_DATE 5 MAR 1990
```

83: Continuing the issue of GEDCOM's pseudo-assertion tag `TEXT`. Or `DATA.TEXT` I should say, since the other allowed use (`SOUR.TEXT`) is pretty vague. "The census says Mary was born in 1843" is not useful. The assertion has to be linked to the citation, not the source: "The 1850 Corn County, Kentucky census on page 47 says Mary was 7 years old in 1850" is a meaningful assertion.

Here again is Gramps' blue moon example:

```
0 @I0044@ INDI
1 NAME Louis /Garner/
...
2 SOUR @S0002@
3 PAGE Page 11 2/3.
...
3 DATA
4 DATE 5 MAR 1990
4 TEXT On every third blue moon, Lewis...
5 CONC e dress and claim that his name was...
```

This is the correct specified syntax, but it's wrong. I'm harping on this theme because the correct way to treat assertions has to be nailed down once and for all. It has to be understood, in terms of cardinality i.e. how pairs of elements interrelate in the real world so we can translate that into the right kind of relationships in a SQL database structure like UNIGEDS. So I don't mind being repetitive. Here again is my correction of the GEDCOM structure:

```
0 @I0044@ INDI
```

```
1 NAME Louis /Garner/
2 SOUR @S0002@
3 PAGE Page 11 2/3
4 ASRTN On every third blue moon, Lewis...
5 CONC e dress and claim that his name was...
5 ASRTN_DATE 5 MAR 1990
```

It has to be reiterated and driven home that (in GEDCOM's terms) a citation (PAGE) should not be a sibling of an assertion (TEXT). You don't see two OCCU tags as subordinates because they aren't linked to each other. Here we see an assertion as a sibling to a citation, as if they were not linked to each other. Because of the cardinality, assertion should also not be linked to source. So the corrected version shows ASRTN as subordinate to PAGE and PAGE subordinate to SORC. This isn't perfect yet, just because GEDCOM's notion of subordination is not a consistently useful structure (except to the eye, because the human mind is a self-teaching machine and a computer is not; thus we are fooled into thinking those GEDCOM line numbers are something great, simply by looking at them).

The problem with PAGE and TEXT being siblings is that it introduces us to one of GEDCOM's insidious Special Needs. Since GEDCOM is a text file and therefore can't read minds or guess what those line numbers mean, it has a special need, which is it hopes that everyone who uses the DATA.TEXT tag (according to the specs, as a sibling of the PAGE tag) will place the PAGE tag ahead of the TEXT tag in the SOUR record. There is a special need to detect subordination of same-numbered lines here, in order to link

```
3 PAGE Page 11 2/3
```

to

```
3 DATA
4 TEXT On every third blue moon...
```

If the order of these siblings (the lines that start with "3") were reversed in the record, the situation would be

even worse. We'd have to link the assertion to a citation which *follows* it. None of this is insurmountable(?), but it would be a miracle if all developers were out there making sure their DATA.TEXT tags come after their PAGE tags. In the GEDCOM files I've looked at, the PAGE tag does in fact always precede the DATA.TEXT tag. This suggests that vendors have realized there's a relationship between the two that is built into such ordering.

Instead of expecting vendors to magically know that the sibling tags should be in a certain order in order to describe subordination (linking) not taken into account by the specs, the specs should correctly reflect relationships of real-world data. But in fact, vendors usually just ignore the vague-sounding DATA.TEXT tag, or else use it as an oddball, catch-all junk-drawer, because you have to read the specs pretty carefully as well as get lucky doing additional research to learn that "assertion" is not just a word, but an element of genealogy.

84. Just in case you thought I was finished trying to get the point(s) across about assertions and their critical role in genealogy--a role which we'd like to see acknowledged, appreciated, and correctly reflected in genieware design--well no, I wasn't finished yet. I just realized that I need to explain in slightly more detail why the TEXT tag doesn't cut the mustard to represent assertions. For one thing, "assertion" starts with an A and it doesn't have an X in it anywhere, so where does this vague and overused term "TEXT" come from?

Unlike citations, which should be stored once and repeated by referencing a citation ID, every assertion is unique even if it is identical text as another assertion linked to the same event or name. The two assertions seem identical in every way, so why do they need separate assertion IDs? The answer is simple. Here's an example of an assertion: "Jennifer Qualradzhe III was born in a London bomb shelter in 1943." The user could have made three separate assertions: "JQ was born in London,", "JQ was born in a

bomb shelter," and "JQ was born in 1943". This would be better but the user didn't do this and didn't have to. (People won't use your software if there are a lot of rules.)

The data structure still has to work in a case like this where the user copied & pasted the same assertion three times. Once as a date assertion (1943), once as a place assertion (London), and once as a particulars assertion (bomb shelter). Here's why the date assertion, place assertion, and particulars assertion have to have different IDs even though the assertion text and linked event are identical. It's because in reality, Jennifer was born in a *Dresden* bomb shelter and we have *other* assertions which clarify this.

Assertions are limited and specific things, but the DATA.TEXT tag in GEDCOM has forced users to lump separate assertions into one undifferentiated mass of text. The user has to be able to see and appreciate that the date assertion is correct but the place assertion is wrong. Without separate IDs for each asserted factoid, the assertion would find itself pushed to the back of the bus, as it always has been, ever since the first genieware came out and GEDCOM was invented to serve its limited abilities.

85: Assertions again? Yep. The below GEDCOM construction might explain why one member of the GEDCOM replacement committee announced in an online post that assertions are too abstract to bother with:

```
0 @1@ INDI
1 NAME Robert Eugene/Williams/
1 SEX M
1 BIRT
2 DATE 02 OCT 1822
2 PLAC Weston, Madison, Connecticut
2 SOUR @6@
3 PAGE Sec. 2, p. 45
3 EVEN BIRT
```

4 ROLE CHIL

Here we have an event linked to a source linked to an event, i.e. INDI.BIRTH.SOUR.EVEN.ROLE. The redundancy of a BIRT tag on one line being re-used as a value on a subordinate line is pretty much unforgiveable, but let's leave morality out of it and just try to figure out what was actually intended. Why would an event, and the role played by someone in that event, be linked to a source? Because some well-meaning GEDCOM creator was trying to recognize the existence of assertions (things that sources say about events, names, etc.). The problem is that the specified conglomeration of ambiguous, redundant GEDCOM tags in a structure pretending to be a hierarchy created the illusion that assertions are hopelessly murky, abstract pseudo-elements. This was accomplished by not naming them assertions in the tags that actually refer to them, and by not separating the user's conclusion about an event from the source's assertion about the event.

To handle the above gobble-de-goop, the least obvious but most correct strategy is to wash your hands of it. It's clearly redundant to say that the role played by the person being born in a birth event is "child". I mean, come on. We are still curious, however, what was intended here. Will any data be lost if this configuration of ambiguous tags is just ignored?

Regarding assertions in general, because of the nature of the relationships of the elements involved (the cardinality of source, citation, and assertion in relation to each other), in order to correctly link a source to a citation, the GEDCOM import code must create an assertion ID that's not in the GEDCOM, link the citation (PAGE) to the assertion, link the source to the citation, and give the user an opportunity to say what type of assertion was being made about the birth event (date assertion? place assertion? details assertion? role assertion? age assertion?) and ask the user to state specifically what the assertion was, e.g. "born on October 2, 1822" or "born in Weston, Connecticut".

This is not abstract, but the next objection is that it sounds redundant. It is not redundant, because six other sources might say that the person was born in Madagascar in 1830 or 1831. Getting back to the current complaint, the purpose of the SOUR.EVEN.ROLE tag as stated in the specs is to mention what type of event occurred (even though we already know it was a birth event.) The writer of the GEDCOM specification was obviously trying to put stress on a subtle point: the source is a *child's birth certificate*, while the assertion was about *the mother's name*.

The reason this whole thing sounds so abstract is that the GEDCOM creator, in his zeal to split hairs, missed another point that is not abstract at all: without the source's having said something, genealogy would not exist. So we have to make assertions--what the source says--explicit. To treat assertions as proper primary elements of genealogy (since genealogy would not exist without them, whether we recognize their existence or not), we must not lump them together, or let them merge into the conclusion of the genealogist. Conclusions and assertions will always be completely separate things, but they've always been confused with each other by inadequate, complicated, or abstract descriptions, and because the assertion has been swept under the rug since the dawn of computer genealogy.

For now, in my import program, I ignore EVEN when it's subordinate to a SOUR (source) pointer. Possibly there's a way to use this tag to indicate a role that we don't already know from the lines above it. Maybe the example in the specs should have said "midwife" instead of "CHIL" as a value, then something of value could be gleaned from these lines. But I have not been able to find a GEDCOM file that actually uses this curious construct, and the convention _WITN._ROLE could be used instead but I'm against all use of custom tags, because using them just perpetuates GEDCOM's overly-long life. Unfortunately, the board-certified ASSO tag is available and that is how roles are actually supposed to be handled, but it suffers from the same problem: purporting to record roles played, without saying what event the roles were played in.

Source Types Pre-exist Media Types

86. I was trying to figure out why GEDCOM's OBJE tag has always given me the heebie-jeebies, when it occurred to me that maybe the creators of GEDCOM haven't managed to separate media (an element of genealogy) from source (a different element of genealogy). They've gone so far as to create two separate tags, OBJE and SOUR. I never liked either of these tags right off the bat because you don't know what they mean by looking at them. When I see "SOUR", I think of a lemon. When I see OBJE, I think "Object? Could I get a second helping of vagueness, please?", but that's beside the point. I know they're really talking about multimedia objects, but naming the category "OBJE" is sort of like if the biologists had named the canine species "animal" instead of "canine". The specs also have this habit of defining vague tags with longer, vaguer pseudonyms. One of the specs' standard complications is to paste a bunch of nouns together like this:

SOURCE_MEDIA_TYPE

and then make you look in another chapter of the specs to figure out what that is supposed to mean, all of which is supposed to help me understand the MEDI tag, a tag which makes me think "medical". The values for MEDI a.k.a. SOURCE_MEDIA_TYPE are [audio, book, card, electronic, fiche, film, magazine, manuscript, map, newspaper, photo, tombstone, video], and a final definition for MEDI's value, along with three different ways to do the same thing, is given, "A code, selected from one of the media classifications choices above, that indicates the type of material in which the referenced source is stored." Since we don't have the specs' permission to add user-defined types, the vaguest (most useless and irrelevant) of the values--"electronic"--is one of the most popular.

Due to the extreme slowness of my nearly useless mind, I can't see what they're trying to accomplish here, unless it's to analyze the description of a source down to a schizophrenic gnat's hallucinated eyelashes. Except for "audio" and "video"--and maybe "fiche" and "film" which

should be spelled out as "microfiche" and "microfilm"-- isn't this the same list of types found in a category of source types? Don't ask me, I'm just an amateur genealogist, but doesn't "multimedia" in genealogy tend to refer to means of communicating information that require a machine to use, like a computer or a microfiche reader? Isn't the purpose of OBJE kinda being smeared or diffused here by giving the user a secret second hidey-hole to say that a source is a book or an index card? Heck, I don't know, but I don't wonder why the MEDI tag is rarely used.

I did find one vendor whose GEDCOM made liberal use of the MEDI tag. Legacy's sample GEDCOM uses it anti-specswise, directly subordinate to the zero-line of the SOUR record, like this:

```
0 @S173@ SOUR
1 MEDI Email
1 ABBR Email - from Nancy Newman to Geoff Rasmussen - 2002
Jul 27
1 TITL Email message to Geoffrey D. Rasmussen, 27 Jul 2002
1 AUTH Nancy Newman <nancym@unitelc.com>
```

Now that is a pretty pickle of poetry, I mean, do those lines all rhyme and alliterate for an artistic reason, or is someone just taking advantage of the ease with which GEDCOM lends itself to being employed with nauseating redundancy? Whatever the case may be, I imagine I could try to handle this custom use of a non-custom tag, but if it takes too long to ferret out yet another set of un hoped-for conditions to detect and respond to, I might just relegate something like the above monkey chatter to the exceptions log, down near the bottom somewhere.

87. To make matters even murkier, the specs imagine another extraneous use for this tag wherein we are specifying MEDI (source type and/or media type), through some flight of fantasy, as if it were subordinate to a call number or other locator in a repository (REPO) subrecord:

```
0 @S6@ SOUR
```

```
1 TITL Madison County Birth, Death, and Marriage Records
1 REPO @R7@
2 CALN 13B-1234.01
3 MEDI Microfilm
```

What a fascinating hodge-podge of unwanted choices to slog through in order to convey very little information.

In the process of learning how to import GEDCOM, I'm constantly being challenged to wonder whether I've missed something. I stand to learn from GEDCOM, and substantially so; after all, those fellas down at GEDCOM Central have been at it since I was a young hippie trying to save the world from my overactive imagination back in the 1980s. For example, because of GEDCOM's insistence on believing that a family element exists in genealogy, although I refused to listen for quite some time, it eventually became apparent that GEDCOM's FAM tag had meant to call itself a COUPLE element, and such a thing turned out to be very useful and helped me get rid of some convoluted SQL gymnastics I was calling "kin type".

More recently I had to face the locator, and its parent element the repository. Getting these things right is not always simple. Locator has a type (Dewey Decimal System? LLC call number? etc.) A locator can link to either a source or a citation or both. A repository can have multiple repository types. One result was an unusually complex many-to-many relationship among repository_type_id, source_id, citation_id, repository_id, and locator_id in a junction table. The result of today's complication is that I've decided to create a separate repository_links table for this linkfest and retiring the multi-purpose links_links junction table that I've never liked.

Now GEDCOM hands me this food for thought, (on a full stomach, I might add).

```
0 @S6@ SOUR
1 TITL Madison County Birth Records
1 REPO @R7@
2 CALN 13B-1234.01
```

3 MEDI Microfilm

Here we have what appears to be a media type (MEDI) linked to a locator (CALN). Conceivably--and this is a stretch, but I'm trying to be accomodating--this could cover a situation where a given source might exist in a variety of forms. Maybe there's a book version in the rare books room, a University Microfilms photocopy version on the shelf, and a microfilm version in a steel cabinet between the drinking fountain and the photocopy machine. All would have different call numbers (locators). The link to record would then be a four-way link among the source, repository, call number, and... is that MEDI value really a media type, or is it a source type? I'm going with source type. [Wrong.] I don't want to expand the meaning of "multimedia" to include magazines and index cards, no matter what the specs say.

I wasn't going to handle this oddity, but a few vendors use it, so what the heck, life is long and I'm so young and I have all the time in the world, let's get it done.

Here's another case where a vendor uses MEDI for source type. This should be a custom tag since it's not in the specs for 5.5.1 or 5.5:

```
0 @S36@ SOUR
1 MEDI Email
```

I haven't decided exactly how to proceed. I could just assume the specs got it right for SOUR.REPO.CALN.MEDI and Legacy got it right for SOUR.MEDI. But doing it the easy way now will mean doing it the hard way later--doing it over--so I'll have to think it through and do it the way that matches how these data relate to each other in the real world.

88. Re: FILE, FORM, MEDI. Here's a new twist on GEDCOM's habit of defining a given thing two different ways, in order to give us wanna-be GEDCOM dev-users more practice writing conditional tests. We've mentioned before that tags often get multiple meanings in GEDCOM. The twist this time

is that a given usage (what a tag gets used for) is assigned to two different tags:

multimedia_record

```
0 @65@ OBJE
+1 FILE <MULTIMEDIA_FILE_REFN>
+2 FORM <MULTIMEDIA_FORMAT>
+3 TYPE <SOURCE_MEDIA_TYPE>
+2 TITL <DESCRIPTIVE_TITLE>
```

multimedia_link

```
n OBJE
+1 FILE <MULTIMEDIA_FILE_REFN>
+2 FORM <MULTIMEDIA_FORMAT>
+3 MEDI <SOURCE_MEDIA_TYPE>
+1 TITL <DESCRIPTIVE_TITLE>
```

Here "SOURCE_MEDIA_TYPE"--wherein the creators of GEDCOM can't decide whether media is media or media is source so they named it both things--can be represented by two different tags. In the primary media record, source type is called "TYPE". In the downgraded shoulda-been-primary media subrecord, source type is called "MEDI". I'm getting woozy already and haven't even gotten any code written today, but unless I'm just seeing pink elephants on the ceiling, source type doesn't belong in either place. I'm not going to say a lot more about this, for fear of being tiresome. I do suggest that anyone fascinated by these topics who sincerely enjoys solving simple problems simply should grab a copy of SQLite, set out to write a simple genealogy database by actually doing it, and in the process, be careful not to research GEDCOM at all until you have a good understanding of the basics of how data relate to each other. I wouldn't want to confuse a SQL beginner by referring them to any information relating to GEDCOM.

89. I figured out how to tell the difference between media type and source type, by talking to myself about it until I

said something that couldn't possibly be true, and then figuring out how to correct myself.

Some source types are multimedia files and some aren't. The point is that multimedia files are files. That's why the most important tag in this group is FILE: the file name that points to the file on the computer of the computer user who input the data which was later exported via the GEDCOM tool. If not for the FILE tag, the MEDI/TYPE and FORM tags in the same subrecord would be meaningless. [NOT ALWAYS TRUE.] And just because GEDCOM thinks MEDI (source type? media type?) is subordinate to call number doesn't make it true.

We want to file this data in the right column of the right table in the database. Guessing usually gets the wrong answer. The key is to define both words, "source" and "multimedia" as they pertain to genealogy. To do this, look to each of the corresponding types in order to come up with a useful generalization. It doesn't matter what these two words mean outside of genealogical research, and academic hairsplitting is the last thing that's needed here. Questions like this have to boiled down to a decisive position, a definition, or there's no way to design a database schema that can be justified and explained to the satisfaction of anyone.

Attempting to separate these two categories means cleaning up the overlap between them. If the two categories can't be separated cleanly, they should not be separated at all (like events and attributes, which are different but should be treated the same in genealogy data). The first thing I can see is that the types tables I've started in UNIGEDS are too muddy, so here is how I will define (separate) them from each other.

Sources pre-exist media. Source types refer to objects and other real-world things such as interviews and reunions. Media are representations of sources, in fact that's what "medium" means. A medium stands between a sender and a receiver and conveys information from one to the other, so that if you can't stand in front of the gravestone at the

cemetery, you can get the same information from a medium--a photograph--that communicates the information instead.

Some media come with file names and machines are needed to use them, but this is not true of all media, so it's not where the line should be drawn. Here's what finally cleared this up for me: *Sources pre-exist media*. Would the source still exist (or once existed) even if there were no media to represent it?

SOURCE	SOURCE TYPE	MEDIA FILE	MEDIA TYPE
1) Jane Smith's headstone	grave marker	photo_of_JS_grave.jpg	photo
2) Smith Family Bible	family bible	smith_bible.pdf	book
3) 1962 Smith family reunion video	video record	none	VHS cassette
4) 1993 Smith family reunion story	family story	none	conversation
5) Interview with Grannie Jane	interview	none	reel-to-reel tape
6) Grannie Jane interview--CD	interview	grannie_smith.mp3	audio file
7) Jane Smith and daughters--1922	photo	jane__daughters.jpg	scan

Under "media type", the specs give us "audio" but these days you need more information. To use this group of tags, vendors are defining their own types whether the specs suggested it or not. Here's a case where the specs should not take themselves so seriously, where they should make as few restrictions as possible in a scenario that needs to be left open for future changes instead of left open for future versions of GEDCOM. We want one version of GEDCOM--or something better than GEDCOM--a version that would ideally never change because of the forethought put into the first version.

Take a look at how "photo" can be a source type or a media type in the above chart. The key to it is that the source pre-exists the media. The media conveys or tries to convey the source's content, nature, and/or appearance. The medium is evidence that the source exists, and evidence, for example, that a transcription is accurate. In line 1, Jane's headstone is the source because it pre-exists the photo of the headstone, which is the media. In line 7, the photo *is* the source, and the media is an electronic file, a scan of the photo which conveys the photo's content, and the back of the photo which outright states that the women with Jane are her daughters. In this way, we show how to cut through the fog. The source pre-exists the media, and

the media is a representation of that pre-existing source:
how the information offered by the source is communicated.

Similarly, a videotape, which can be a media type, could also be a source, but it is a specific video tape, of Grannie Jane talking at the family reunion in 1962 about moving to New Mexico in a covered wagon as a young girl. The media type might be "VHS cassette" but the fact that the reunion was taped constitutes the source, while 'VHS cassette' tells us what kind of equipment is needed to use the source. If the video recorder didn't make it to the reunion the following year, source type might be "family story" and media type might be "conversation".

Now I said all that so I could say this: the MEDI tag which attempts to record media type should not be second-guessed. UNIGEDS can't tell whether the vendor used it correctly. So we have to record its value as a media type. But when GEDCOM tries to make media type subordinate to call number... well I guess the real purpose of that is to make us think. These specs aren't really specifications; they are thinkifications.

90. The goal here is to handle several different usages of the MEDI tag and to handle each of them correctly as to what real-world data the tag is trying to convey in each case, and how the tags' values are actually related to each other in the real world.

OBJE, FILE, FORM, MEDI:

1) handle Legacy's anti-specs `SOUR.MEDI` directly subordinate to SOUR record zero-line:

```
0 @36@ SOUR
1 MEDI Email
```

A peek at the .ged file exported by Legacy's sample tree makes it obvious that MEDI here is a source_type and has nothing to do with multimedia type except when it specifies "electronic", which is too vague to be of any use. Put it in the exceptions log or ignore it. We have to stop playing

GEDCOM's silly game: GEDCOM is a text file, not a computer program that can oversee itself. It is not enforceable, and we who try to write GEDCOM import programs should not be trying to cover its tracks by making the effort to get right what GEDCOM's creators got wrong.

Even getting the right data into an exceptions log can be challenging. Not that I think this should be easy. But a real standard would not bend to each vendor's whim. A real standard would not be usable if its rules were not followed. GEDCOM's flexibility is its worst flaw. But it has to be flexible because it is wrong so often.

2) source_repository_citation

Why is a source type being linked to a locator (call number/CALN)?

```
0 @6@ SOUR
1 REPO @7@
2 CALN 13B-1234.01
3 MEDI Microfilm
```

It's hard to say here exactly how vendors are using this MEDI since they don't appear to be using it. It makes me wonder whether GEDCOM even has a source-type category. A survey of all the ways a TYPE tag can be used doesn't turn up anything like that; just a usage of TYPE wherein the TYPE tag is a synonym for MEDI. Gramps' sample tree uses the REPO.CALN.MEDI tag with placeholder data so there seems to be no intent to actually use the tag for anything specific. Kith & Kin seems to take the tag seriously as a media type such as "internet" but it's being used subordinately to a CALN (call number) tag with no call number or anything else given as a value. I don't have any other examples of REPO.CALN.MEDI.

I'll have to say it's a media type. GEDCOM's linking it to a call number at a repository seems to suggest that if there were two versions of the source--say a book and a microfilm of the book--they'd have two different call numbers. The link seems to be to a particular physical

version of the source. A given source can exist in multiple repositories, and it can exist as different copies and in different media formats in a single repository. But rather than try to figure out what CALN.MEDI might possibly be good for when contemplating it makes me fear an impending mudslide, let's see what a real GEDCOM expert has to say.

Here is Tamura Jones' suggestion for dealing with this tag in his annotation of the 5.5.1 specs:

"REPO.CALN.MEDI

There is a puzzling mistake that has, for the sake of full backward compatibility with (the not deprecated part of) the GEDCOM 5.5.1 structure, not been corrected in GEDCOM 5.5.5; the <<SOURCE_REPOSITORY_CITATION>> definition allows the CALN subrecord to have MEDI subrecord to specify a <<SOURCE_MEDIA_TYPE>>. That doesn't make sense. The media type is a property of the source (<<SOURCE_RECORD>>), not of the repository, and certainly not of the call number. Consider REPO.CALN.MEDI strongly deprecated."

The notion that some of GEDCOM's tags should be deprecated suggests that GEDCOM itself should not be deprecated. In fact, GEDCOM itself should be deprecated, used only because we are desperate for some tool that claims to import and export genealogy data. While using GEDCOM as a placeholder for a data transfer tool just for now is understandable, it has to be understood that intending to continue its use indefinitely while making little adjustments to it every so often--thus damning us all to the hell of dealing with a proliferation of versions--is not a rational plan.

I don't think any GEDCOM savant will ever agree that GEDCOM can be replaced, because once you've made the supreme effort to become a GEDCOM savant, you will want to justify having made such an effort, and who doesn't want to sound like an expert, anyway? The rest of us can just tell it like it is: GEDCOM is deprecated, period, because GEDCOM's little mistakes and mis-assumptions and wrong-headed

strategies and ill-fitting structures do not add up to something that should be fixed. They add up to something that "can't be fixed," which is a shorthand way to say that starting over from scratch would be a better use of our precious time than forever tweaking the facade of functionality that a bad tool accretes to itself just because it is so widely used as to generate the illusion that it's not a complete and total nuisance to have around.

The main reason GEDCOM is still being used is that we keep on using it. If not for that unfortunate circumstance, the void created by its absence would have pulled in any number of better GEDCOMs by now.

While Mr. Tamura Jones could never be replaced, GEDCOM certainly can be. And he's right: REPO.CALN.MEDI is a mistake.

91. The simple reason no one is getting the connection or understanding the underlying possibly good intentions of this klunky tag combo REPO.CALN.MEDI is that GEDCOM doesn't treat most of the elements of genealogy as primary elements. If every element was treated as an element by being given a unique identifier, then instead of clumsily asserting such an odd juxtaposition of supposedly subordinately related data, foreign keys could have been used to show that a group of elements are all related to each other. Without the elements of genealogy being properly separated from each other in their own records, GEDCOM has no choice but to create mud by throwing everything together in the few primary records that it does recognize.

Since GEDCOM's creators only *started* to make GEDCOM work like a database, but stopped with only seven of genealogy's primary elements being recognized as such, the cardinality or relationship type among data had to be guessed at during the rest of the process of designing GEDCOM. In particular, there's no dedicated structure to fit pointers into that have a many-to-many relationship with each other, and that's where the most indecipherable of tag chains come from, like this one.

When the members of a data pair relate to each other on a one-to-one basis, such as *source:source_title*, they just go in the same record. When a data pair such as *source:citation* has a one-to-many relationship, the pointer for the *one* side goes in the record for the *many* side. For each source, there can be multiple citations, but each citation refers to that one source, so the source pointer goes in the citation record. What we're talking about here, and the ultimate answer to this long-winded discussion, is the 3rd and last kind of cardinality, the many-to-many relationship.

This is the solution to the current discussion because *source:repository* has a many-to-many relationship. Each source can occur in multiple repositories, and each repository can contain multiple sources. GEDCOM has no dedicated way to record many-to-many relationships, so it's a perfect application for *eenie-meenie-mynie-moe* and plop the data down wherever it kinda seems like it might sorta fit. Thus the klunky, bothersome tags like SOUR.REPO.CALN.MEDI. I knew I should be ignoring this tag but it took days to figure out exactly why. Assuming that I am right, and there's no guarantee of that.

Here's how this relationship should have been done, but first I'll show the wrong way. The OBJE pointer is missing completely from this scenario, and that's just part of the problem. Since media type is "image", we need a medium (OBJE) to point to. GEDCOM substitutes a call number because the creators of GEDCOM have confused themselves by pretending there are only seven elements to genealogy.

GEDCOM's way (clumsy, wrong, unintelligible, and certifiably weird):

```
0 @S9@ SOUR
1 TITL 1900 US Census
1 REPO @RP3@
2 CALN https://www.familysearch.org/ark:/5...
3 MEDI image
```

gedMOM's way:

```

* *
SORC 9
SORC_TITLE 1900 US Census
* *
CTTN 48
SORC_FK 9
CTTN_STRG ED 145 Sheet 14B
* *
RPST 3
RPST_NAME FamilySearch.org
* *
LCTR 71
LCTR_STRG https://www.familysearch.org/ark:/619...
* *
MDIA 92
MDIA_TYPE_FK 1
MDIA_FILE 1900_census_Paris_Texas_14B.jpg
* *
MDIA_TYPE 1
MDIA_TYPE_STRG image
* *
RPST_LINKS 431
CTTN_FK 48
RPST_FK 3
LCTR_FK 71
MDIA_FK 92

```

This is a hypothetical solution. It can't be implemented because too many parts are missing to translate the GEDCOM to something that makes sense. The lack of a citation is especially disheartening, since the citation is usually what a call number links to, although in some cases a call number would link to a source, and sometimes one locator could refer to both a source and a citation. I will have to agree with Tamura Jones that this tag can't be handled. We'd have to guess what citation it's referring to, which we can't do. As you can see from the simple gedMOM version, source doesn't even link directly here, unless the call number refers to the whole 1900 census, which it doesn't. I have no choice but to send every usage of

SOUR.REPO.CALN.MEDI to the exceptions log. The exception log can have it.

92. Final verdict on the illegal INDI.MEDI tag which one or more vendors use: ignore it. There is no link between source and media type. The source (for example, a block of stone with words and numbers etched into it) must pre-exist the medium that communicates to us what the source says if we are working online and can't visit every cemetery that holds a gravestone of interest to our research. The medium is a photo or a scan of a photo, but we who try to import data from wrongly-written .ged files should not require ourselves to bend over backwards to accomodate a vendor that doesn't know the difference between a source and an image of a source. Who actually wanted to know that the picture of the gravestone is a media type of "image"? One of the secrets of completing a GEDCOM import program is to get the broad strokes done in a well-designed code base, and the easiest way to not get this done is to start twisting the design into an ever-increasingly contorted monsterization of what the code coulda/shoulda been, with each edge-case and detail of limited usefulness which is encountered treated as if it is of the utmost importance. You have to draw the line somewhere.

93. Here's an example of something GEDCOM can't do even if we follow the specs' prescriptions.

I'm writing a procedure for communicating call numbers and other locator codes to the user in the exceptions log, because CALN is a nonsensical tag that should be linked to citation and/or source but is instead linked to REPO, as if a call number either 1) has to be linked to the source that REPO is linked to, or 2) only refers to a source but not to a location within the source (a citation). Now that I'm trying to tell the user what the problem is in the exceptions log, so he can correct it manually using the Treebard interface after the import is finished, I find that it is not possible--or seemingly very difficult, which at this stage of GEDCOM burnout amounts to kinda the same

thing--to get the name of the repository that the CALN was subordinate to in the .ged file.

The reason why this happens can be explained on a practical level and in terms of data relationships, so I'll do both. The simple reason is that REPO subordinates such as REPO.NAME tend to occur later in the .ged (a text file being read line-by-line) than the SOUR.REPO pointers that refer to them. In the Python dictionary where I'm collecting data as the lines are read, the REPO.NAME hasn't been encountered yet, when the REPO.CALN exception needs to be recorded in the `exceptions` dictionary. To overcome this problem, something that I don't want to do at all because no one uses the tag as far as I know would have to be done earlier in the process, on an earlier read-through of the file, forcing me to pollute clean code with edge-case blather.

The database explanation is that GEDCOM has no facility for handling many-to-many relationships. This is because it only has a handful of primary tags, so the pointers available don't meet the needs of a real database which uses foreign keys (the equivalent of GEDCOM's pointers) for every datum that's repeated anywhere in the database. In a pretense at linking a call number to something vaguely relevant, GEDCOM does this:

```
0 @SRC9@ SOUR
1 TITL 1900 US Census
1 REPO @RP3@
2 CALN https://www.familysearch.org/ar...
```

And later on, GEDCOM does this:

```
0 @RP3@ REPO
1 NAME FamilySearch.org
```

The solution (to just do it right) is beyond the scope of GEDCOM, so here is the gedMOM solution, wherein locators and citations also have primary status, not just repositories and sources and multimedia. Since gedMOM is also a gedcomoid (a text file read linearly), as long as

records are written and/or read in the right order, the existence of the needed foreign keys makes cross-referencing data within the .mom file as easy as eating apple pie with homemade vanilla ice cream on top of it. This is because the needed foreign keys do in fact exist.

```
SORC 9
SORC_TITL 1900 US Census
* *
RPST 3
RPST_NAME FamilySearch.org
* *
LCTR 19
LCTR_STRG https://www.familysearch.org/ark:/61903/3:1:S...
* *
RPST_LINKS 66
SORC_FK 9
RPST_FK 3
LCTR_FK 19
```

Unlike GEDCOM specs, reality demands that many-to-many relationships be handled specifically and on purpose, as the many-to-many relationships that they are. Since a repository can contain multiple sources and each source can be found in multiple repositories, the junction table or many-to-many table `repository\_links` has a foreign key (pointer) for both source and repository in the same record (table row).

But even the gedMOM example above is dumbed-down to match GEDCOM, which is wrong. Sure, a locator can be linked to a source, for example a book in a library has a locator, a call number. But the reference above should be to a citation within a greater source, not the whole source, as well as a multimedia object which represents that citation as a computer file. Below is the full representation of the data which GEDCOM only hinted at vaguely. Don't mistake this level of clarity for complexity. Undue complexity is when data appears in the wrong place for the wrong reason, or when any given datum must be repeated in order to be

used. What's complex is trying to squeeze meaning out of GEDCOM's *stuff it anywhere* attitude.

In order to link events--which are really the user's conclusions about events--to sources, we can't leave out the *assertion* element: what the source says. The genieware user is not required to link sources to conclusions, but without assertions, it's not possible to do right. There's no other data-relationship-respecting way to link conclusions (events, names, and attributes) to sources, and until computer genealogy admits assertions into its pantheon, computer genealogy is doomed to remain the spawning ground for vagueness, assumptions and other inaccuracies that it currently is.

In case there are any mistakes in the example below, let me say that the notion of gedMOM providing data to a UNIGEDS database has been tested and has passed the test with flying colors. The below fragment looks good to me but it hasn't been tested. gedMOM is not being used in my actual import process--I'm using Python dictionaries instead--but it's a perfect teaching tool so wanna-be creators of genieware and GEDCOM import programs can look at what a database does instead of looking at a database.

Besides offering a GEDCOM export tool, Treebard might someday also offer a gedMOM export tool. Of course, if all vendors used UNIGEDS to store their primary genealogy data, export tools wouldn't be needed; you'd just share the Treebard file, which is a SQLite database.

```
PRSN 12
* *
EVNT_TYPE 13
EVNT_TYPE_STRG residence
* *
EVNT 29
PRSN_FK 12
EVNT_TYPE_FK 13
* *
SORC 9
SORC_TITL 1900 US Census
```

* *

CTTN 45

CTTN_STRG Precinct 2, Trevor County, Texas, E.D...

SORC_FK 9

* *

ASRTN 25

ASRTN_DETL resident of Trevor County on 14 April 1900

CTTN_FK 45

EVNT_FK 29

* *

RPST 3

RPST_NAME FamilySearch.org

* *

LCTR 19

LCTR_STRG <https://www.familysearch.org/ark:/...>

* *

MDIA 83

MDIA_FILE 1900_prct2_trevor_co_tx_ed74_p41b.jpg

* *

RPST_LINKS 66

CTTN_FK 45

RPST_FK 3

LCTR_FK 19

MDIA_FK 83

Reserved for Future Comments About GEDCOM

94. Add to the converstion at <https://treebard.proboards.com>

Reserved for Comments About UNIGEDS, gedMOM, or Treebard

95. Let me know what you think at <https://treebard.proboards.com>

EPILOGUE

Every time I see the phrase "the GEDCOM standard", I want to toss my cookies.

Genealogy doesn't have one standard, it has many. Some of these might be: historical accuracy, a level of detail that matches real-world circumstances, and absence of sarcastic smarty-pants loudmouths like me (just kidding there; I'm

actually OK till you get to know me). Nobody decided, one day, that genealogy was seeking some pretending-to-be-software tool to serve as a standard.

This phrase "the GEDCOM standard" came from where all misnomers come from: Santa Claus's elves. Someone said "the GEDCOM standard" one day because it made him sound smartly professionalistic--like engineers and surgeons and nuclear power plant supervisors, who have plenty of important standards to enforce--and it caught on, because the bull geese genealogists, understandably enough, are not only financially motivated to sound smart; it even fits their personality to sound fairly intelligent. Professional genealogists tend to be smart, detail-oriented people who care a lot about the things they care about. Nothing wrong with that.

But in these days of phony personas easily generated online, trying to sound sophisticated and become internet famous is a trap that many are falling into, often not aware they're doing it. Heck, you used to have to buy expensive ads in the backs of magazines to accomplish what you can now do way better for free with an online presence. Sounding sophisticated to people who will never actually meet us is pretty easy for anyone who can manipulate the turn of a phrase and blather around with a bit of fancy-sounding lingo.

The cloaking nature of the internet fuels this tendency to build an image of oneself which is actually a rickety, ramshackle house of cards. I'm not accusing anybody of anything; I've learned about this by spending too much time on Facebook years ago and making the mistake myself. When I deleted my Facebook account, I soared free of the two-dimensional image I had become, and it freed up the time that soon became the Treebard project.

I have nothing bad to say about those who are pretending to replace GEDCOM, nothing bad at all. Be that as it may, we

do in fact need a new tool for transferring data among genealogy applications, whether I'm a smarty-pants loudmouth or not. This tool has to reflect the real standards of genealogy. In the end, genealogy itself is our standard, and we need tools that measure up to the practice of genealogy, an art which is even more ancient than my bad jokes.

This book is not perfect and is not meant to be. It should be good enough to start a productive conversation or two. I hope its blind spots and mistakes don't cause its main points to be completely missed, due to the resistance to change which is built into the human mold.

Genealogy deserves better than GEDCOM and genealogy will get better. We can make it happen now or we can wait another 40 years. It's your decision, not mine.

Sincerely,

Professor U. d'Guru writing as Uncle Buddy writing as
Luther Rangely writing as Donald Scott Robertson

January 2, 2024
Southern Mindanao



GEDCOM RESOURCES

<https://www.gedcom.io>

<https://www.gedcom.org/gedcom.html>

<https://www.beholdgenealogy.com/blog/?p=1074>

<https://archive.fhiso.org/BetterGEDCOM/>

<http://www.gaenovium.com/presentations/2014/Gaenovium%202014%20-%20Louis%20Kessler%20-%20Reading%20Wrong%20GEDCOM%20Right.pdf>

<https://www.beholdgenealogy.com/forum/>

<https://www.fhug.org.uk/kb/kb-article/gedcom-extension-list/>

https://www.legacyfamilytree.se/WEB_US/user_defined_gedcom_tags.htm

AUTHOR OF UNIGEDS, gedMOM, Elucidom, and Treebard GPS

<https://treebard.proboards.com>

<https://treebard.com>