Plumbing the kitchen sink — The Tcl Standard Library

Andreas Kupries Hewlett-Packard Enterprise 1700-409 Granville V6T 1C2 Vancouver, BC, Canada
andreas.kupries@hpe.com

ABSTRACT

This paper is part retrospective, part explanation, part discussion, and all Tcllib. Plumbing the depths of the Tcl Standard Library it describes where Tcllib came from, what forces were and are shaping it, what it currently looks like, and what might be in the future.

1. INTRODUCTION

Tcllib, also called the Tcl Standard Library, seems to be relatively well-known at its about 14.5 years of age. The full set of functionality it provides on the other hand seem to be less less so. Nor how to navigate its site and/or structure to determine if some sought-for functionality is present or not.

In this paper we will inspect first how it came to be, the forces which shaped it over the years, and the larger context of its existence.

After that we will discuss at a high level the currently provided functionality, and, more importantly, what tools are available to locate functionality within. This is followed by information abouts its internal structure, supporting tools, the (not so) mandated parts a package, relevant file formats, and utility commands.

At the end we pose some questions for the future, how we might be able to solve Tcllib's current problems, and directions it we could go into.

2. PLUMBING THE PAST

The first recorded commit, as per the history found in the current fossil repository was done on Feb 24, 2000, by Eric Melski, employed by Scriptics. The first package was a pure Tcl "profiler", one file implementing it, another file holding the testsuite, and 9 more files for a standard buildsystem of "configure", "Makefile", etc.

It was this high overhead for pure Tcl packages which, I believe, generated the desire to create a standard library which would hold more than just one package, to amortize this overhead across many.

From these small beginnings in the next 15 years each of the 19 releases added ever-more modules and packages. Their exact details can be found in table 1.

Not shown in the table are the locations where Tcllib was hosted over this time. First at Scriptics, I believe, then at Sourceforge (`https://sourceforge.net/p/tcllib`), and since about 2012 on the same machine as the Tcl core [2] itself (`https://core.tcl.tk/tcllib` [1]).

As a side note, the aforementioned "modules" are groups of similar and related packages, used to keep things organized and manageable.

Yet even with that aid and about 85% of all packages documented Tcllib has become a veritable kitchen-sink where even people who know of Tcllib are losing track of just what is all provided. As Clif told me "For me, tcllib is a lot like the hall closet - lots of stuff, but I forget what's been put there and how to find it under all the winter coats and left-hand-mittens."

This leads us to a question more and more asked when a new package or functionality is contributed: "Could this not be handled through its own package ?". And the fundamental question hidden behind that is "Do we still wish to expand ?".

And with question in our mind, let us go to the next section, which will talk in more detail not only about the content currently available, but more importantly also the tools we have to navigate the onslaught.

3. PLUMBING THE CONTENTS

3.1 Location

The current location of Tcllib is `https://core.tcl.tk/tcllib`. With one exception everything for Tcllib is there, i.e. the source repository, online documentation, ticket tracking, wiki pages.

The exception are the two mailing lists `tcllib-devel` (dev discussions) and `tcllib-bugs` (ticket notifications). The lists are still managed by and located at SourceForge. The links to the list archives, subscription pages, etc. however can be found at the url above.

Version	When	Modules	Packages
0.4	2000-04-26	10	11
0.5	2000-06-09	11	12
0.6	2000-07-19	14	15
0.6.1	2000-08-22	14	15
0.8	2000-11-02	18	19
1.0	2001-07-17	24	25
1.1	2001-10-17	26	27
1.2	2002-01-18	31	42
1.3	2002-06-10	34	48
1.4	2003-05-06	38	61
1.6	2004-02-16	43	67
1.6.1	2004-05-23	43	67
1.7	2004-10-06	56	117
1.8	2005-10-07	68	184
1.9	2006-10-04	77	231
1.10	2007-09-12	81	241
1.11	2008-10-17	85	254
1.11.1	2008-12-15	87	260
1.12	2009-12-07	95	333
1.13	2011-01-25	103	377
1.14	2011-12-13	107	398
1.15	2013-02-06	110	405
1.16	2014-02-11	115	416
1.17	2015-04-30	119	422
current	2015-09-03	122	432

Table 1: Releases and their statistics

3.2 Functionality

Table 2 gives us an overview of the spread of functionality which can be found in Tcllib, placing the (currently) 122 modules into 17 categpories.

An important thing which many are missing is that Tcllib does contain pieces of C-code, all based on `critcl` [3]. They are allowed because they are optional, i.e. all the packages will work without them installed. However having them built and installed provides the using packages with a speed-boost. This is also the reason why these parts are often called “accelerators”. See table 3 for the details. Note that some of the packages can also make use of the separate package `Trf` as their accelerator.

For the statistically inclined, some numbers.

As expected, we have a minimum of 1 package per module. The maximum on the other hand is 41 packages in a module. The mean is 3.54 with a standard deviation of 64.78. The median is 7.

I believe that these skewed statistics are there mainly due to a few of what I call “super-modules”, which contain a very high number of related packages. These are `math, struct, pt` and its predecessor `page,` plus the `doctools*` modules.

These mere 8 modules (7%) combined contain 164 packages (38%).

Taking them out of the statistics our numbers change to 285 packages in 113 modules. The maximum falls down to 15. Mean and standard deviation move down to 2.5 and 26.31 respectively. The median goes to 4.

3.3 Navigation

An important tool for the navigation and inspection of the modules and packages provided by Tcllib is its website at `https://core.tcl.tk/tcllib`, served by `fossil` [5]. It’s pages are generated by both `fossil` and the documention (tools) of Tcllib itself.

The main feature used from `fossil` is its ability for “embedded documentation” where the integrated web server directly accesses artifacts stored in the repository. Tcllib uses this to publish HTML pages and a number of tables of contents and indices generated from the manpages for its modules packages.

These adjunct pages (Table of packages, modules, applicaton, and index of keywords) are the main tool for finding specific functionality within Tcllib. All of them are accessible from the main “Documentation” link in the toplevel navigation bar of the Tcllib website.

Alternatively, when using either an unpacked distribution tarball or repository checkout the documentation tree is found in the directory `embedded/,` with sub-directories for the HTML mentioned above, and for regular `roff-based` manpages.

The main directories for the sources on the other hand are `modules/, apps/,` and `examples/.` The difference between the last two is that the examples are more fragmentary and not polished for proper usability. It should be noted that the examples are usually **not** installed, especially not by the various OS distributions which may carry Tcllib (OpenBSD is a notable exception). ActiveTcl does not install them either. To see and play with them either a checkout or a source distribution of Tcllib are needed.

Category	Modules
Crypto	aes, blowfish, des, pki, rc4
Data structures	cache, struct, tie, treeql
Dates & Times	calendar, clock
Debugging	debug, log, profiler
Documentation	doctools, dtplite
Encodings	asn, base32, base64, bee, crc, json, yaml
File formats	bibtex, csv, docstrip, exif, gpx, inifile, jpeg, png, rcs, nmea, tar, tiff, zip
Files & Channels	fileutil, fumagic, virtchannel_
Hashes and checksums	md4, md5, md5crypt, ripemd, otp, sha1, soundex
Language extension	control, coroutine, generator, hook, interp, lambda, namespacex, ooutil, pluginmgr, snit, stooop, try, uev
Math	math, units
Network & Communication	amazon-s3, comm, dns, ftp(d), http(d), ident, imap4, irc, ldap, mime, multiplexer, nettool, nns, nntp, ntp, oauth, pop3(d), rest, sasl, smtpd, transfer, websocket
Parsing	grammar_, page, pt, wip
Performance	bench
String manipulation	report, string, stringprep, textutil
Web processing	html, htmlparse, javascript, markdown, ncgi, scgi, uri
Miscellaneous	cmdline, counter, cron, map, mapproj, processman, simulation, tepam, term, uuid, valtype

Table 2: Content categories

Module	Package	Users
base32	tcllibc	base32, base32::hex
base64	tcllibc, Trf	base64, uuencode, yEnc
crc	tcllibc, Trf	crc, crc::sum, crc32
dns	tcllibc	ip
json	tcllibc	json
md4	tcllibc	md4
md5	tcllibc, Trf	md5
md5crypt	tcllibc	md5crypt
pt	tcllibc	pt::rde
rc4	tcllibc	rc4
sha1	tcllibc, Trf	sha1, sha256
struct	tcllibc	struct::graph, struct::queue, struct::sets, struct::stack, struct::tree
uuid	tcllibc	uuid

Table 3: Accelerators and users

The `support/` directory contains the helper code and files for the `sak.tcl` tool which is of no interest to plain users of Tcllib, only to maintainers and developers. More details about it will be found in the next section, **??**.

A plain user is likely more interested in either `installer.tcl,` a pure Tcl installer application for Tcllib, or the more conventional build-system consisting of `configure,` `Makefile,` and the `config/` helper directory. Note however that this conventional system is just a wrapper around the `installer.tcl.`

3.4 Contribution

Despite the size of Tcllib even beginners can contribute. Simply start with documentation, and/or testsuites.

The following subsections will provide the necessary information to get started, i.e. how documentation, testsuites and benchmarking are handled for the beginner, and how to add a new module/package, for the more advanced.

The main tool for all is `sak.tcl.` Use this application whenever documention, etc. have been written to check that everything is ok. Note that it has a basic integrated help system accessible via `sak.tcl help` and `sak.tcl help $thecommand.`

First ...

3.4.1 Get The Sources

The location of the Tcllib sources was already provided, in section 3.1.

The easiest way of retrieving a single specific revision is to:

- Log into the specified site, as "anonymous", using the semi-random password in the captcha.
- Go to the "Timeline".

- Choose the revision you wish to have and follow its link to its detailed information page. On that page, choose either the "ZIP" or "Tarball" link to get a copy of this revision in the format of your choice.

For a developer-to-be however it is better to install the `Fossil[5]` SCM for their platform and to then run the commands of listing 1 to get a checkout of the head of the trunk.

Listing 1: Clone Tcllib

```
fossil clone  http://core.tcl.tk/tcllib tcllib.fossil

mkdir tcllib
cd    tcllib
fossil open ../tcllib.fossil
```

3.4.2 Documentation

Remember, a package without user documentation is a bad thing.

Forcing the user to read the code to understand how to use a package is not a good experience for said user. Currently 61 packages, about 15% of the total are without documentation. This is a good point to start for anybody wishing to contribute to Tcllib.

The standard format for documentation used by all (documented) packages of Tcllib is called "doctools", with adjunct formats for tables of contents ("doctoc") and keyword indices ("docidx"). The packages needed to process these formats are part of Tcllib itself.

Introductions to the system and the main file format can be found online at `https://core.tcl.tk/tcllib/doc/trunk/embedded/www/tcllib/files/modules/doctools/doctools_intro.html` and `https://core.tcl.tk/tcllib/doc/trunk/embedded/www/tcllib/files/modules/doctools/doctools_lang_intro.html` respectively[1]. The latter especially is based around a series of examples introducing all the available features. Beyond that all existing documentation files (with extension `.man`) can serve as examples. A simplified example can be seen in listing 2.

Listing 2: Simple manpage

```
[manpage_begin csv n 0.8]
[see_also matrix queue]
[keywords csv matrix package queue tcllib]
[copyright {2002-2013 Andreas Kupries <andreas_kupries@users.sourceforge.net>}]
[moddesc   {CSV processing}]
[titledesc {Procedures to handle CSV data.}]
[category  {Text processing}]
[require Tcl 8.4]
[require csv [opt 0.8]]
[description]

The [package csv] package provides commands to manipulate information
in CSV [sectref FORMAT] (CSV = Comma Separated Values).

[section COMMANDS]

The following commands are available:

[list_begin definitions]

[call [cmd ::csv::joinmatrix] [arg matrix] [opt [arg sepChar]] [opt [arg delChar]] [opt [arg delMode]]]

Takes a [arg matrix] object following the API specified for the
struct::matrix package and returns a string in CSV format containing
these values. The separator character can be defined by the caller,
but this is optional. The default is a comma (","). The quoting
character can be defined by the caller, but this is optional. The
default is a double-quote.

[list_end]

[vset CATEGORY csv]
[include ../doctools2base/include/feedback.inc]
[manpage_end]
```

After the documentation is written use `sak.tcl doc validate $themodule` to check it for syntactical correctness. Always run this command before committing any new or modified documentation. When everything is found to be correct use the

[1]The part of the path starting with `embedded` is the path in the checkout, for local reading

command `sak.tcl localdoc` to regenerate the embedded documentation. That operation may take a while, depending on the capabilities of your machine, as it will process all documentation in Tcllib. It currently has no ability to incrementally detect and update only changed documentation.

3.4.3 Testsuites

Packages should have testsuites. While they are secondary to documentation in that packages can be used without them, their absence still is a glaring hole in the defense against any type of bugs and therefore something to be avoided.

Tcllib is unfortunately much worse regarding them than for documentation. About 44% of all packages are without tests.

Those that have them are based on `tcltest`, plus additional helper commands supplied by `modules/devtools/testutilities.tcl`, giving them a semi-standard format. A simplified example can be seen in listing 3.

Listing 3: Simple testsuite

```
# -*- tcl -*-
# ---------------------------------------------------------------------------

source [file join \
        [file dirname [file dirname [file join [pwd] [info script]]]] \
        devtools testutilities.tcl]

testsNeedTcl      8.3
testsNeedTcltest  2.0

support {
    use struct/queue.tcl   struct::queue
    use struct/matrix.tcl  struct::matrix
}
testing {
    useLocal csv.tcl csv
}

# ---------------------------------------------------------------------------

test csv-1.7 {split on join} -setup {
    set x [list "\"hello, you\"" a b c]
    # csv 0.1 was exposed to the RE \A matching problem with regsub -all
} -body {
    ::csv::split [::csv::join $x]
} -cleanup {
    unset x
} -result [list "\"hello, you\"" a b c]

# ---------------------------------------------------------------------------

testsuiteCleanup
return
```

All `.test` files in Tcllib should follow the standard set by the example. It is **strongly** recommended to report `.test` files which don't follow that standard.

Testsuites are stored in files with extension `.test`, sibling to the corresponding `.tcl` files. All existing test files can serve as examples.

After the tests for a package are written use `sak.tcl test run $themodule` to execute them, and collate their results. Always run this command before committing any changes to of a package's code.

A better command, actually, is `sak.tcl test run -l X $themodule`, which will save test results into a collection files with the common prefix `X.`, allowing proper analysis of any failures found.

The output generated by `tcltest` is the standard format for test results within the Tcl community. I believe that the wider OSS community has a different standard format, the details however are not known.

3.4.4 Benchmarking

Only very few packages have benchmark suites, i.e. code to measure their performance. Only 33, i.e. about 7% of the total. That said, not many packages actually are performance-sensitive and in need of benchmarking.

Those packages that have them are use the `bench` package in Tcllib itself, giving them a semi-standard format, although much looser than for the testsuites. There are no additional helper commands available. A simplified example can be seen in listing 4.

Listing 4: Simple benchmarks

```
# -*- tcl -*-
# Tcl Benchmark File

if {![package vsatisfies [package provide Tcl] 8.2]} {
    return
}

# ### ### ### ######### ######### ######### ###########################

set moddir [file dirname [file dirname [info script]]]
lappend auto_path $moddir

package forget rc4
catch {namespace delete ::rc4}
source [file join [file dirname [info script]] rc4.tcl]

set i [binary format H* 0000000000000000]
set p [binary format H* 0123456789ABCDEF0123456789ABCDEF]]

set k [binary format H* FEDCBA9876543210]
set c [binary format H* ED39D950FA74BCC4ED39D950FA74BCC4]

# ### ### ### ######### ######### ######### ###########################

bench -desc "RC4 encryption" -body {
    rc4::rc4 -key $k $p
}

# ### ### ### ######### ######### ######### ###########################
```

Benchmarks are stored in files with extension `.bench,` sibling to the corresponding `.tcl` files. All existing benchmarks can serve as examples.

Use `sak.tcl bench $themodule` to execute the benchmarks for a module, and collate their results.

There are three additional commands, `bench/show,` `bench/edit,` and `bench/del,` to show, reformat and manipulate result files.

If there is a standard format for benchmark results, either in the Tcl or in the larger OSS community, it is not known.

3.4.5 Adding modules and packages

While `sak.tcl` can automatically deal with a new module and/or package, as long as it is a directory under `modules/` the `installer.tcl` application has no such.

To make a new module known to the installer it is necessary to edit the file `support/installation/modules.tcl.` Its main part is a series of `Module` statements which declare the modules which should be known to the installer. The four arguments in each statement are first the name of the module, followed by the names of the commands to run when to install the packages of the module, the documentation, and related examples.

The standard commands for packages without special needs are `_tcl,` `_man,` and `_exa.` A module without documentation or examples can also use `_null` to declare such. The implementations of these and their variants used by special-need modules can found in `support/installation/modules.tcl`

With all of the above the checklist for creating a new module with package now looks like this:

- Create a new sub-directory `foo` of `modules/.`
- Place the package implementation into `foo,` under whatever name `bar.tcl` you would like.
- Create a `pkgIndex.tcl` file with a proper `package ifneeded` statement.
- Create package documentation as per section 3.4.2. Do not forget to validate the syntax of said documentation.
- Create a package testsuite as per section 3.4.3. Do not forget to validate the testsuite. Do not forget run it either, and then fix any problems found by it in either package or testsuite itself.
- If it makes sense, create a benchmark for the package, as per section 3.4.4.
- Update and extend the installer as decribed above.
- Add any new files to fossil (`fossil add),` then `fossil commit` the change. Be sure to use the option `--branch` to place the change into its own branch.

3.4.6 Change submission

Whether a change contributes a full module/package, or " just" fixes for documentation, testsuites, etc., at the end we have a commit i.e. new revision in the **local** clone of Tcllib.

Submission of a revision always begins with Tcllib's ticket system, reachable through the front page, and creating a new ticket which describes the change.

If the repository containing the change is publicly visible on the internet then simply put a link to the revision into the ticket. The maintainers of Tcllib can then pull from the contributing repository and start the evaluation.

If the repository is private then the best way is to use the `fossil bundle` ensemble to export the change and supporting revisions into an archive file which in turn is attached to the ticket. Note, attachments can only be made after the ticket is created, not during the creation.

4. PLUMBING WHAT MIGHT BE

Before looking into the future, let us recap a bit.

Starting from very small beginnings Tcllib now contains 432 packages across 122 modules. Impressive, but only if you are subscribing to the credo of "more is better". Otherwise, what began as an attempt to amortize the overhead needed to manage a pure-Tcl package has now moved to the other extreme, a kitchen-sink. A broad range of functionality, a jack of all trades, without true focus, giving potential users trouble to determine if what they need is part of it or not. Writing your own code becomes the easier route.

What can we do about that ?

- We have documentation, even for most of the packages (85%).
- Can we make this more visible ?

  We already link to it from both the homepage itself, and the navigation bar shown on each page presented by the repository.
- As a side note, is there anybody with ideas for a Tcllib logo ? And the ability to execute such into an image proper ? Or knowing somebody who can ?
- Might some sort of blog help ?

  Maybe use timeline technotes for short intros, and then a wiki page for the full article. And as these are stored in the repository everybody will have access, even offline, if they have a clone.
- More radical changes ?
- For example, splitting out focused pieces of Tcllib into their repository ?

  We would still amortize most the overhead, whether by keeping a copy of the sak system, or by going to `kettle[4]`. The massive modules mentioned before would easily lend themselves to that.
- Then the C parts. Currently the step-children of Tcllib's build system, hindering their uptake.

  Should we make them proper packages inside of Tcllib, i.e. giving up on Tcllib's restriction that the pure-Tcl parts must be functional ?

  Should we move them into separate packages ? Again using `kettle[4]` would allow for a low-overhead build system. We might however have to discuss again what a good architecture is for packages with multiple possible implementations.
- What about easing contributors into the system with simpler tasks ?

  For example writing documentation, tests, and benchmarks.

  These could be helped by providing information on the website about which packages need documentation, tests, etc. Plus proper templates demonstrating the various standard formats and utilities for these. While the listings found in the previous section are a step into that direction, they are also much to small.
- Could a document providing general information on the workflow to follow for contribution, with a check list of actions, help too ?
- The two previous items could of course be expanded into a full template of a package in its own module on which contributors can base their work on.

  And thus we circle back to the question posed at the end of our look into the past beginning this paper: "Do we still want to expand" ?

We have no answer to this, yet. Although I personally keep tending towards no, we should not. This however may be because I am the person the maintenance work keeps falling on, and I do not scale.

An interesting counter point was made by Clif Flynt, when he and I discussed the draft. He saw Tcllib in the line of archives like `Neosoft[6], Alcatel, Procplace,` as something which came to be for the community to have a central place where packages could be collected instead of having to find and then email package authors and ask for a copy of their sources. With the difference that Tcllib set a higher standard, requiring documentation or tests. Being more restrictive. Another difference, to me: In Tcllib we many packages sharing one build system. In the other, more traditional, archives each package came with their own. If any.

From that point of view, Tcllib should clearly keep expanding. Keep being a (or going to be the) one-stop shop and archive to find functionality in an easy to install bundle.

APPENDIX

A. REFERENCES

[1] Various, Tcllib. `https://core.tcl.tk/tcllib`

[2] Various, Tcl Core. `https://core.tcl.tk/tcl/`

[3] Andreas Kupries, Critcl `https://github.com/andreas-kupries/critcl/`

[4] Andreas Kupries, Kettle. `https://core.tcl.tk/akupries/kettle/`

[5] Richard Hipp et. al. `https://fossil-scm.org/`

[6] Various, Neosoft, Alcatel, Procplace `ftp://ftp.tcl.tk/pub/tcl/`