Tcl Improvement Proposals: TIPs 0–37

The Tcl Community

22nd June 2001

Contents

List of Figures

TIP #0: Tcl Core Team Basic Rules

TIP #0:	**Tcl Core Team Basic Rules**
Author:	John Ousterhout ⟨ouster@interwoven.com⟩
Created:	Monday, 11th December 2000
Type:	Process
State:	Final
Vote:	Done
Version:	`$Revision: 2.3 $`
Post-History:	

Abstract

This TIP describes the mission, structure, and operating procedures of the Tcl Core Team (TCT). When in doubt about how the TCT works, consult this document as the final authority.

0.1 Introduction

The Tcl Core Team is a self-organizing group of Tcl experts who are responsible for the evolution and management of the Tcl core. The Tcl Core Team decides what goes into releases; it implements, tests, and documents new features and bug fixes; it manages the release process; and it also manages the Tcl Developer Exchange Web site.

0.2 Scope: the Tcl core

The phrase "Tcl core" refers to the Tcl interpreter and the Tk toolkit (the packages previously released by Sun, Scriptics, and Ajuba). We also include the Tcl Developer Exchange Web site and the Tcl bug database in the Tcl core. The Tcl Core Team may also choose to take on additional responsibilities such as the creation of more comprehensive "batteries included" releases. We expect other Tcl development teams to form independently from the Tcl Core Team to manage additional projects, such as popular extensions. The Tcl Core Team may eventually spin off some of its activities into separate teams.

0.3 Team membership

The Tcl Core Team is a small group of people who are making major contributions to the development of the Tcl core and who are highly respected and trusted by the Tcl community. Team members are expected to invest significant amounts of their time to improve the Tcl core.

The original group of Team members was elected by the Tcl community, but the TCT now handles its own membership according to rules described here. To become a member of the Team you must be nominated by an existing member and voted on by the existing Team; you must receive 2/3 of the votes cast. If you would like to join the Tcl Core Team, you should first demonstrate your development skills and leadership by participating in development projects under the auspices of an existing team member.

Inactive or disruptive members of the team can be removed by a vote of other Team members: a 2/3 majority of those voting is required to remove a Team member.

0.4 Communication

The primary mechanism for communicating with the Tcl Core Team is the mail alias tcl-core@lists.sourceforge.net. This is a public mailing list; anyone interested in following the discussions of the TCT is welcome to join the mailing list. Email sent to this alias is archived, so you can review previous discussions at SourceForge.

0.5 Basic organizational structure

The team structure is simple and flat. All members have equal standing: there is no Chairman. The Tcl Core Team makes its own rules and chooses its own members as described in this document. Anyone on the Tcl Core Team can propose a change in the rules; after discussion, the change is voted on by the Team and must receive 2/3 of the votes cast. The person proposing a rules change is responsible for making sure that the change is properly implemented after it has been approved (e.g. by modifying this TIP, creating additional tools, etc.).

0.6 2/3 vote

Wherever a 2/3 vote is called for in this document, it means that a proposal must receive *at least two-thirds of the votes cast*, not votes from at least two-thirds of all TCT members.

0.7 Projects and maintainers

Tcl improvements are organized around two key ideas: *projects* and *maintainers*. Most of the activities of the Tcl Core Team consist of projects. A project can consist of a bug fix, a new feature in the Tcl core, a new facility in the Tcl Developer Exchange, or anything else except a change to this TIP. We divide projects into two general categories: bug fixes and feature changes. In general, if a project requires manual entries to be updated then it is a feature change; when in doubt, a project is a feature change . Bug fixes use a more streamlined process for implementation, whereas feature changes require discussion and approval in advance.

A maintainer is someone who has taken primary responsibility for a portion of the Tcl sources. Many maintainers will be members of the Tcl Core Team, but the Team may also select maintainers from outside the Tcl Core Team. We hope to find enough maintainers to cover all of the Tcl sources, but we will appoint a *default maintainer* to handle the parts of Tcl for which no other maintainer has volunteered. We'll also try to have backup maintainers who can step in when the primary maintainers are on vacation or otherwise unavailable.

A maintainer accepts several responsibilities, including the following:

- Monitoring the bug database for bugs in his/her area.
- Arranging for bugs to be fixed, either by doing it himself/herself or finding someone else to do it.
- Coordinating and reviewing all modifications to his/her area.
- Providing assistance to other people working in his/her area.

0.8 Project life-cycle: approval, implementation, integration; TYANNOTT

The project for a feature change goes through three stages: approval, implementation, and integration.

A project starts when a member of the Tcl Core Team proposes it to the Team. Proposals are submitted by emailing TIPs (Tcl Improvement Proposals) to the Tcl Core Team. The format of TIPs is described in a separate TIP. Whoever proposes a project is responsible for making sure it is properly implemented. A proposal without a committed implementor cannot be approved.

Project approval is done through a process called *TYANNOTT*: Two Yesses And No No's Or Two Thirds. In order for a project to be approved it must have support from at least one other member of the Tcl Core Team besides the proposer. Once a project has been proposed and discussed, if there are no objections and there is a vote of confidence from a second team member ("Two Yesses And No No's"), then the project is approved. If objections remain after the discussion, then the proposer must summarize the objections and call for a vote of the TCT; a 2/3 vote is required for approval. The idea here is that most projects will be no-brainers and we want a simple decision process that doesn't get in the way of progress. On the other hand, the Tcl Core Team can only work effectively if it is highly collegial; if the Team can't reach pretty clear agreement on a project (i.e more than 1/3 of the TCT objects to it) then the project needs to be rethought.

The second phase of a project is implementation. The proposer is responsible for the implementation, either doing it himself/herself or arranging for someone else to do it. The implementation is done in a private work area and may not be integrated with the official sources until the third phase, below.

The third phase of a project is integrating the results back into the official Tcl repository. This is where maintainers come in. First, before any change can be applied to the official Tcl sources, the implementor must post it as a patch to the SourceForge patch manager. This rule applies regardless of the type of change (anything from a 1-line bug fix to a major new feature) and regardless of who is proposing the change. We use the SourceForge patch manager to record all changes and also to facilitate discussion about the changes before they are applied.

When a patch arrives in the SourceForge patch manager, the appropriate maintainer reviews it and works with the proposer to revise it as necessary. Other people can also review the patch, since it is public. If changes are needed, a revised patch is logged in the patch manager (the final version of the patch must always appear in the SourceForge patch manager). Once the maintainer is satisfied with the patch, it can be applied to the Tcl sources. If the patch implementor has write access to the sources that he or she can apply the patch once the maintainer has approved it. If the patch implementor doesn't have write access to the sources than the maintainer applies the patch.

Maintainers are responsible for watching the SourceForge patch manager to make sure that incoming patches in their area are dealt with quickly.

If the implementor of a patch is the maintainer, then he/she can apply the patch to the Tcl sources immediately after logging it in the SourceForge patch manager, without waiting for additional approval. However, if someone objects to the patch then the maintainer must be prepared to revise it after the fact.

0.9 Fast path for bug fixes

For a bug fix, no initial proposal or approval is required. The only approval needed is for the maintainer to review the patch before it is applied to the sources. For example, we invite everyone in the Tcl community to fix bugs and submit patches to the SourceForge patch manager.

0.10 Implementors outside the Tcl Core Team

We encourage people outside the Tcl Core Team to get involved with Tcl development. For example, anyone can submit patches for bug fixes. It's also fine for someone outside the Tcl core team to propose a feature change and then implement it, but there must be a sponsor on the Tcl Core Team who will take personal responsibility for it. Typically the sponsor will be the maintainer for the area of the change. It is the sponsor's responsibility to provide whatever level of supervision is appropriate to ensure that the project is executed well. If the implementor for a project is not a TCT member then they cannot vote for approval: TYANNOTT requires the sponsor plus one other Team member.

0.11 Raising concerns

If you have concerns about a project, the best time to raise them is during the initial discussion. Once a project has been approved, the best approach is to raise the issue directly with the implementor; most issues should get resolved quickly this way. If you can't find the implementor or can't reach agreement, and if the implementor is not a member of the Tcl Core Team, the next person to talk to is the Tcl Core Team member in charge of the project. If you still can't get satisfaction, then raise the issue with the entire Tcl Core Team by leading a discussion. Once all the issues are out, you can either withdraw your objection or summarize the issues (on both sides!) and call for a vote. If you aren't a member of the Tcl Core Team you will need to convince a Team member to manage the discussion and vote.

Even if a project has received initial approval, a Team member can object to the project later (e.g. if they believe it hasn't been implemented properly). If the objection isn't resolved there will be an additional vote of the Team, and the project cannot be applied to the official sources unless it receives a 2/3 majority of the votes cast. At the same time, Team members are expected to raise their objections as early as possible; it would be somewhat anti-social to raise a basic design objection late in the implementation of a project when it could have been raised during the initial approval.

0.12 Disagreements over patches

Normally, patches are not reviewed by the entire TCT; once the relevant maintainer has reviewed and approved them then they can be integrated. However, everyone is invited to review as many patches as they wish. If someone on the TCT objects to a patch and can't resolve the objection with the implementor and/or maintainer, then it gets discussed by the entire Tcl Core Team with the usual rules: if anyone on the Tcl Core Team has an objection that isn't resolved by the discussion, then a 2/3 vote is required to retain the patch. Thus if an implementor reaches a disagreement with a maintainer he/she can appeal to the entire Tcl Core Team. Or, if someone on the Tcl Core Team objects to a patch applied by a maintainer, they too can start a discussion in the whole team. The goal of the maintainer mechanism is to simplify and speed up improvements in the common case where everyone is in agreement, while still allowing the entire Tcl Core Team to offer input and resolve disagreements.

0.13 Changes that span areas

If a change involves several different areas of the Tcl sources, with different maintainers, then one of the maintainers acts as coordinator (presumably the one whose area has the most changes). It is their responsibility to consult with other maintainers whose areas are affected, so that all relevant maintainers are happy before the patch is applied to the sources.

0.14 Write access to the Tcl sources and the Web site

Everyone in the Tcl Core Team has write access to all the sources and the Web site, but they may only make changes consistent with approved projects. The Tcl Core Team can also give access to other people who are working on projects. For example, as part of a project proposal a Tcl Core Team member can propose that the work will be done by someone outside the team, and that that person should have write access for putting back changes. Giving out write access is part of a project decision, with the associated rules for approval. However, if someone outside the Tcl Core Team has write access, it must be under the auspices of a Tcl Core Team member; the Tcl Core Team member is personally responsible for making sure the project is completed satisfactorily and/or cleaning up any messes.

0.15 Deadlock resolution

If something should go wrong with the TCT organization and the Tcl Core Team deadlocks to a point where it can't make meaningful progress, then John Ousterhout will step in as benevolent dictator and make enough unilateral decisions to break the deadlock.

0.16 Copyright

TIP #1: TIP Index

TIP #1:	**TIP Index**
Author:	TIP Editor ⟨donal.fellows@cs.man.ac.uk⟩
Created:	Thursday, 14th September 2000
Type:	Informative
State:	Active
Vote:	No voting
Version:	`$Revision: 1.4 $`
Post-History:	

Abstract

This TIP contains the index of all TIPs published over the lifetime of the TCT. It will be continually and automatically updated.

1.1 Index

TIP ID	Type	State	Title
TIP #0	Process	Final	Tcl Core Team Basic Rules
TIP #1	Inform.	Active	TIP Index
TIP #2	Process	Draft	TIP Guidelines
TIP #3	Process	Accep.	TIP Format
TIP #4	Inform.	Draft	Tcl Release and Distribution Philosophy
TIP #5	Project	Final	Make TkClassProcs and TkSetClassProcs Public and Extensible
TIP #6	Project	Rejec.	Include [Incr Tcl] in the Core Tcl distribution
TIP #7	Project	Final	Increased resolution for TclpGetTime on Windows
TIP #8	Project	Final	Add Winico support to the wm command on windows
TIP #9	Project	Draft	Tk Standard Library
TIP #10	Project	Final	Tcl I/O Enhancement: Thread-Aware Channels
TIP #11	Project	Accep.	Tk Menubutton Enhancement: -compound option for menubutton
TIP #12	Inform.	Draft	The “Batteries Included” Distribution
TIP #13	Process	Accep.	Web Service for Drafting and Archiving TIPs
TIP #14	Project	Draft	Access (via tkInt) to Tk Photo Image Transparency
TIP #15	Project	Final	Functions to List and Detail Math Functions
TIP #16	Process	Accep.	Tcl Functional Areas for Maintainer Assignments
TIP #17	Project	Accep.	Redo Tcl’s filesystem
TIP #18	Project	Accep.	Add Labels to Frames
TIP #19	Project	Accep.	Add a Text Changed Flag to Tk’s Text Widget
TIP #20	Project	Draft	Add C Locale-Exact CType Functions
TIP #21	Project	Final	Asymmetric Padding in the Pack and Grid Geometry Managers
TIP #22	Project	Accep.	Multiple Index Arguments to lindex
TIP #23	Process	Accep.	Tk Toolkit Functional Areas for Maintainer Assignments
TIP #24	Inform.	Draft	Tcl Maintainer Assignments
TIP #25	Project	Draft	Native tk_messageBox on Macintosh
TIP #26	Project	Draft	Enhancements for the Tk Text Widget
TIP #27	Project	Accep.	CONST Qualification on Pointers in Tcl API’s
TIP #28	Inform.	Draft	How to be a good maintainer for Tcl/Tk
TIP #29	Project	Rejec.	Allow array syntax for Tcl lists
TIP #30	Inform.	Draft	Tk Toolkit Maintainer Assignments
TIP #31	Inform.	Draft	CVS tags in the Tcl and Tk repositories
TIP #32	Project	Draft	Add Tcl_Obj support to traces
TIP #33	Project	Accep.	Add ’lset’ Command to Assign to List Elements.
TIP #34	Project	Draft	TEA 2.0
TIP #35	Project	Draft	Enhanced Support for Serial Communications
TIP #36	Project	Draft	Library Access to ’Subst’ Functionality
TIP #37	Project	Draft	Uniform Rows and Columns in Grid

White backgrounds indicate that the TIP is still a draft, yellow backgrounds highlight TIPs being voted on, and where a TIP has been rejected, withdrawn or obsoleted its index entry has a dark grey background.

1.2 Explanations and How To Submit New TIPs

See [TIP #2] for a description of the editorial process a TIP has to go through and [TIP #3] for a description of their structure and the commands used to write them. You submit a TIP to this archive by emailing it (preferably in source form) to the TIP editor ⟨donal.fellows@cs.man.ac.uk⟩ who will check it for following of the guidelines,

style and general relevance to Tcl/Tk before checking it into the CVS archive and notifying the author, the rest of the Tcl Core Team, and the relevant newsgroups.

1.3 Copyright

TIP #2: TIP Guidelines

TIP #2:	**TIP Guidelines**
Author:	Andreas Kupries ⟨a.kupries@westend.com⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
	Don Porter ⟨dgp@users.sourceforge.net⟩
	Mo DeJong ⟨no@spam.com⟩
	Larry W. Virden ⟨lvirden@yahoo.com⟩
Created:	Tuesday, 12th September 2000
Type:	Process
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.17 $`
Post-History:	

Abstract

This document describes and defines the editorial process a TCT document (TIP) has to go through before accepted as official.

2.1 What is a TIP?

TIP stands for Tcl Improvement Proposal. A TIP is a design document providing information to the Tcl community, or describing a new feature for Tcl. The TIP should provide a concise technical specification of the feature and a rationale for the feature.

We intend TIPs to be the primary mechanisms for proposing new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Tcl. The TIP author is responsible for building consensus within the community and documenting dissenting opinions.

Because the TIPs are maintained as text files under revision control, their history is the historical record of the feature proposal. This historical record is available by the normal (CVS?) commands for retrieving older revisions. For those without direct access to the CVS tree, you can browse the current and past TIP revisions via http://www.cs.man.ac.uk/fellowsd-bin/TIP/ .

Further details on the arguments behind the evolution of the TIP concept and formatting can be found in the archive of the *tclcore* mailing list at http://www.geocrawler.com/redir-sf.php3?list=tcl-core .

2.2 Kinds of TIPs

There are three kinds of TIPs. A project TIP describes a new (or significantly updated) feature or implementation for Tcl. An informative TIP describes a Tcl design issue, or provides general guidelines or information to the Tcl community, but does not propose a new feature. A process TIP is like an informative TIP but the provided guidelines are mandatory in a certain context (as specified in the TIP itself).

Voting by the TCT as per the charter (see [TIP #0]) is required to make a project or process TIP official.

2.3 TIP Workflow

The TIP editor, Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ *pro tem*, assigns numbers for each TIP and changes its status.

Everyone in the community can submit a TIP to the TIP editor. It should contain at least a proposed title and a rough, but fleshed out, draft of the TIP.

If the TIP editor approves, he will assign the TIP a number, label it as either project, process or informational, give it status *Draft*, and create and check-in the initial draft of the TIP. The TIP editor will not unreasonably deny a TIP. Reasons for denying TIP status include gross malformatting, inappropriate copyright, duplication of effort, being technically unsound, or not in keeping with the Tcl philosophy; the TCT and after that John Ousterhout ⟨ouster@pacbell.net⟩ is the final arbitrator of the latter, as defined in the charter ([TIP #0]).

Discussion concerning a TIP should initially be kept out of the tclcore and tct mailing lists. Instead, comments should be sent to, and collected by, the TIP author, who has the responsibility to incorporate these comments into the document.

Note: It has been proposed to create a new mailing list for each TIP to handle its discussion. Rejection and finalization of the TIP closes the mailing list, but not the archive. Together with the CVS history a complete record of the development of a TIP will be available.

The authors of the TIP are responsible for writing the TIP and marshaling community support for it. The structure of a TIP is described in detail in [TIP #3].

A project TIP consists of two parts, a design document and a reference implementation. The TIP should be reviewed and accepted before a reference implementation is begun, unless a reference implementation will aid people in studying the TIP. The implementation can be given in the form of code, patch, or URL to same — before it can be considered *Final*; small reference implementations may be placed inside the TIP itself, and large reference implementations should be held externally and linked to by reference (typically URL.)

Process and Informational TIPs do not need an implementation.

TIP authors are responsible for collecting community feedback on a TIP before submitting it for review (the

creation of a TIP is a part of that review process.) However, wherever possible, long open-ended discussions on public mailing lists should be avoided. A better strategy is to encourage public feedback directly to the TIP author, who collects and integrates the comments back into the TIP.

Once the authors have completed a TIP, they must inform the Tcl Core Team that it is ready for review. TIPs are reviewed by the Tcl Core Team and (for Project TIPs) the maintainers for the relevant parts of the core, who may accept or reject a TIP or send it back to the author(s) for revision (as detailed in [TIP #0].) The acceptance or rejection of a TIP will cause its state to be changed accordingly to *Accepted* or *Rejected*.

Once a TIP requiring a reference implementation has been accepted, the reference implementation must be completed. When the reference implementation is complete and accepted by the TCT (who can reject it if they feel the implementation would damage the rest of the core) the status will be changed to *Final*.

A TIP can also be assigned status *Deferred*. The TIP author or the editor can assign the TIP this status when no progress is being made on the TIP. Once a TIP is deferred, the TIP editor can re-assign it to *Draft* status.

A TIP can also be *Withdrawn* by the author. Perhaps after all is said and done, the author believes it was not a good idea. It is still important to have a record of this fact. It is expected that *Accepted* TIPs will only be withdrawn very rarely, and *Final* TIPs only under exceptional circumstances.

TIP workflow is as follows:

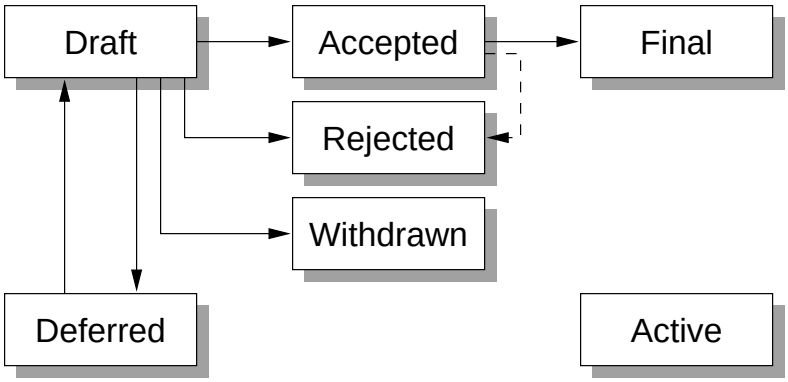

Figure 2.1: TIP Workflow

Some informative TIPs may also have a status of *Active* if they are never meant to be completed. For example: [TIP #1].

2.4 What belongs in a successful TIP?

Each TIP should have the following parts:

1. *Title* — a short, descriptive title.
2. *Author*(s) — names and contact info (email addresses) for each author.
3. *Abstract* — a short (typically <200 word) description of the technical issue being addressed.
4. *Copyright*/public domain — Each TIP must either be explicitly labelled in the public domain (the preferred 'license') or the Open Publication License (http://www.opencontent.org/openpub/). It is recommended that this be done by making the last section of the document be a copyright heading, with the body describing what copyright (if any) the document is released under.
5. *Specification* — Project TIPs should have a technical specification that should describe the syntax and semantics of any new language feature. The specification should be detailed enough to allow (competing) interoperable implementations for any of the current Tcl platforms.

6. *Rationale* — The rationale fleshes out the specification by describing what motivated the design and why particular design decisions were made. It should describe alternate designs that were considered and related work, *e.g.* how the feature is supported in other languages.

   The rationale should provide evidence of consensus within the community and discuss important objections or concerns raised during discussion.

7. *Reference Implementation* — The reference implementation must be completed before any TIP requiring such is given status *Final*, but it need not be completed before the TIP is accepted. It is better to finish the specification and rationale first and reach consensus on it before writing code.

   The final implementation must include test code and documentation appropriate for either the Tcl language reference or the standard library reference.

2.5 TIP Style

TIPs are written in plain ASCII text with an RFC822-like header and embedded sequences suitable for Wiki-like processing as indicated in [TIP #3].

There are Tcl scripts that convert the TIP into HTML for viewing on the web. Scripts for producing other formats are available too, for example LaTeX and plain ASCII.

2.6 Sample Project TIP

(With thanks to William H. Duquette ⟨William.H.Duquette@jpl.nasa.gov⟩ for suggesting this.) Note that the TIP Editor is responsible for allocating TIP numbers, so you can leave that unfilled.

```
TIP:           ???
Title:         The TIP Title as Plain Text
Version:       $Revision: 1.17 $
Author:        Author Name <author@somewhere.com>
State:         Draft
Type:          Project
Tcl-Version:   8.4
Vote:          Pending
Created:       31-Feb-1999
Post-History:

~ Abstract

This is an example of how to write a simple project TIP.  This is the
abstract which should consist of a single paragraph of under 200
words.  If you need more than this, you should stop and think about
writing a real abstract, not a whole section!  :^)

~ Some Sections

Yada yada yada.  Look at the sources to the various TIPs for tricks
on how to do various things.  ''Note that for complete legal safety,
you must specify what copyright is used.''  We prefer public domain,
as it allows others (notably the TIP editor(s)) to maintain the TIP
as necessary without having to seek permission.

I also prefer to make sure TIP and section titles are capitalized
according to the usual rules of English.

~ Copyright

This document has been placed in the public domain.
```

A more complex example is [TIP #7] by Kevin Kenny ⟨kennykb@acm.org⟩ (the source is at http://www.cs.man.ac.uk/fellowsd-bin/TIP/7.tip) and which includes demonstrations of how to use advanced features like figures and verbatim text. His is a very high quality TIP though, and it has been though several revisions; don't feel too put off if your first attempt isn't quite as good...

2.7 Patches

For preference, patches to Tcl should be stored separately on another website or submitted as a separate file. This is because (quite rightly) the news:comp.lang.tcl.announce moderator does not allow patches to be posted on that newsgroup. If you want a patch to be incorporated into the archive, please contact the TIP Editor.

2.8 Comments

From Don Porter ⟨dgp@users.sourceforge.net ⟩ :

1. It is confusing that "project" TIPs defined here do not correspond to "projects" defined in [TIP #0].

   (b) The TIP Workflow section should mention the web-based drafting service [TIP #13] as another way for members of the community to add their comments to a draft TIP.

   (c) The TIP Workflow section calls for TCT acceptance of a project TIP implementation to move it from state Accepted to state Final. This conflicts with [TIP #0] which delegates that acceptance task to maintainers.

   (d) It is not clear how the TIP workflow state diagram applies to non-project TIPs. Non-project TIPs do not have an implementation. Do they move straight from state Draft to state Final when they receive TCT approval? Or do they just stay in state Accepted forever with no need to move to state Final?

   (e) It should be noted in the TIP Workflow section that according to [TIP #0] a project TIP cannot be approved (and therefore should not be sponsored) until the TIP names a committed implementor.

   (f) The Patches section indicates that patches in the TIP are incompatible with posting to comp.lang.tcl.announce, so earlier sections of this TIP should not indicate that including patches in the TIP is an acceptable practice.

   (g) Should process TIPs be reserved for those proposals that revise [TIP #0] and require 2/3 support of the entire TCT?

From Mo DeJong:

It seems like we have a documentation problem with respect to how a TIP becomes "un-rejected".

The text says:

> ... may accept or reject a TIP or send it back to the author(s) for revision"

But the state transition diagram shows no way to go from *Rejected* to *Draft*. What becomes of a TIP after it is put up for a vote? If it is rejected, should the author create a new TIP number or can they rewrite the original TIP?

Larry W. Virden writes: I would really find it useful if upon submission of a TIP, a web page, perhaps on the Tcl'ers Wiki or elsewhere, would be referenced in the TIP itself. This web page would contain a summary of the discussion to date, perhaps containing urls to relevant postings within the tct archives, etc. This page could be used by those looking at the archive to quickly determine the state of the discussion, as well as the reasons for the final disposition of the TIP.

2.9 Copyright

TIP #3: TIP Format

TIP #3:	**TIP Format**
Author:	Andreas Kupries ⟨a.kupries@westend.com⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Thursday, 14th September 2000
Type:	Process
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.3 $`
Post-History:	

Abstract

This TIP is a companion document to the TIP Guidelines [TIP #2] and describes the structure and formatting to use when writing a TIP.

3.1 Rationale

The major goals of this document are to define a format that is

- easy to write,
- easy to read,
- easy to search, and
- acceptable to the community at large.

The latter is important because non-acceptance essentially means that the TIP process will be stillborn. This not only means basically plain text without much markup but also that we should reuse formats with which people are already acquainted.

As the concept of TIPs borrows heavily from Python's PEPs at http://python.sourceforge.net/peps/ their definition on how to structure and format a PEP was reviewed for its suitability of use by the TCT and the community at large.

The major points of the format are:

- Plain ASCII text without special markup for references or highlighting of important parts.
- Mail-like header section containing the meta-information.
- Uses indentation to distinguish section headers from section text.

A header section like is used in mail or news is something people are acquainted with and fulfils the other criteria too. In addition it is extendable. Using indentation to convey semantic and syntactic information on the other hand is something Pythonistas are used to but here in the Tcl world we are not to the same extent.

Looking at bit closer to home we find the Tcl'ers Wiki at http://www.purl.org/thecliff/tcl/wiki/

It does use a plain text format with some very light formatting conventions to allow things like links, images, enumerated and itemized lists.

Given the rather high acceptance of this site by the community using its format should be beneficiary to the acceptance of TIPs too.

It is therefore proposed to use a combination of a header in mail/news style together with a body employing a slightly extended/modified Wiki format (mostly backward compatible) as the format for TIPs. This proposed format is specified in detail below.

Note that the use of TAB characters within a TIP is discouraged (but permitted) as some mailers (notably Outlook Express) make a mess of them. Please be considerate and avoid their use...

3.2 Rejected Alternatives

But before we specify the format a (short) discussion of possible alternatives and why they where rejected.

There were three primary competitors to the format specified below, these are SGML/XML, HTML and a markup based upon plain text with embedded tcl-commands, for example like ... [section Abstract] ...

- The main disadvantage of SGML and XML based solutions is that they require a much more heavyweight infrastructure for editing and processing documents using them, like specialized editors and extensions for parsing. The format below on the other hand can be processed using pure tcl without extensions. with respect to the specialized editors it should be said that an editor operating on plain ASCII is possible too, but then the text will be difficult to read for humans because of the many occurrences of $<$ and $>$, conflicting with the requirement to have an 'easy to read' format.

  While there are commercial products which can gloss over this, making the editing of XML fairly easy, not everyone currently has access to one or the desire to spend what might be quite a lot of money to acquire one. It is far better to let everyone continue to use their current favourite plain-text editor.

- The main problem of HTML is that it is focused on visual and not logical markup. This will make it, although not impossible, but very difficult to parse documents for automatic handling. It is also a poor format for producing printed versions of the documentation from. Experience has also shown that different people have widely different ideas about how the content of TIP documents should be rendered into HTML, an indication that using the language would prove problematic! We can still use HTML as a generated format, but we should not write the documents themselves in it.
- The approach of embedding tcl commands into the text of a TIP is (at least) as powerful as XML when it comes to automatic processing of documents but much more lightweight. Because of this it is seen as the best of the three rejected alternatives. It was rejected in the end because it was still seen as too heavyweight/demanding for the casual user with respect to learning, easy writing and reading.

3.3 Header Format

The general format of the header for a TIP is specified in RFC 822 (http://www.rfc-editor.org/rfc/rfc822.txt). This leaves us to define and explain the keywords, their meaning and their values. The following keywords are *required*, and unless otherwise stated, should occur exactly once:

TIP The number of the TIP as assigned by the TIP editor. Unchangeable later on.

Title Defines the title of the document using plain text. May change during the discussion and review phases.

Version Specifies the version of the document. Usually something like $Revision: 1.3 $. (Initially $Revision: 1.3 $ should be used, which is then changed by the version control to contain the actual revision number.

Author Contact information (email address) for each author. The email address has to contain the real name of the author. If there are multiple authors of the document, this header may occur multiple times (once per author.) The format should be approximately like this: *Firstname Lastname <emailaddress>*

State Defines the state the TIP is currently in. Allowed values are *Draft*, *Active*, *Accepted*, *Deferred*, *Final*, *Rejected* and *Withdrawn*. This list will be influenced by the finalization of the workflow in [TIP #2].

Type The type of the TIP. Allowed values are *Process*, *Project* and *Informative*. See [TIP #2] for more explanations about the various types.

Vote The current state of voting for the TIP. Allowed values are *Pending*, *In progress*, *Done* and *No voting*. The latter is used to indicate a TIP which doesn't require a vote, for example [TIP #1].

Created The date the TIP was created, in the format dd-mmm-yyyy. *mmm* is the (English) short name of the month. The other information is numerical. Example: 14-Sep-2000

All numeric dates, though more easily internationalised, are not used because the ordering of particularly the month and day is ambiguous and subject to some confusion between different locales. Unix-style timestamps are unreadable to the majority of people (as well as being over-precise,) and I (⟨fellowsd@cs.man.ac.uk⟩) don't know ISO 8601 well enough to be able to comment on it.

Post-History A list of the dates the document was posted to the mailing list for discussion.

Tcl-Version This indicates the version of Tcl that a Project TIP depends upon (where it is required.) Process and Informative TIPs *must not* have this keyword.

The following headers are *optional* and should (unless otherwise stated) occur at most once:

Discussions-To While a TIP is in private discussions (usually during the initial Draft phase), this header will indicate the mailing list or URL where the TIP is being discussed.

Obsoletes Indicates a TIP number that this TIP renders obsolete. (Thanks to Joel Saunier ⟨Joel.Saunier@agriculture.gouv.fr⟩ for suggesting this!)

Obsoleted-By Indicates a TIP number that renders this TIP obsolete. (Thanks to Joel Saunier ⟨Joel.Saunier@agriculture.gouv.fr⟩ for suggesting this!)

Keywords A comma-separated list of keywords relating to this TIP, to facilitate automated indexing and improve search engine results.

The following headers are *proposed* (by Donald G. Porter ⟨dgp@cam.nist.gov⟩) but not currently supported:

Sponsor A TCT member that is sponsoring this TIP. May occur multiple times, once per sponsor.

Supporter A person (not necessarily a TCT member) who is supporting this TIP. May occur multiple times, once per supporter.

Objector A person (not necessarily a TCT member) who is opposed to this TIP. May occur multiple times, once per supporter.

3.4 Body Format

The body of a TIP is split by visually blank lines (i.e. lines containing nothing other than conventional whitespace) into units that will be called paragraphs. Each paragraph is in one of the following forms.

If the paragraph consists of exactly four minus symbols "----" then it is a separator paragraph and should be rendered as a horizonal rule.

If the paragraph consists of a vertical bar "|" followed by text, then it is a verbatim paragraph. The bar will be stripped from the front of each line and the rest of the text will be formatted literally. Tab characters will be expanded to 8-character boundaries. (*Note that this is completely incompatible with the Tcl'ers Wiki.*)

If the paragraph consists of a tilde "~" followed by text, then it is a section heading. The text following is the name of the section. In the name of good style, the section heading should have its significant words capitalised.

If the paragraph consists of the sequence "#index:" followed by some optional text, then it is a request to insert an index. The text following (after trimming spaces) indicates the kind of index desired. The default is a "medium" index, and fully compliant implementations should support "short" (expected to contain less detail) and "long" (expected to contain all header details plus the abstract) as well. Support for other kinds of indices is optional.

If the paragraph consists of the sequence "#image:" followed by some text, then it is a request to insert an image. The first word of the following text is a reference to the image, and the other words are an optional caption for the image (in plain text.) Image references that consist of just letters, numbers, hyphens and underscores are handled specially by the current implementation, which can map them to the correct media type for its current output format (assuming it has a suitable image in its repository.)

All other paragraphs that start with a non-whitespace character are ordinary paragraphs.

If a paragraph starts with a whitespace character sequence (use three spaces and keep the whole paragraph on a single line if you want compatability with the Tcl'ers Wiki,) a star "*" and another whitespace character, it is an item in a bulleted list.

If a paragraph starts with a whitespace character sequence, a number, a full stop "." and another whitespace character, it is an item in an enumerated list. If the number is 1 then the number of the item is guessed from the current list context, and any other value sets the number explicitly. If you want compatability with the Tcl'ers Wiki, make the initial whitespace sequence be three spaces, the number be 1, and keep the whole paragraph on a single line.

If a paragraph starts with a whitespace character sequence, some text (that includes no tabs or newlines but can include spaces), a colon and another whitespace character, then it is an item in a descriptive (a.k.a. definition) list.

If a paragraph does not start with a whitespace character sequence, a greater than symbol ">", and then another whitespace character, it is also an ordinary paragraph. (*Note that this is completely incompatible with the Tcl'ers Wiki.*)

Where a paragraph does begin with the sequence described in the preceding paragraph, it is a nested list item (if the paragraph contained is a list item) or a subsequent paragraph (if the paragraph contained is an ordinary paragraph.) If there's no suitable "enclosing" list context (i.e. if the preceding paragraph was not part of a list) the paragraph will be a quotation instead. (The rules for these continuation paras seem complex at first glance, but seem to work out fairly well in practise, especially since they are only rarely used.)

Within the body text of a (non-verbatim) paragraph, emphasis is indicated by enclosing the text within inside double apostrophes "''". (Wiki-style triple-apostrophes for bold are not supported, due to it being more trouble than it is worth to implement and the fact that we support section headings by a different mechanism.) Special URLs of the form tip:tipnumber are expanded into full URLs to the given TIP through the current formatting engine (where applicable.) References of the form [tipnumber] are also expanded as links to the given TIP, but are not displayed as URLs (the expansion is format dependent, of course.) Doubled up square brackets are converted into matching single square brackets. Email addresses (of the form ⟨email@address⟩) might also be treated specially.

The first paragraph of the body of any TIP must be an abstract section title ("~Abstract" or "~ Abstract"), and the second must be an ordinary paragraph (and should normally be just plain text, to make processing by tools easier.)

You can compare these rules with those for the Tcl'ers Wiki which are described at http://www.purl.org/thecliff/tcl/wiki/14.html, with the following modifications:

1. The text for an item in an itemized, enumerated or tagged list can be split over multiple physical lines. The text of the item will reach until the next empty line.
2. All paragraphs *must* be split with whitespace. This is a corollary of the above item.
3. A paragraph starting with the character ~ is interpreted as a section heading. Consequently it should be very short so that it renders onto a single line under most circumstances.
4. A full verbatim mode is added. Any line starting with the bar character is reproduced essentially verbatim (the bar character is removed). This allows embedding of code or other texts containing formatting usually recognized as special by the formatter without triggering this special processing. This applies especially to brackets and the hyperlinking they provide and their role in tcl code. This is used in preference to the whitespace rule of the Tcl'ers Wiki which is potentially far more sensitive. Our rule makes it extremely obvious what lines are verbatim, and what those lines will be rendered as.
5. Only one style of emphasis within paragraphs is supported. Having multiple emphasis styles (italic and bold) not only fails to carry across well in all media, but also makes for confusion on the part of authors and is more difficult to write renderers for too.
6. Images are only supported in a limited way, since under HTML the support for images varies a lot more than most people would like to think, and the concept of an inline image can vary quite a lot between different rendered formats too.

3.5 Reference Implementation

A reference renderer was created by Donal Fellows ⟨fellowsd@cs.man.ac.uk⟩ and is installed (as a behind-the-scenes rendering engine) on a set of TIP documents at http://www.cs.man.ac.uk/fellowsd-bin/TIP with the source code to the rendering engine being available at http://www.cs.man.ac.uk/~fellowsd/tcl/render/tip0.2/

Note that this code does support nested lists and multi-paragraph items, but this is experimental right now. Examples are presented behind the code itself.

3.6 Examples

This document itself is an example of the new format.

Examples for nested lists, multi-paragraph items in list's, and quotations.

Here is the source (itself a demonstration of verbatim text)

```
 * This is a paragraph

 > * This is an inner paragraph
```

```
     that goes onto two lines.

 > > * This one’s even further in!

 > > * So’s this one.

 > * Out again

 > > And a second paragraph here...

 > * Yet another item.

 * Outermost level once more.

 1. Enumerate?

 > 1. Deeper?

 2. Out again?

 list item: body text that is relatively long so that we can tell
   that it laps round properly as a paragraph even though this takes a
   ridiculous amount of text on my browser...

| VERB IN LIST?

 > nested: body

Top-level paragraph once more.

 > A quotation from someone famous might be rendered something like
   this.  As you can see, it is inset somewhat from the surrounding
   text. - ’’Donal K. Fellows <fellowsd@cs.man.ac.uk>’’

And back to the top-level yet again.

----
```

and the rendered result

- This is a paragraph
  - This is an inner paragraph that goes onto two lines.
    - This one’s even further in!
    - So’s this one.
  - Out again

    And a second paragraph here...
  - Yet another item.
- Outermost level once more.

1. Enumerate?
   (a) Deeper?
2. Out again?

list item body text that is relatively long so that we can tell that it laps round properly as a paragraph even though this takes a ridiculous amount of text on my browser...

```
 VERB IN LIST?
```

nested body

Top-level paragraph once more.

> A quotation from someone famous might be rendered something like this. As you can see, it is inset somewhat from the surrounding text. — *Donal K. Fellows* ⟨*fellowsd@cs.man.ac.uk*⟩

And back to the top-level yet again.

Examples of index generation and image paragraphs.

Here is the code

```
#index:

#index:short

#index: long

#image:3example This is a test caption

This is an example long TIP reference tip:3 that should be expanded in
a renderer-specific way...

This is an example non-reference - ''index[[3]]'' - that should not
be rendered as a link (to this document or anywhere else) at all.
Note that the dashes in the previous sentence (with whitespace on
each side) are candidates for rendering as long dashes (em-dashes) on
output-media which support this.

 Supported URLs: should be http, https, mailto, news, newsrc, ftp and
   gopher.  Test here...

 > HTTP URL - http://purl.org/thecliff/tcl/wiki/

 > HTTPS URL - https://sourceforge.net/

 > FTP URL - ftp://src.doc.ic.ac.uk/packages/tcl/tcl/

 > NEWS URL - news:comp.lang.tcl

 > MAILTO URL - mailto:fellowsd@cs.man.ac.uk?subject=TIP3

 > Others (might not be valid links!) - gopher://info.mcc.ac.uk,
   newsrc:2845823825
```

and here is the rendered result.

TIP ID	Type	State	Title
TIP #0	Process	Final	Tcl Core Team Basic Rules
TIP #1	Inform.	Active	TIP Index
TIP #2	Process	Draft	TIP Guidelines
TIP #3	Process	Accep.	TIP Format
TIP #4	Inform.	Draft	Tcl Release and Distribution Philosophy
TIP #5	Project	Final	Make TkClassProcs and TkSetClassProcs Public and Extensible

TIP #6	Project	Rejec.	Include [Incr Tcl] in the Core Tcl distribution
TIP #7	Project	Final	Increased resolution for TclpGetTime on Windows
TIP #8	Project	Final	Add Winico support to the wm command on windows
TIP #9	Project	Draft	Tk Standard Library
TIP #10	Project	Final	Tcl I/O Enhancement: Thread-Aware Channels
TIP #11	Project	Accep.	Tk Menubutton Enhancement: -compound option for menubutton
TIP #12	Inform.	Draft	The “Batteries Included” Distribution
TIP #13	Process	Accep.	Web Service for Drafting and Archiving TIPs
TIP #14	Project	Draft	Access (via tkInt) to Tk Photo Image Transparency
TIP #15	Project	Final	Functions to List and Detail Math Functions
TIP #16	Process	Accep.	Tcl Functional Areas for Maintainer Assignments
TIP #17	Project	Accep.	Redo Tcl’s filesystem
TIP #18	Project	Accep.	Add Labels to Frames
TIP #19	Project	Accep.	Add a Text Changed Flag to Tk’s Text Widget
TIP #20	Project	Draft	Add C Locale-Exact CType Functions
TIP #21	Project	Final	Asymmetric Padding in the Pack and Grid Geometry Managers
TIP #22	Project	Accep.	Multiple Index Arguments to lindex
TIP #23	Process	Accep.	Tk Toolkit Functional Areas for Maintainer Assignments
TIP #24	Inform.	Draft	Tcl Maintainer Assignments
TIP #25	Project	Draft	Native tk_messageBox on Macintosh
TIP #26	Project	Draft	Enhancements for the Tk Text Widget
TIP #27	Project	Accep.	CONST Qualification on Pointers in Tcl API’s
TIP #28	Inform.	Draft	How to be a good maintainer for Tcl/Tk
TIP #29	Project	Rejec.	Allow array syntax for Tcl lists
TIP #30	Inform.	Draft	Tk Toolkit Maintainer Assignments
TIP #31	Inform.	Draft	CVS tags in the Tcl and Tk repositories
TIP #32	Project	Draft	Add Tcl_Obj support to traces
TIP #33	Project	Accep.	Add ’lset’ Command to Assign to List Elements.
TIP #34	Project	Draft	TEA 2.0
TIP #35	Project	Draft	Enhanced Support for Serial Communications
TIP #36	Project	Draft	Library Access to ’Subst’ Functionality
TIP #37	Project	Draft	Uniform Rows and Columns in Grid

TIP #0: Tcl Core Team Basic Rules

TIP #1: TIP Index

TIP #2: (*Draft*) TIP Guidelines

TIP #3: TIP Format

TIP #4: (*Draft*) Tcl Release and Distribution Philosophy

TIP #5: Make TkClassProcs and TkSetClassProcs Public and Extensible

TIP #6: (*Rejected*) Include [Incr Tcl] in the Core Tcl distribution

TIP #7: Increased resolution for TclpGetTime on Windows

TIP #8: Add Winico support to the wm command on windows

TIP #9: (*Draft*) Tk Standard Library

TIP #10: Tcl I/O Enhancement: Thread-Aware Channels

TIP #11: Tk Menubutton Enhancement: -compound option for menubutton

TIP #12: (*Draft*) The “Batteries Included” Distribution

TIP #13: Web Service for Drafting and Archiving TIPs

TIP #14: (*Draft*) Access (via tkInt) to Tk Photo Image Transparency

TIP #15: Functions to List and Detail Math Functions

TIP #16: Tcl Functional Areas for Maintainer Assignments

TIP #17: Redo Tcl's filesystem

TIP #18: Add Labels to Frames

TIP #19: Add a Text Changed Flag to Tk's Text Widget

TIP #20: (*Draft*) Add C Locale-Exact CType Functions

TIP #21: Asymmetric Padding in the Pack and Grid Geometry Managers

TIP #22: Multiple Index Arguments to lindex

TIP #23: Tk Toolkit Functional Areas for Maintainer Assignments

TIP #24: (*Draft*) Tcl Maintainer Assignments

TIP #25: (*Draft*) Native tk_messageBox on Macintosh

TIP #26: (*Draft*) Enhancements for the Tk Text Widget

TIP #27: CONST Qualification on Pointers in Tcl API's

TIP #28: (*Draft*) How to be a good maintainer for Tcl/Tk

TIP #29: (*Rejected*) Allow array syntax for Tcl lists

TIP #30: (*Draft*) Tk Toolkit Maintainer Assignments

TIP #31: (*Draft*) CVS tags in the Tcl and Tk repositories

TIP #32: (*Draft*) Add Tcl_Obj support to traces

TIP #33: Add 'lset' Command to Assign to List Elements.

TIP #34: (*Draft*) TEA 2.0

TIP #35: (*Draft*) Enhanced Support for Serial Communications

TIP #36: (*Draft*) Library Access to 'Subst' Functionality

TIP #37: (*Draft*) Uniform Rows and Columns in Grid

TIP #0:	**Tcl Core Team Basic Rules**
Version:	`$Revision: 2.3 $`
Author:	John Ousterhout ⟨ouster@interwoven.com⟩
State:	Final
Type:	Process
Vote:	Done
Created:	11 December 2000
Posting History:	
Abstract:	This TIP describes the mission, structure, and operating procedures of the Tcl Core Team (TCT). When in doubt about how the TCT works, consult this document as the final authority.

TIP #1:	**TIP Index**
Version:	`$Revision: 1.4 $`
Author:	TIP Editor ⟨donal.fellows@cs.man.ac.uk⟩
State:	Active
Type:	Informative
Vote:	No voting
Created:	14 September 2000
Posting History:	

Abstract:	This TIP contains the index of all TIPs published over the lifetime of the TCT. It will be continually and automatically updated.

TIP #2:	**TIP Guidelines**
Version:	`$Revision: 1.17 $`
Authors:	Andreas Kupries ⟨a.kupries@westend.com⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ Don Porter ⟨dgp@users.sourceforge.net⟩ Mo DeJong ⟨no@spam.com⟩ Larry W. Virden ⟨lvirden@yahoo.com⟩
State:	Draft
Type:	Process
Vote:	Pending
Created:	12 September 2000
Posting History:	
Abstract:	This document describes and defines the editorial process a TCT document (TIP) has to go through before accepted as official.

TIP #3:	**TIP Format**
Version:	`$Revision: 1.3 $`
Authors:	Andreas Kupries ⟨a.kupries@westend.com⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
State:	Accepted
Type:	Process
Vote:	Done
Created:	14 September 2000
Posting History:	
Abstract:	This TIP is a companion document to the TIP Guidelines [TIP #2] and describes the structure and formatting to use when writing a TIP.

TIP #4:	**Tcl Release and Distribution Philosophy**
Version:	`$Revision: 1.10 $`
Authors:	Brent Welch ⟨welch@acm.org⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ Larry W. Virden ⟨lvirden@cas.org⟩ Larry W. Virden ⟨lvirden@yahoo.com⟩
State:	Draft
Type:	Informative
Vote:	Pending
Created:	26 October 2000
Posting History:	
Discussions To:	news:comp.lang.tcl
Abstract:	This document outlines how Tcl should be distributed, with particular reference to issues related to building a distribution with the *batteries included* so that most people can have access to the useful extensions without having to chasing halfway across the 'net for them.

TIP #5:	**Make TkClassProcs and TkSetClassProcs Public and Extensible**
Version:	`$Revision: 1.2 $`
Author:	Eric Melski ⟨ericm@ajubasolutions.com⟩
State:	Final
Type:	Project
Tcl Version:	8.4
Vote:	Done
Created:	17 October 2000
Posting History:	
Abstract:	At certain critical moments in the lifetime of a Tk widget, Tk will invoke various callbacks on that widget. These callbacks enable the widget to do lots of interesting things, such as react to configuration changes for named fonts, or create and manage truly native widgets (such as the scrollbar widget on Windows platforms). The API for setting up these callbacks for a particular window are, as of Tk 8.3.2, private. This prohibits extension widget authors from fully utilizing this powerful system; those developers can either copy the private declarations into their own source code (leading to future maintenance hassles), or forego the system entirely, hampering their ability to make truly native and well-integrated widgets. This proposal offers an extensible means for making that API public.

TIP #6:	**Include [Incr Tcl] in the Core Tcl distribution**
Version:	`$Revision: 1.5 $`
Author:	Mark Harrison ⟨markh@usai.asiainfo.com⟩
State:	Rejected
Type:	Project
Tcl Version:	8.4.0
Vote:	Done
Created:	16 October 2000
Posting History:	
Abstract:	Include [Incr Tcl] in the Core Tcl distribution.

TIP #7:	**Increased resolution for TclpGetTime on Windows**
Version:	`$Revision: 1.3 $`
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
State:	Final
Type:	Project
Tcl Version:	8.4
Vote:	Done
Created:	26 October 2000
Posting History:	
Discussions To:	news:comp.lang.tcl
Abstract:	Tcl users on the Windows platform have long been at a disadvantage in attempting to do code timing studies, owing to the poor resolution of the Windows system clock. The *time* command, the *clock clicks* command, and all related functions are limited to a resolution of (typically) 10 milliseconds. This proposal offers a solution based on the Windows performance counter. It presents a means of disciplining this counter to the system clock so that *TclpGetTime* (the underlying call that the above commands use) can return times to microsecond precision with accuracy in the tens of microseconds.

TIP #8:	**Add Winico support to the wm command on windows**
Version:	`$Revision: 1.7 $`
Author:	Vince Darley ⟨vince.darley@eurobios.com⟩
State:	Final
Type:	Project
Tcl Version:	8.4.0
Vote:	Done
Created:	06 November 2000
Posting History:	
Abstract:	Add to *wm* the ability to do the windows-titlebar-icon manipulation that the Winico extension currently provides, without the bugs noted in that extension.

TIP #9:	**Tk Standard Library**
Version:	`$Revision: 1.3 $`
Authors:	Marty Backe ⟨mgbacke@usa.net⟩ hellins ⟨hellins@263.net⟩ Larry W. Virden ⟨lvirden@yahoo.com⟩
State:	Draft
Type:	Project
Tcl Version:	8.4
Vote:	Pending
Created:	07 November 2000
Posting History:	
Abstract:	A Tk standard library shall be bundled with the core Tcl/Tk distribution. The library will consist of general purpose widgets and composite widgets for use in constructing Tcl/Tk applications. The library of Tk components will be written in Tcl/Tk.

TIP #10:	**Tcl I/O Enhancement: Thread-Aware Channels**
Version:	`$Revision: 1.6 $`
Author:	Andreas Kupries ⟨a.kupries@westend.com⟩
State:	Final
Type:	Project
Tcl Version:	8.4
Vote:	Done
Created:	08 November 2000
Posting History:	
Abstract:	This TIP describes how to change the generic I/O layer in the Tcl core to make channels aware of the thread they are managed by.

TIP #11:	**Tk Menubutton Enhancement: -compound option for menubutton**
Version:	`$Revision: 1.4 $`
Author:	Todd Helfter ⟨tmh@purdue.edu⟩
State:	Accepted
Type:	Project
Tcl Version:	8.4
Vote:	Done

Created:	16 November 2000
Posting History:	
Abstract:	This TIP describes how to change the menubutton in the Tk core to add a -compound option to display both text and images. This behavior already exists in the button widget.

TIP #12:	**The "Batteries Included" Distribution**
Version:	`$Revision: 1.3 $`
Authors:	George A. Howlett ⟨gah@siliconmetrics.com⟩ Larry W. Virden ⟨lvirden@yahoo.com⟩
State:	Draft
Type:	Informative
Vote:	Pending
Created:	15 September 2000
Posting History:	
Discussions To:	news:comp.lang.tcl
Abstract:	This document describes a comprehensive Tcl/Tk distribution. Its primary purpose is to create a standard source tree that includes Tcl, Tk, and extensions so that they can be built and installed in an simple and easy manner.

TIP #13:	**Web Service for Drafting and Archiving TIPs**
Version:	`$Revision: 1.24 $`
Authors:	Don Porter ⟨dgp@users.sourceforge.net⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
State:	Accepted
Type:	Process
Vote:	Done
Created:	21 November 2000
Posting History:	
Abstract:	This document proposes the TCT provide a service on the World Wide Web for drafting and archiving TIPs and for providing TIPs in a variety of formats. A reference implementation is provided, and its server requirements are outlined.

TIP #14:	**Access (via tkInt) to Tk Photo Image Transparency**
Version:	`$Revision: 1.3 $`
Author:	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
State:	Draft
Type:	Project
Tcl Version:	8.4.0
Vote:	Pending
Created:	22 November 2000
Posting History:	
Keywords:	Tk, photo, transparency, internal, access
Abstract:	It is useful for some extensions to have access to the transparency information in photo images for various reasons, but this is not currently available, even via an internal structure defined in *generic/tkInt.h*. This TIP is aimed at making the information available in a way that can be kept backwardly compatible even if the internal structure definitions change.

TIP #15:	**Functions to List and Detail Math Functions**
Version:	`$Revision: 1.8 $`
Author:	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
State:	Final
Type:	Project
Tcl Version:	8.4.0
Vote:	Done
Created:	22 November 2000
Posting History:	
Keywords:	Tcl, expr, function, introspection
Abstract:	Provides a way for the list of all math functions defined in the current interpreter to be discovered, and for discovering what arguments might be passed to an existing math function. This may be useful in tests as well as more general use.

TIP #16:	**Tcl Functional Areas for Maintainer Assignments**
Version:	`$Revision: 1.9 $`
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩
State:	Accepted
Type:	Process
Vote:	Done
Created:	21 November 2000
Posting History:	
Abstract:	This document proposes a division of Tcl's source code into functional areas so that each area may be assigned to one or more maintainers.

TIP #17:	**Redo Tcl's filesystem**
Version:	`$Revision: 1.17 $`
Author:	Vince Darley ⟨vince@santafe.edu⟩
State:	Accepted
Type:	Project
Tcl Version:	8.4.0
Vote:	Done
Created:	17 November 2000
Posting History:	
Abstract:	Many of the most exciting recent developments in Tcl have involved putting virtual file systems in a file (e.g. Prowrap, Freewrap, Wrap, TclKit) but these have been largely *ad hoc* hacks of various internal APIs. This TIP seeks to replace this with a common underlying API that will, in addition, make porting of Tcl to new platforms a simpler task as well.

TIP #18:	**Add Labels to Frames**
Version:	`$Revision: 2.2 $`
Author:	Peter Spjuth ⟨peter.spjuth@space.se⟩
State:	Accepted
Type:	Project
Tcl Version:	8.4
Vote:	Done

Created:	12 December 2000
Posting History:	
Abstract:	This TIP proposes to add a labelled frame widget to Tk.

TIP #19:	**Add a Text Changed Flag to Tk's Text Widget**
Version:	`$Revision: 1.5 $`
Author:	Neil McKay ⟨mckay@eecs.umich.edu⟩
State:	Accepted
Type:	Project
Tcl Version:	8.4a2
Vote:	Done
Created:	03 January 2001
Posting History:	
Obsoleted By:	TIP #26
Abstract:	This TIP adds a *text changed* flag to the Tk text widget. The flag would initially be reset, but would be set whenever the contents of the text widget changes.

TIP #20:	**Add C Locale-Exact CType Functions**
Version:	`$Revision: 1.1 $`
Author:	Jeffrey Hobbs ⟨jeff.hobbs@acm.org⟩
State:	Draft
Type:	Project
Tcl Version:	8.4a2
Vote:	Pending
Created:	08 January 2001
Posting History:	
Abstract:	This TIP adds functions to Tcl that are a subset of the standard ctype functions (isspace, isalpha, ...) that are ensured to operate only in the C locale (char $<$ 0x80).

TIP #21:	**Asymmetric Padding in the Pack and Grid Geometry Managers**
Version:	`$Revision: 1.6 $`
Author:	D. Richard Hipp ⟨drh@hwaci.com⟩
State:	Final
Type:	Project
Tcl Version:	8.4
Vote:	Done
Created:	14 January 2001
Posting History:	
Abstract:	Proposes modifying the *pack* and *grid* geometry managers to support asymmetric padding.

TIP #22:	**Multiple Index Arguments to lindex**
Version:	`$Revision: 1.21 $`
Authors:	David Cuthbert ⟨dacut@kanga.org⟩ Kevin Kenny ⟨kennykb@acm.org⟩ Don Porter ⟨dgp@users.sourceforge.net⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩

State:	Accepted
Type:	Project
Tcl Version:	8.4a2
Vote:	Done
Created:	19 January 2001
Posting History:	
Discussions To:	news:comp.lang.tcl, mailto:kennykb@acm.org
Keywords:	lindex, multiple arguments, sublists
Abstract:	Obtaining access to elements of sublists in Tcl often requires nested calls to the *lindex* command. The indices are syntactically listed in most-nested to least-nested order, which is the reverse from other notations. In addition, the nesting of command substitution brackets further decreases readability. This proposal describes an extension to the *lindex* command that allows it to accept multiple index arguments, in least-nested to most-nested order, to automatically extract elements of sublists.

TIP #23:	**Tk Toolkit Functional Areas for Maintainer Assignments**
Version:	`$Revision: 1.19 $`
Authors:	Kevin Kenny ⟨kennykb@acm.org⟩ Jim Ingham ⟨jingham@apple.com⟩ Don Porter ⟨dgp@users.sourceforge.net⟩
State:	Accepted
Type:	Process
Vote:	Done
Created:	22 January 2001
Posting History:	
Abstract:	This document proposes a division of the Tk toolkit's source code into functional areas so that each area may be assigned to one or more maintainers.

TIP #24:	**Tcl Maintainer Assignments**
Version:	`$Revision: 1.21 $`
Authors:	Don Porter ⟨dgp@users.sourceforge.net⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ Kevin B KENNY ⟨kennykb@acm.org⟩ ⟨dgp@user.sourceforge.net⟩
State:	Draft
Type:	Informative
Vote:	Pending
Created:	29 January 2001
Posting History:	
Abstract:	This document keeps a record of who maintains each functional area of Tcl ([TIP #16]).

TIP #25:	**Native tk_messageBox on Macintosh**
Version:	`$Revision: 1.1 $`
Author:	Mats Bengtsson ⟨matben@privat.utfors.se⟩
State:	Draft

Type:	Project
Tcl Version:	8.4a2
Vote:	Pending
Created:	07 February 2001
Posting History:	
Abstract:	This is a replacement for the *tk_messageBox* on the Macintosh with a native implementation which is compliant with the Appearance Manager in Mac OS 8 and later.

TIP #26:	**Enhancements for the Tk Text Widget**
Version:	`$Revision: 1.5 $`
Authors:	Ludwig Callewaert ⟨ludwig_callewaert@frontierd.com⟩ Ludwig Callewaert ⟨ludwig.callewaert@belgacom.net⟩
State:	Draft
Type:	Project
Tcl Version:	8.4
Vote:	Pending
Created:	20 February 2001
Posting History:	
Discussions To:	news:comp.lang.tcl
Obsoletes:	TIP #19
Abstract:	This TIP proposes several enhancements for the Tk text widget. An unlimited undo/redo mechanism is proposed, with several user available customisation features. Related to this, a text modified indication is proposed. This means that the user can set, query or receive a virtual event when the content of the text widget is modified. And finally a virtual event is added that is generated whenever the selection changes in the text widget.

TIP #27:	**CONST Qualification on Pointers in Tcl API's**
Version:	`$Revision: 1.5 $`
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
State:	Accepted
Type:	Project
Tcl Version:	8.4
Vote:	Done
Created:	25 February 2001
Posting History:	
Discussions To:	news:comp.lang.tcl, mailto:kennykb@acm.org
Abstract:	Many of the C and C++ interfaces to the Tcl library lack a CONST qualifier on the parameters that accept pointers, even though they do not, in fact, modify the data that the pointers designate. This lack causes a persistent annoyance to C/C++ programmers. Not only is the code needed to work around this problem more verbose than required; it also can lead to compromises in type safety. This TIP proposes that the C interfaces for Tcl be revised so that functions that accept pointers to constant data have type signatures that reflect the fact. The new interfaces will remain backward-compatible with the old, except that a few must be changed to return pointers to CONST data. (Changes of this magnitude, in the past, have been routine in minor releases; the author of this TIP does not see a compelling reason to wait for Tcl 9.0 to clean up these API's.)

TIP #28:	**How to be a good maintainer for Tcl/Tk**
Version:	`$Revision: 1.8 $`
Authors:	Don Porter ⟨dgp@users.sourceforge.net⟩ ⟨dgp@user.sourceforge.net⟩
State:	Draft
Type:	Informative
Vote:	Pending
Created:	23 February 2001
Posting History:	
Abstract:	This document presents information and advice to maintainers in the form of a Frequently Asked Questions (FAQ) list.

TIP #29:	**Allow array syntax for Tcl lists**
Version:	`$Revision: 1.7 $`
Authors:	Kevin Kenny ⟨kennykb@acm.org⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
State:	Rejected
Type:	Project
Tcl Version:	9.0
Vote:	Done
Created:	07 March 2001
Posting History:	
Discussions To:	news:comp.lang.tcl, mailto:kennykb@acm.org
Abstract:	Most popular programming languages provide some sort of indexed array construct, where array subscripts are integers. Tcl's lists are, in fact, arrays, but the existing syntax obscures the fact. Moreover, the existing list commands make it difficult to manipulate lists as arrays without running into peculiar performance issues. This TIP proposes that the syntax of *variableName(value)* be extended to function as an array selector if *variableName* designates a list. This change is upward compatible with existing Tcl scripts, because the proposed syntax results in a runtime error in every extant Tcl release.

TIP #30:	**Tk Toolkit Maintainer Assignments**
Version:	`$Revision: 1.21 $`
Authors:	Don Porter ⟨dgp@users.sourceforge.net⟩ Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ Jan Nijtmans ⟨j.nijtmans@chello.nl⟩ Todd M. Helfter ⟨tmh@purdue.edu⟩ Chengye Mao ⟨chengye.geo@yahoo.com⟩ George B. Smith ⟨gbs@k9haven.com⟩ Miguel Ban ⟨bagnonm@safelayer.com⟩
State:	Draft
Type:	Informative
Vote:	Pending
Created:	09 March 2001
Posting History:	
Abstract:	This document keeps a record of who maintains each functional area of Tk ([TIP #23]).

TIP #31:	**CVS tags in the Tcl and Tk repositories**
Version:	`$Revision: 1.5 $`
Authors:	Don Porter ⟨dgp@users.sourceforge.net⟩ miguel sofer ⟨mig@utdt.edu⟩ Jeff Hobbs ⟨JeffH@ActiveState.com⟩ Kevin Kenny ⟨kennykb@acm.org⟩
State:	Draft
Type:	Informative
Vote:	Pending
Created:	12 March 2001
Posting History:	
Abstract:	This document keeps a record of the CVS tags used in the Tcl and Tk repositories and their meanings.

TIP #32:	**Add Tcl_Obj support to traces**
Version:	`$Revision: 1.3 $`
Authors:	David Cuthbert ⟨dacut@kanga.org⟩ Kevin Kenny ⟨kennykb@acm.org⟩
State:	Draft
Type:	Project
Tcl Version:	8.4a2
Vote:	Pending
Created:	23 March 2001
Posting History:	
Discussions To:	news:comp.lang.tcl
Keywords:	trace, Tcl_Obj
Abstract:	This document proposes to add Tcl_Obj support for trace procedures written in C.

TIP #33:	**Add 'lset' Command to Assign to List Elements.**
Version:	`$Revision: 1.11 $`
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
State:	Accepted
Type:	Project
Tcl Version:	8.4
Vote:	Done
Created:	15 May 2001
Posting History:	
Discussions To:	news:comp.lang.tcl, mailto:kennykb@acm.org
Abstract:	Most popular programming languages provide some sort of indexed array construct, where array subscripts are integers. Tcl's lists are implemented internally as indexed arrays, but it is difficult to use them as such because there is no convenient way to assign to individual elements. This TIP proposes a new command, *lset*, to rectify this limitation.

TIP #34:	**TEA 2.0**
Version:	`$Revision: 1.1 $`
Author:	Mo DeJong ⟨mdejong@cygnus.com⟩
State:	Draft
Type:	Project
Tcl Version:	8.4
Vote:	Pending
Created:	03 May 2001
Posting History:	
Abstract:	The original TEA specification, documentation, and implementation have fallen out of date. Numerous complaints about the difficulty of creating a TEA compliant package have appeared on news:comp.lang.tcl The existing build system works but it is a pain to maintain mostly because there are two build systems, one for unix and another for windows. This document describes how some of these concerns can be addressed.

TIP #35:	**Enhanced Support for Serial Communications**
Version:	`$Revision: 1.6 $`
Author:	Rolf Schroedter ⟨rolf.schroedter@dlr.de⟩
State:	Draft
Type:	Project
Tcl Version:	8.4
Vote:	Pending
Created:	06 June 2001
Posting History:	
Abstract:	Tcl's support for RS-232 is very rudimentary. Mainly it allows to setup the communication rate [fconfigure -mode] and to read and write data with the standard Tcl functions. Real serial communications are often more complex. Therefore it is proposed to add support for hardware and software flow control, polling RS-232 (modem) status lines, and watching the input and output queue. This is all to be implemented via additional [fconfigure] options.

TIP #36:	**Library Access to 'Subst' Functionality**
Version:	`$Revision: 1.1 $`
Author:	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
State:	Draft
Type:	Project
Tcl Version:	8.4
Vote:	Pending
Created:	13 June 2001
Posting History:	
Abstract:	Some applications make very heavy use of the *subst* command — it seems particularly popular in the active-content-generation field — and for them it is important to optimise this as much as possible. This TIP adds a direct interface to these capabilities to the Tcl library, allowing programmers to avoid the modest overheads of even *Tcl_EvalObjv* and the option parser for the *subst* command implementation.

TIP #37:	**Uniform Rows and Columns in Grid**
Version:	`$Revision: 1.2 $`
Author:	Peter Spjuth ⟨peter.spjuth@space.se⟩
State:	Draft
Type:	Project
Tcl Version:	8.4
Vote:	Pending
Created:	19 June 2001
Posting History:	
Abstract:	This TIP proposes to add a *-uniform* option to *grid rowconfigure* and *grid columnconfigure* so as to make it easier to create layouts where cells are constrained to have identical dimensions.

Figure 3.2: This is a test caption

This is an example long TIP reference http://www.cs.man.ac.uk/fellowsd-bin/TIP/3.tex that should be expanded in a renderer-specific way...

This is an example non-reference — *index[3]* — that should not be rendered as a link (to this document or anywhere else) at all. Note that the dashes in the previous sentence (with whitespace on each side) are candidates for rendering as long dashes (em-dashes) on output-media which support this.

Supported URLs should be http, https, mailto, news, newsrc, ftp and gopher. Test here...

HTTP URL — http://purl.org/thecliff/tcl/wiki/

HTTPS URL — https://sourceforge.net/

FTP URL — ftp://src.doc.ic.ac.uk/packages/tcl/tcl/

NEWS URL — news:comp.lang.tcl

MAILTO URL — mailto:fellowsd@cs.man.ac.uk?subject=TIP3

Others (might not be valid links!) — gopher://info.mcc.ac.uk, newsrc:2845823825

3.7 Copyright

This document has been placed in the public domain.

TIP #4: Tcl Release and Distribution Philosophy

TIP #4:	**Tcl Release and Distribution Philosophy**
Author:	Brent Welch ⟨welch@acm.org⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
	Larry W. Virden ⟨lvirden@cas.org⟩
	Larry W. Virden ⟨lvirden@yahoo.com⟩
Created:	Thursday, 26th October 2000
Type:	Informative
State:	Draft
Vote:	Pending
Version:	`$Revision:  1.10 $`
Post-History:	
Discussions-To:	news:comp.lang.tcl

Abstract

This document outlines how Tcl should be distributed, with particular reference to issues related to building a distribution with the *batteries included* so that most people can have access to the useful extensions without having to chasing halfway across the 'net for them.

4.1 Overview

Tcl has traditionally been a "core" that is extensible with binary extensions and Tcl scripts. There have been two styles of Tcl distributions: source and binary. The Tcl source distribution contains the Tcl "core" and a small number of support scripts. The binary distributions have included Tk, and in some cases (e.g., TclPro) other extensions like [incr Tcl], TclX, and Expect. Users with access to a compiler can get source distributions of the various extensions and compile them for their own installation. *(Thanks to Bob Technetin ⟨techentin.robert@mayo.edu⟩ for the inspiration for these pictures — DKF.)*

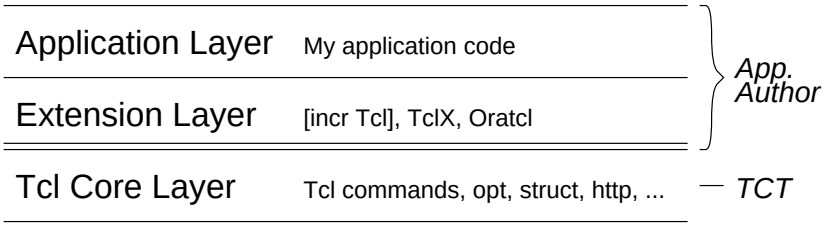

Figure 4.3: Traditional Tcl Distribution Architecture

This proposal formalizes the notion of a small Tcl source core and larger distribution bundle that includes one or many extensions. The distribution can be in source or binary form. The goal is to keep a small core that is suitable for embedding with the smallest footprint, while acknowleding that desktop users and application developers want a larger *standard distribution* that has a set of well known and widely used extensions.

The goal of this proposal is to establish a standard for future Tcl distributions. There will be two kinds of Tcl distributions: a small core suitable for specialized embedded applications, and a larger bundled distribution suitable for more general application development.

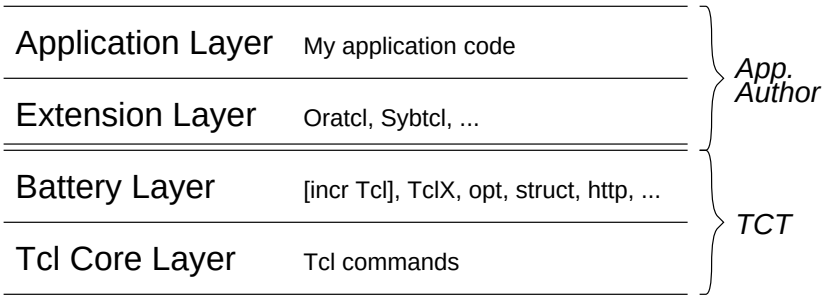

Figure 4.4: Batteries-Included Tcl Distribution Architecture

4.2 The Tcl Core Distribution

The Tcl "core" should remain as small as possible, and could become smaller in the future as certain features are moved into extensions. The "core" distribution must include:

1. The C sources required to create the Tcl binary library.
2. The C sources required to create a "Tcl Shell" application. This is commonly known as "tclsh".
3. The Tcl script libraries that implement the code library and packaging systems. This includes the "unknown" command and various commands related to auto loading of packages.
4. The Tcl test framework used for testing the Tcl binary library and the support scripts.

Additional items may appear in the "core" distribution, especially for historical reasons. But, some Tcl scripts and binary extensions that currently exist (as of Tcl 8.3) in the Tcl source distributions may migrate into the larger distribution described below.

4.3 The Bundled Distribution

The *bundled* Tcl distribution will contain Tcl, various binary extensions, and various Tcl script packages. This proposal establishes an initial set of binary extensions, but following the model and using the distribution infrastructure we create, various bundles should be easily created.

Each package included in the bundled distribution must have a test suite and documentation. At this stage the documentation will probably be in a variety of formats, but ultimately we should standardize on an XML-based representation and supply tools that generate other formats from that representation.

4.4 Mandatory Packages

The bundled distribution must include (but is not limited to):

1. The "core" distribution described above.
2. The Tk toolkit for GUI applications. This includes the well known "wish" shell application.
3. The registry and dde extensions for the Windows platform.
4. The [incr Tcl] extension.
5. The TclX extension. There are some historical features of TclX that should not necessarily be included, including the tclx shell and its alternate library format. However, the TclX help system should not only be included, but updated to include info on all commands included in the distribution.
6. The Expect extension for UNIX platforms.
7. The TkCon enhanced console application.

4.5 Optional Packages

In addition, it is likely that several of the following packages will be included in the bundled distribution, as well as others not listed.

1. The Standard Tcl Library of Tcl scripts. Currently this includes packages for:
   (a) base64 encoding/decoding
   (b) file utilities
   (c) command line processing
   (d) FTP client library
   (e) FTP server
   (f) HTML and JavaScript generation
   (g) Math and statistics utilities
   (h) MIME encoder and parser
   (i) CGI processing (ncgi)
   (j) NNTP client
   (k) POP3 client

(l) Profiler for Tcl scripts
(m) Event counters and interval timers
(n) Structures, including tree, stack, graph, queue
(o) URI parsing
(p) Text string manipulation utilities (trim, tab, etc.)

2. BLT.
3. [incr Tk] and [incr Widgets].
4. TkTable.
5. The Standard Tk Library of Tcl/Tk scripts. Currently this includes packages for:
   (a) BWidgets
   (b) mclistbox
6. Img

4.6 Rationale

The small "core" distribution must retain its identity for those applications that embed the Tcl interpreter into constrained environments and require a small footprint. The footprint must remain small, and in fact it should grow smaller, if possible. For example, in the early days of Tcl it was possible at compile time to remove all the file system and exec commands to create a very small Tcl core. There are wide variety of vendors that embed Tcl into, e.g., CAD applications, router firmware, and other limited environments. They only need the basic commands for procedural programming and basic data types.

The larger, bundled distribution must become the standard for desktop distributions (e.g., Linux) so that application writers have a richer set of Tcl commands that they can assume are available. This includes the [incr Tcl] class system and the OS-specific commands provided the TclX and the registry and dde extensions.

The set of packages in the bundled distribution are divided into *mandatory* and *optional* packages. The intent of this distinction is to set a goal for the initial *bundled* distribution, but not close the door to inclusion of other packages. Over time the set of packages in the bundled distribution will surely grow, and some packages may become superceeded by other better packages. The *mandatory set* of packages, however, should be common among all bundles to application writers know what to expect.

In particular, the mandatory set includes [incr Tcl] to promote object oriented programming, Tk to promote easy GUI development, TclX, Dde and Registry to provide access to OS-dependent functionality, and Expect to support automated test environments.

At this time there are a variety of Tk widgets that are optional because there is some overlap and we anticipate continuted evolution of the Tk widget set. I expect that the first bundle will include all the major widget sets, including BLT, [incr Tk] and [incr Widgets], TkTable, the "vu" collection, and possibly Tix.

4.7 The Role of the TCT

The larger bundled distribution will contain packages that are "owned" by the TCT and some that are not. The whole process will be more scalable if responsibility for packages can be split out to other individuals and groups. The role of the TCT should be to set up the infrastructure for the bundled distribution and to make *official* bundled distributions.

4.8 Issues

The main purpose of this proposal is to establish three things:

1. The continued existence of a small Tcl “core” that is identifiable unto its own and useful to various specialized embedded applications.
2. The creation of infrastructure to create bundled distributions. The exact nature of this bundling is not specified. The first bundles may well be created by hand crafted Makefiles and distribution-creation scripts.
3. The set of *mandatory* extensions that should be included in any Tcl bundle. The list in the first draft of this TIP is likely to be wrong, and will surely be amended in the future.
4. Whether further distinctions should be introduced to better support people who wish to target Tcl towards small devices or embedded environments better.

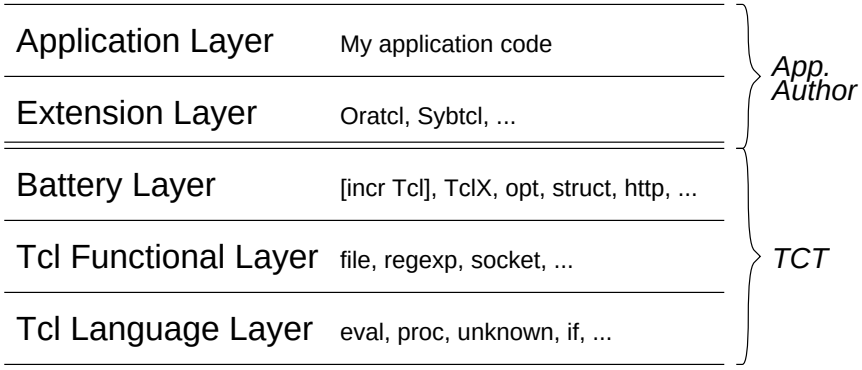

Figure 4.5: Refined Tcl Distribution Architecture

There are a number of related topics that are deliberately outside the scope of this TIP:

1. Documentation format.
2. Network aware downloading of packages and more sophisticated package management.
3. Details of the compile and build environment. Currently there is the TEA standard, and the packages listed in the Mandatory set have all been set up for TEA as part of TclPro.

4.9 Copyright

TIP #5: Make TkClassProcs and TkSetClassProcs Public and Extensible

TIP #5:	**Make TkClassProcs and TkSetClassProcs Public and Extensible**
Author:	Eric Melski ⟨ericm@ajubasolutions.com⟩
Created:	Tuesday, 17th October 2000
Type:	Project
Tcl Version:	8.4
State:	Final
Vote:	Done
Version:	`$Revision: 1.2 $`
Post-History:	

Abstract

At certain critical moments in the lifetime of a Tk widget, Tk will invoke various callbacks on that widget. These callbacks enable the widget to do lots of interesting things, such as react to configuration changes for named fonts, or create and manage truly native widgets (such as the scrollbar widget on Windows platforms). The API for setting up these callbacks for a particular window are, as of Tk 8.3.2, private. This prohibits extension widget authors from fully utilizing this powerful system; those developers can either copy the private declarations into their own source code (leading to future maintenance hassles), or forego the system entirely, hampering their ability to make truly native and well-integrated widgets. This proposal offers an extensible means for making that API public.

5.1 Rationale: Why make TkClassProcs and TkSetClassProcs public?

(The following text is adapted from George Howlett — http://dev.scriptics.com/lists/tclcore/2000/10/msg00143.html)

The Tk toolkit was originally written strictly for Xlib. It created wrappers for many of the Xlib calls. A good example is creating a window. Tk's *Tk_CreateWindow* call in turn calls Xlib's *XCreateWindow*. This is so that the toolkit can perform bookkeeping on the window and manage it in various ways. The down side was that if you needed to pass specific information/flags to the *XCreateWindow* call you couldn't. But this only affected extensions.

Now when Tk 8.0 added native widgets, Tk also had the same problem. For example to create a Win32 button control, you have to pass information through the X emulation layer to the eventual Win32 CreateWindow or CreateWindowEx call.

So the Sun Tk developers created this notion of class procedures. A widget of particular type may need to make different calls at the time the window is created. They added to the TkWindow structure pointers to both the widget instance (i.e. the data the represents the specific widget) and a structure of function pointers (such as one to call when the window is to be created).

```
TkClassProcs tkpButtonProcs =

    CreateProc,             /* createProc. */
    TkButtonWorldChanged,   /* geometryProc. */
    NULL                    /* modalProc. */
};
```

Inside of Tk, such as in *Tk_MakeWindowExist*, code was added to check if the *createProc* of the structure isn't NULL and call that routine to create the native window.

This mechanism was also used to handle font aliasing. I can create a font "fred" that is really a { courier bold } font and use it with any Tk widget.

```
font create fred -family courier -weight bold
button .b -font fred
```

The widget will get the real font and use it in its graphics context. Think of GCs like a pen drawing a particular color. A GC draws with a particular font.

Now if I change the font, the widget's GC must be updated too.

```
font create fred -family helvetica -weight medium
```

You can see where a *geometryProc* is needed to indicate when font aliases change. It gets called for all the widgets using the font.

Another callback is used to handle modal events. This is currently needed only for the Win32 native scrollbar.

So here's the private structure and *TkSetClassProcs* call.

```
typedef Window (TkClassCreateProc) _ANSI_ARGS_((Tk_Window tkwin,
 Window parent, ClientData instanceData));
typedef void (TkClassGeometryProc) _ANSI_ARGS_((ClientData instanceData));
typedef void (TkClassModalProc) _ANSI_ARGS_((Tk_Window tkwin,
 XEvent *eventPtr));

/*
 * Widget class procedures used to implement platform specific widget
 * behavior.
 */

typedef struct TkClassProcs {
    TkClassCreateProc *createProc;
 /* Procedure to invoke when the
```

```
    platform-dependent window needs to be
    created. */
    TkClassGeometryProc *geometryProc;
 /* Procedure to invoke when the geometry of a
    window needs to be recalculated as a result
    of some change in the system. */
    TkClassModalProc *modalProc;
 /* Procedure to invoke after all bindings on a
    widget have been triggered in order to
    handle a modal loop. */
} TkClassProcs;

void
TkSetClassProcs(tkwin, procs, instanceData)
    Tk_Window tkwin;        /* Token for window to modify. */
    TkClassProcs *procs;    /* Class procs structure. */
    ClientData instanceData;/* Data to be passed to class procedures. */
{
    register TkWindow *winPtr = (TkWindow *) tkwin;

    winPtr->classProcsPtr = procs;
    winPtr->instanceData = instanceData;
}
```

Extension developers could not use this interface, however, because it was private to Tk. The original authors of the interface didn't think that anything outside of the Tk widgets would need it. Of course, hindsight is 20-20, and we have since found that this is not true. Extension developers do need to use this system: widget writers that use fonts obviously need to know when a font alias changes, and new Win32 native widgets also need access to this mechanism.

Most extensions authors had/have already found a workaround: copy in the *TkClassProcs* structure and *TkSetClassProcs* routine into your code. However, this workaround leads to future code maintenance problems. Because the structure is private, its members and usage are not guaranteed to remain constant between versions of Tk. If the structure changes, the extension authors have to update all of their code accordingly.

Making the system public locks in the format and usage of the system, so that extension authors can rely on it existing from one version to the next, and they will no longer have to maintain parallel redundant copies of the structure and function definition.

5.2 Rationale: Why make TkClassProcs and TkSetClassProcs extensible?

Every time we've made a public structure, we've regretted it later when we needed to extend it to handle some new feature that we didn't originally anticipate. In general we should avoid designing new API's that preclude making future changes without introducing incompatibilities. http://dev.scriptics.com/lists/tclcore/2000/10/msg00083.html

This system is one that seems likely to require extension in the future. There are currently three callbacks: create window, geometry change, and modal event. Already one request to extend the mechanism has been made, to support the notion of a "client area" related to geometry management and labelled frame widgets (http://dev.scriptics.com/lists/tclcore/2000/10/msg00121.html and http://dev.scriptics.com/lists/tclcore/2000/10/msg00170.html). Another possible extension is a focus management callback, to allow for smoother focus transitions between native widgets and Tk widgets; note that this focus management callback is a purely hypothetical extension at this time.

If the system is one that we are likely to want to extend with additional callbacks in the future, it behooves us to make it public in a manner that allows us to extend it while causing the minimum amount of disruption for extension authors. There are two concerns here. First is binary compatibility: will an extension compiled against a version of Tk which features the base (three callback) *TkClassProcs* system work with a version of Tk that

features an extended *TkClassProcs* system? Second is source compatibility: will an extension author have to update their sources when they want to recompile their extension against a version of Tk that features an extended *TkClassProcs* system? Ideally, the system that we make public will allow extension while retaining binary and source compatibility between versions of Tk.

5.3 Specification

I propose that the following steps be taken to make *TkClassProcs* and *TkSetClassProcs* public:

1. Rename *TkClassProcs* to *Tk_ClassProcs*; rename *TkSetClassProcs* to *Tk_SetClassProcs*; rename *TkClassCreateProc*, etc., to *Tk_ClassCreateProc*, etc. Move the structure definition, function prototype, and callback typedefs from tkInt.h to tk.h. This is in keeping with Tk public interface naming conventions.

2. Add a single size field to the *Tk_ClassProcs* structure. This field is initialized at the time that the structure is allocated, and always contains the size of the structure. This field will be used to provide a simple versioning scheme for the structure. Portions of Tk that use the class proc callbacks will inspect this size field to ascertain whether or not a particular instance of the *Tk_ClassProcs* structure is of a version that contains a given callback. See the example below.

3. Rename the *geometryProc* callback to *worldChangedProc*. The name *geometryProc* is somewhat misleading. Currently, the callback is used only to support font aliasing, as described above. This is sort of geometry related, but it doesn't necessarily mean that geometry of the widget must change, it just means that the widget will have to update its world view to reflect the current state of the world. In addition, the callback will likely be used to support color aliasing when that is added to Tk (imagine defining a color "myColor" to mean "#c4d3a2" and then configuring widgets to use "myColor" instead of the literal value; this provides all the benefits for colors that font aliasing does for fonts). When that is done, *geometryProc* will be truly misleading, since a color change probably does not mean a geometry change for the widget.

4. Change the order of the callback fields in the *Tk_ClassProcs* structure, making *worldChangedProc* the first of the callbacks listed in the structure. In the existing private *TkClassProcs* structure, the first callback is the *createProc*. It is not strictly necessary to make *worldChangedProc* the first callback. However, most widgets in Tk (canvas, entry, scale, text, message, listbox, menu, menubuttons, scrollbars on Unix and Mac, and buttons on Unix and Mac) use only this callback. Making it first in the structure (after the size field, which must be the very first entry) means a little bit less work for widget authors in the common case, because they need not include the NULL declaration for the *createProc* slot in the structure. Compare:

   ```
   static Tk_ClassProcs myClassProcs = {
       sizeof(Tk_ClassProcs), NULL, myWorldChangedProc
   };
   ```

   with:

   ```
   static Tk_ClassProcs myClassProcs = {
       sizeof(Tk_ClassProcs), myWorldChangedProc
   };
   ```

   Since the *createProc* is used so infrequently, why require that all widget authors explicitly declare it to be NULL? This change just simplifies everybody's life that much more.

Usage of the public API will be very similar to usage of the existing private API:

```
static Tk_ClassProcs myClassProcs = {
    sizeof(Tk_ClassProcs),
    myWorldChangedProc
};

static int Tk_MyWidgetObjCmd(...) {
    ...
    Tk_SetClassProcs(widgetPtr->tkwin,myClassProcs,(ClientData)widgetPtr);
```

```
    ...
    return TCL_OK;
}
```

Portions of Tk that need to use a particular callback, such as *Tk_MakeWindowExist*, use code like the following:

```
Tk_ClassProcs *thisClassProcs = tkwin->classProcs;
createProc *procPtr;

/* Make sure the structure we were given has the createProc field
 * in it by checking that the size of the structure is at least
 * big enough to have that slot.
 */

if (thisClassProcs->size <= Tk_Offset(Tk_ClassProcs, createProc)) {
    procPtr = NULL;
} else {
    procPtr = thisClassProcs->createProc;
}

if (procPtr != NULL) {
    /* Invoke the createProc for this window. */
    ...
} else {
    /* Use the default Tk window creation mechanism. */
    ...
}
```

5.4 Benefits of this implementation

Benefits of this implementation are as follows:

1. Usage of *Tk_ClassProcs* and *Tk_SetClassProcs* very, very closely parallels the usage of the existing private API. In fact, the only difference is a small change in the particular fields of the *Tk_ClassProcs* structure (especially, the new size field, for version information, and the reordering of the callback fields).

2. All instances of "mywidget" reference the same *Tk_ClassProcs* structure. This is memory efficient.

3. We do not need to explicitly initialize to NULL those fields of myClassProcs that we don't use. The ANSI C specification states that static variables (and members of statically declared structures) that are not explicitly initialized are initialized to zero.

4. This retains binary compatibility. The size field of the *Tk_ClassProcs* structure is set at compile time, so when a later version of Tk checks the size field to see if a new callback can be used, it will fail. That is, if extension author A compiles the extension against version X of Tk, which has three fields in *Tk_ClassProcs*, the size field of myClassProcs will be set to 12 (assuming 4-byte pointers). When using that extension with Tk version Y, which may have four fields in *Tk_ClassProcs*, the size check for that fourth field will fail, since the size field, set to 12, will be less than or equal to the offset of the fourth field in the structure.

5. This retains source compatibility. Because of #3 above, unless the extension author wants to use the new callbacks, they need not worry about their addition, because the new fields will be automatically set to zero.

6. There is minimal API bloat. Only one public API is added, *Tk_SetClassProcs*.

7. The system is "type safe" with respect to the function signatures of the callback functions. Any type mismatches will be caught at compile time.

8. If desired, widget authors can directly reference elements of the *Tk_ClassProcs* structure:

   ```
   myClassProcs.createProc = myCreateProc;
   ```

5.5 Drawbacks of this implementation

The drawbacks of this implementation are as follows:

1. The required value of the size field will seem like a bit of black magic to developers new to the system. The question *"Why does this field have to be set to this value? If it's always the same thing, why is it stored at all?"* Of course, experienced programmers will recognize why it has to be set, and that in fact, it is not always the same value. This issue can best be addressed by appropriate documentation.

2. Extensions that use the existing private *TkClassProcs* and *TkSetClassProcs* mechanism and which were compiled against versions of Tk <= 8.3 will not work with new versions of Tk, since the format of the *Tk_ClassProcs* structure will change. However, this is the consequence of using private structures and API's in your extensions: when those private structures and API's change, you have to update your extension accordingly. We cannot allow ourselves to be overly constrained by this issue. The existing mechanism is private, period. Authors that use it do so knowingly and willfully.

5.6 Reference Implementation

http://sourceforge.net/patch/?func=detailpatch&patch_id=102213&group_id=10894

5.7 Copyright

TIP #6: Include [Incr Tcl] in the Core Tcl distribution

TIP #6:	**Include [Incr Tcl] in the Core Tcl distribution**
Author:	Mark Harrison ⟨markh@usai.asiainfo.com⟩
Created:	Monday, 16th October 2000
Type:	Project
Tcl Version:	8.4.0
State:	Rejected
Vote:	Done
Version:	`$Revision:  1.5 $`
Post-History:	

Abstract

Include [Incr Tcl] in the Core Tcl distribution.

6.1 Proposal

[incr Tcl] (http://tcltk.com/itcl/) shall be included in the core Tcl distribution. It shall be included in the Tcl source tree, and built as part of the standard Tcl distribution.

Specific items:

- "itclsh" will not be included
- "itcl_*" commands will not be included
- everything will move from ::itcl to ::
- the "find" subcommands will be reintegrated into "info"

6.2 Rationale

The lack of a standard object and data abstraction system continues to hinder Tcl development.

> "Lets face it, not including any sort of OO system is one of the major failings of Tcl. Indexing into global arrays is a sad hack when compared to a real OO system." - *Mo DeJong* ⟨*mdejong@cygnus.com*⟩

Earlier this year, it seemed that it would finally be included in Tcl 8.4, but that plan was rescinded.

Note that this is distinct from the "batteries included" (BI) distribution, and is not intended to be a model for building the BI distribution. It is a special case for inclusion in the core tcl command set, since a "class" command is a fundamental language construct.

6.3 Alternatives

Include [incr Tcl] in a "batteries included" (BI) distribution.

Many people will not opt for the BI distribution ([TIP #4]) due to its larger size. It is quite likely that (for example) a Linux distribution my include Tcl as a standard component, but place the BI on a supplemental disk.

6.4 Objections

I don't want any object system included!

You can delete the [incr Tcl] library with no harm to your code.

John Ousterhout hates objects!

This misconception is primarily due to a misreading of one of his papers. A better summary of his position is that "scripting is a better solution than objects in many cases." John Ousterhout has told me that he will not stand in the way of adding object-oriented programming to Tcl.

[incr Tcl]'s object model is not good!

[incr Tcl] supports the same object model as C++ and Java. Many programmers are familiar with this model and accept it as a good model.

The CLOS object model is better!

Quoting John Ousterhout, "People vote with their feet". For whatever reason, slot-based systems failed to gain as much popularity as C++/Java-like systems.

There are many Tcl object systems to choose from!

None are even a fraction as long-lived, popular, or well-supported as [incr Tcl].

6.5 Special Provisions

Since [incr Tcl] still exists as a separately named entity, this TIP shall not be construed as relieving any individual from the responsibility of providing appropriate [incr Apparel].

6.6 Copyright

TIP #7: Increased resolution for TclpGetTime on Windows

TIP #7:	**Increased resolution for TclpGetTime on Windows**
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
Created:	Thursday, 26th October 2000
Type:	Project
Tcl Version:	8.4
State:	Final
Vote:	Done
Version:	`$Revision: 1.3 $`
Post-History:	
Discussions-To:	news:comp.lang.tcl

Abstract

Tcl users on the Windows platform have long been at a disadvantage in attempting to do code timing studies, owing to the poor resolution of the Windows system clock. The *time* command, the *clock clicks* command, and all related functions are limited to a resolution of (typically) 10 milliseconds. This proposal offers a solution based on the Windows performance counter. It presents a means of disciplining this counter to the system clock so that *TclpGetTime* (the underlying call that the above commands use) can return times to microsecond precision with accuracy in the tens of microseconds.

7.1 Change history

2 November 2000: Modified the TIP to discuss the issues surrounding the fact that some multiprocessor kernels on Windows NT use the CPU timestamp counter as a performance counter. Modified the proposed patch to test for the two frequencies in common use on 8254-compatible real-time clocks, and enable using the performance counter only if its frequency matches one of them. Included the proposed patch inline for review rather than as a pointer off to dejanews.

7.2 Rationale

The Windows implementation of *TclpGetTime*, as of Tcl 8.3.2, uses the *ftime* call in the C library to extract the current system clock in seconds and milliseconds. While this time value has millisecond precision, its actual resolution is limited by the tick rate of the Windows system clock, normally 100 Hz. Similarly, *TclpGetClicks* uses the *GetTickCount* function of *kernel32.dll* to get the number of milliseconds since bootload; once again, the actual resolution of this call is limited to the tick rate of the system clock.

The Windows Platform APIs offer several timers of different accuracy. The most precise of these is *QueryPerformanceCounter*, which operates at an unspecified frequency (returned by *QueryPerformanceFrequency*) that is typically about 1.19 MHz. http://support.microsoft.com/support/kb/articles/Q172/3/38.asp has details of the call, with sample code.

The documentation for Windows suggests that this function is available only on certain versions of the operating system; in fact, it is implemented in every extant version of Win32 with the exception of Win32s and Windows CE 1.0. Since Visual C++ 6, on which the Tcl distribution depends, will no longer compile code for those two platforms, I assert that they may be safely ignored.

The documentation for Windows also states that *QueryPerformanceCounter* is available only on certain hardware. In practice, this is not an issue; I have never encountered a Windows implementation on an x86 platform that lacks it, and Alpha has it as well. In any case, the reference implementation tests for the success or failure of the system calls in question, and falls back on the old way of getting time should they return an error indication. Users of any platform on which the performance counter is not supported should therefore be no worse off than they have ever been.

A worse problem with the performance counter is that its frequency is poorly calibrated, and is frequently off by as much as 200 parts per million. Moreover, the frequency drifts over time, frequently having a sensitive dependency to temperatures inside the computer's case.

This problem is not insurmountable. The fix is to maintain the observed frequency of the performance counter (calibrated against the system clock) as a variable at run time, and use that variable together with the value of the performance counter to derive Tcl's concept of the time. This technique is well known to electronic engineers as the "phase locked loop" and is used in network protocols such as NTP (http://www.eecis.udel.edu/~ntp/).

One problem that is apparently insurmountable is that certain multiprocessor systems have hardware abstraction layers that derive the performance counter from the CPU timestamp counter in place of a real-time clock reference. This implementation causes the performance counter on one CPU to drift with respect to the other over time; if a thread is moved from one processor to another, it cannot derive a meaningful result from comparing two successive values of the counter. Moreover, if the CPU clock uses a "gearshift" technique for power management (as on Intel SpeedStep or Transmeta machines), the CPU timestamp counter ticks at a non-constant rate.

The proposed implementation addresses the problem by using the performance counter only if its nominal frequency is either 1.193182 MHz or 3.579545 MHz. These two frquencies are the common rates when 8254-compatible real-time clock chips are used; virtually all PCI bus controllers have such chips on board. This solution therefore adapts to the vast majority of workstation-class Windows boxes, and is virtually certain to exclude implementations derived from the CPU clock since no modern CPU is that slow.

The patch has been tested on several desktop and laptop machines from Compaq, Dell, Gateway, HP, Micron, and Packard Bell, with processors ranging from a 50 MHz 486 to a 750 MHz Pentium III, including laptops using SpeedStep technology. It passes the clock-related test cases on all these platforms; it falls back to the old clocks with 10-ms precision on multiprocessor servers from Compaq and HP. (Using the performance counter actually would have worked on the HP server, which apparently has some way of making sure that the results

of *QueryPerformanceCounter* are consistent from one CPU to another. The performance counter on the Compaq machine was observed to be inconsistent between the two CPU's.)

7.3 Specification

This document proposes the following changes to the Tcl core:

1. (tclWinTime.c) Add to the static data a set of variables that manage the phase-locked techniques, including a *CRITICAL_SECTION* to guard them so that multi-threaded code is stable.
2. (tclWinTime.c) Modify *TclpGetSeconds* to call *TclpGetTime* and return the 'seconds' portion of the result. This change is necessary to make sure that the two times are consistent near the rollover from one second to another.
3. (tclWinTime.c) Modify *TclpGetClicks* to use TclpGetTime to determine the click count as a number of microseconds.
4. (tclWinTime.c) Modify *TclpGetTime* to return the time as M*Q+B, where Q is the result of *QueryPerformanceCounter*, and M and B are variables maintained by the phase-locked loop to keep the result as close as possible to the system clock. The *TclpGetTime* call will also launch the phase-lock management in a separate thread the first time that it is invoked. If the performance counter is unavailable, or if its frequency is not one of the two common 8254-compatible rates, then *TclpGetTime* will return the result of *ftime* as it does in Tcl 8.3.2.
5. (tclWinTime.c) Add the clock calibration procedure. The calibration is somewhat complex; to save space, the reader is referred to the reference implementation for the details of how the time base and frequency are maintained.
6. (tclWinNotify.c) Modify *Tcl_Sleep* to test that the process has, in fact, slept for the requisite time by calling *TclpGetTime* and comparing with the desired time. Otherwise, roundoff errors may cause the process to awaken early.
7. (tclWinTest.c) Add a *testwinclock* command. This command returns a four element list comprising the seconds and microseconds portions of the system clock and the seconds and microseconds portions of the Tcl clock.
8. (winTime.test) Add to the test suite a test that makes sure that the Tcl clock stays within 1.1 ms of the system clock over the duration of the test.

7.4 Reference implementation

This change was submitted as a patch to the old bug-tracking system at Scriptics (http://www.deja.com/getdoc.xp?AN=666545441&fmt=text). It is being recycled as a TIP now that the Tcl Core Team is in place, since the process for advancing the old patches to the Core is not well defined. The link above should not be used to retrieve the current version of the patch, which appears below as an Appendix.

Tests on several Wintel boxes have shown that the initial startup transient is less than about 10 seconds (during which time the Tcl clock may be running 500 ppm fast or slow to bring it into step); following this period, the motion of the Tcl clock is highly repeatable and uniform.

If the system clock changes by more than 1 second during a run, as when the operator sets it using the eyeball-and-wristwatch method, the method of adjusting the performance frequency to preserve monotonicity and accuracy of interval measurements is hopeless. This is the only case where the Tcl clock is allowed to jump.

The startup of the calibration loop does not introduce new instabilities in the behavior of [clock clocks] or *TclpGetTime*.

[clock clicks] and other times that derive from *TclpGetTime* also ought to be reliable from the beginning — assuming that *QueryPerformanceFrequency* actually matches the crystal. The worst case while the initial calibration

is going on ought to be that the Tcl clock runs 0.1% fast or slow. The point of the calibration loop is to correct for long-term drift.

The problem, otherwise, is that *QueryPerformanceFrequency* may be off by some tens of parts per million with respect to the system clock. Over a period of days, that would cause the Tcl clock to veer off from the system clock. For instance, once my machine is warmed up (temperature is significant, believe it or not), *QueryPerformanceFrequency* is consistently 0.99985 of the correct value; without calibration, the performance-counter-derived clock drifts 13 seconds per day.

The *capture transient* of the calibration loop is a little different every time, but the one shown below is typical. The Tcl time starts out 2 ms fast with respect to the system time, and the initial estimate of performance frequency is off, too. At 2 seconds in, the calibration loop takes over and makes the clock run 0.1% slow to bring it in line; by 5 seconds in, it's lined up. There's some phase noise over the next 40 seconds or so, by which time the performance frequency is locked on quite closely. The outliers above the line represent the fact that [after] events sometimes arrive late because of various other things going on in Windows.

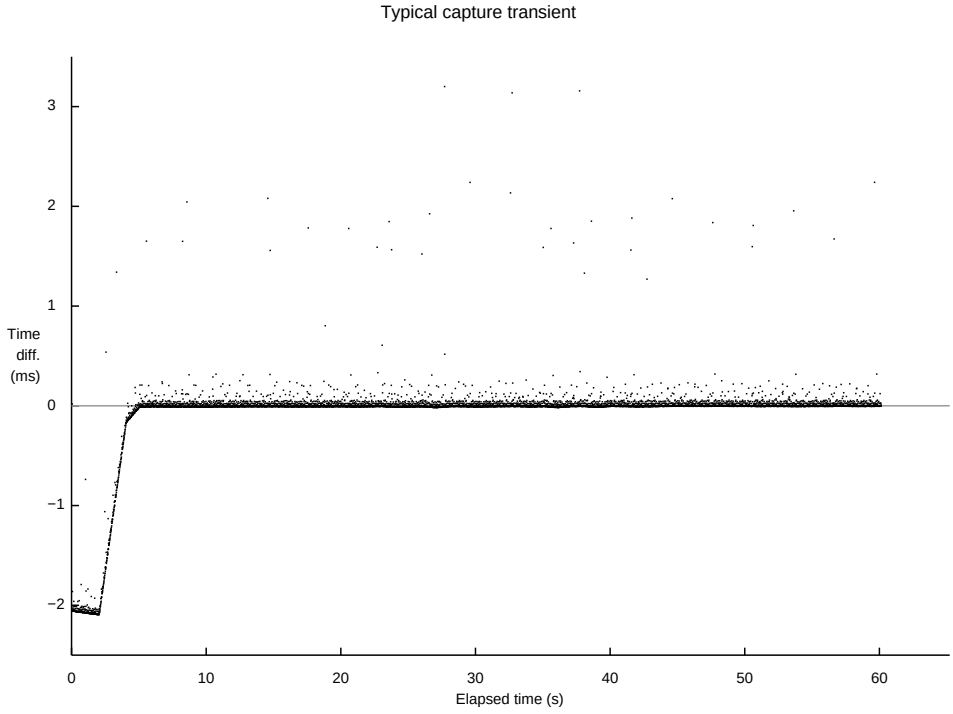

Figure 7.6: Typical capture transient

The script that gathered the raw data plotted above appears below.

```
foreach { syssecs sysusec tclsecs tclusec } [testwinclock] {}
set basesecs $syssecs
set baseusec $sysusec
set nTrials 10000
for { set i 0 } { $i < $nTrials } { incr i } {
    set values {}
    for { set j 0 } { $j < 5 } { incr j } {
foreach { syssecs sysusec tclsecs tclusec } [testwinclock] {}
set systime [expr { ($syssecs - $basesecs)
    + 1.0e-6 * $sysusec - 1.0e-6 * $baseusec }]
set tcltime [expr { ($tclsecs - $basesecs)
    + 1.0e-6 * $tclusec - 1.0e-6 * $baseusec }]
set timediff [expr { $tcltime - $systime }]
lappend values [list $systime $timediff $tcltime]
after 1
```

```
    }
    foreach { elapsed timediff tcltime } \
[lindex [lsort -real -index 1 $values] 0] {}
    lappend history $elapsed $timediff $tcltime
}
set f [open ~/test2.dat w]
foreach { elapsed timediff tcltime} $history {
    puts $f "$elapsed\t$timediff\t$tcltime"
}
close $f
```

To quantify how reproducible the measurements are, I threw a patched tclsh the torture test of executing [time {}] ten million times, and made a histogram of the results. The figure below shows the results. The dots represent individual sample bins, and the solid line is the cumulative count of samples. The vast majority of samples show either five or six microseconds. 99.9% take fewer than nine. There are many samples that take longer, owing to either servicing interrupts or losing the processor to other processes.

The lines at 21, 31 and 42 microseconds show up in repeated runs on my machine; I suspect that they represent time spent servicing different sorts of video interrupts. It's less clear to me what the other outliers might be; Windows has a tremendous amount of stuff going on even when it's apparently idle.

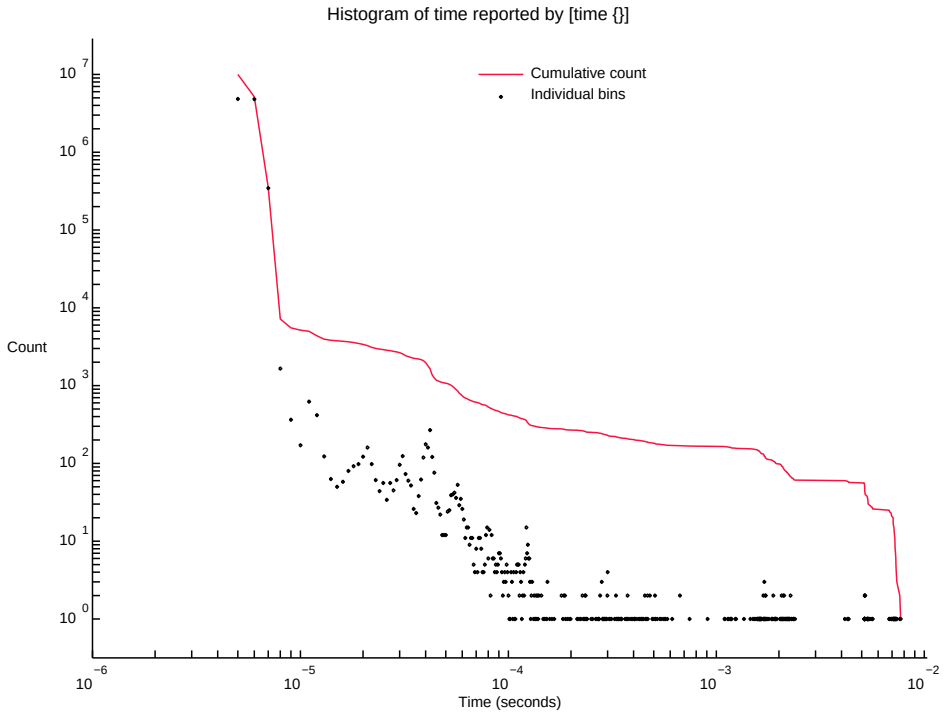

Figure 7.7: Histogram of results of [time {}].

All tests in the test suite continue to pass with the patch applied.

7.5 Notes

If you care about time to the absolute precision that this change can achieve, it is of course necessary to discipline the Windows system clock as well. Perhaps the best way is to use one of the available NTP packages (see http://www.eecis.udel.edu/~ntp/ for further information).

7.6 Copyright

This document has been placed in the public domain.

7.7 Appendix

The proposed set of patches to the Tcl 8.3.2 code base appears here.

```
*** ../tcl8.3.2base/src/tcl8.3.2/win/tclWinNotify.c Fri Jul  2 18:08:30 1999
--- ./src/tcl8.3.2/win/tclWinNotify.c Thu Aug 24 23:29:12 2000
***************
*** 510,514 ****
  Tcl_Sleep(ms)
      int ms; /* Number of milliseconds to sleep. */
  {
!     Sleep(ms);
  }
--- 510,548 ----
  Tcl_Sleep(ms)
      int ms; /* Number of milliseconds to sleep. */
  {
!     /*
!      * Simply calling 'Sleep' for the requisite number of milliseconds
!      * can make the process appear to wake up early because it isn't
!      * synchronized with the CPU performance counter that is used in
!      * tclWinTime.c.  This behavior is probably benign, but messes
!      * up some of the corner cases in the test suite.  We get around
!      * this problem by repeating the 'Sleep' call as many times
!      * as necessary to make the clock advance by the requisite amount.
!      */
!
!     Tcl_Time now; /* Current wall clock time */
!     Tcl_Time desired; /* Desired wakeup time */
!     int sleepTime = ms; /* Time to sleep */
!
!     TclpGetTime( &now );
!     desired.sec = now.sec + ( ms / 1000 );
!     desired.usec = now.usec + 1000 * ( ms % 1000 );
!     if ( desired.usec > 1000000 ) {
!  ++desired.sec;
!  desired.usec -= 1000000;
!     }
!
!     for ( ; ; ) {
!  Sleep( sleepTime );
!  TclpGetTime( &now );
!  if ( now.sec > desired.sec ) {
!      break;
!  } else if ( ( now.sec == desired.sec )
!       && ( now.usec >= desired.usec ) ) {
!      break;
!  }
!  sleepTime = ( ( 1000 * ( desired.sec - now.sec ) )
!        + ( ( desired.usec - now.usec ) / 1000 ) );
!     }
!
  }
*** ../tcl8.3.2base/src/tcl8.3.2/win/tclWinTest.c Thu Oct 28 23:05:14 1999
--- ./src/tcl8.3.2/win/tclWinTest.c Mon Sep  4 22:45:56 2000
***************
```

```
*** 22,27 ****
--- 22,31 ----
  static int TestvolumetypeCmd _ANSI_ARGS_((ClientData dummy,
 Tcl_Interp *interp, int objc,
 Tcl_Obj *CONST objv[]));
+ static int      TestwinclockCmd _ANSI_ARGS_(( ClientData dummy,
+        Tcl_Interp* interp,
+        int objc,
+        Tcl_Obj *CONST objv[] ));
  ^L
  /*
   *----------------------------------------------------------------------
***************
*** 52,57 ****
--- 56,63 ----
       (ClientData) 0, (Tcl_CmdDeleteProc *) NULL);
      Tcl_CreateObjCommand(interp, "testvolumetype", TestvolumetypeCmd,
       (ClientData) 0, (Tcl_CmdDeleteProc *) NULL);
+     Tcl_CreateObjCommand(interp, "testwinclock", TestwinclockCmd,
+             (ClientData) 0, (Tcl_CmdDeleteProc *) NULL);
      return TCL_OK;
  }
  ^L
***************
*** 187,190 ****
--- 193,267 ----
      Tcl_SetResult(interp, volType, TCL_VOLATILE);
      return TCL_OK;
  #undef VOL_BUF_SIZE
+ }
+ ^L
+ /*
+  *----------------------------------------------------------------------
+  *
+  * TestclockCmd --
+  *
+  * Command that returns the seconds and microseconds portions of
+  * the system clock and of the Tcl clock so that they can be
+  * compared to validate that the Tcl clock is staying in sync.
+  *
+  * Usage:
+  * testclock
+  *
+  * Parameters:
+  * None.
+  *
+  * Results:
+  * Returns a standard Tcl result comprising a four-element list:
+  * the seconds and microseconds portions of the system clock,
+  * and the seconds and microseconds portions of the Tcl clock.
+  *
+  * Side effects:
+  * None.
+  *
+  *----------------------------------------------------------------------
+  */
+
+ static int
+ TestwinclockCmd( ClientData dummy,
+  /* Unused */
+   Tcl_Interp* interp,
+  /* Tcl interpreter */
```

```
+   int objc,
+  /* Argument count */
+   Tcl_Obj *CONST objv[] )
+  /* Argument vector */
+ {
+     CONST static FILETIME posixEpoch = { 0xD53E8000, 0x019DB1DE };
+  /* The Posix epoch, expressed as a
+   * Windows FILETIME */
+     Tcl_Time tclTime; /* Tcl clock */
+     FILETIME sysTime; /* System clock */
+     Tcl_Obj* result; /* Result of the command */
+     LARGE_INTEGER t1, t2;
+
+     if ( objc != 1 ) {
+  Tcl_WrongNumArgs( interp, 1, objv, "" );
+  return TCL_ERROR;
+     }
+
+     TclpGetTime( &tclTime );
+     GetSystemTimeAsFileTime( &sysTime );
+     t1.LowPart = posixEpoch.dwLowDateTime;
+     t1.HighPart = posixEpoch.dwHighDateTime;
+     t2.LowPart = sysTime.dwLowDateTime;
+     t2.HighPart = sysTime.dwHighDateTime;
+     t2.QuadPart -= t1.QuadPart;
+
+     result = Tcl_NewObj();
+     Tcl_ListObjAppendElement
+  ( interp, result, Tcl_NewIntObj( (int) (t2.QuadPart / 10000000 ) ) );
+     Tcl_ListObjAppendElement
+  ( interp, result,
+    Tcl_NewIntObj( (int) ( (t2.QuadPart / 10 ) % 1000000 ) ) );
+     Tcl_ListObjAppendElement( interp, result, Tcl_NewIntObj( tclTime.sec ) );
+     Tcl_ListObjAppendElement( interp, result, Tcl_NewIntObj( tclTime.usec ) );
+
+     Tcl_SetObjResult( interp, result );
+
+     return TCL_OK;
  }
*** ../tcl8.3.2base/src/tcl8.3.2/win/tclWinTime.c Tue Nov 30 19:08:44 1999
--- ./src/tcl8.3.2/win/tclWinTime.c Thu Nov  2 14:25:56 2000
***************
*** 38,47 ****
--- 38,114 ----
  static Tcl_ThreadDataKey dataKey;

  /*
+  * Calibration interval for the high-resolution timer, in msec
+  */
+
+ static CONST unsigned long clockCalibrateWakeupInterval = 10000;
+  /* FIXME: 10 s -- should be about 10 min! */
+
+ /*
+  * Data for managing high-resolution timers.
+  */
+
+ typedef struct TimeInfo {
+
+     CRITICAL_SECTION cs; /* Mutex guarding this structure */
+
+     int initialized; /* Flag == 1 if this structure is
```

```
+   * initialized. */
+
+     int perfCounterAvailable; /* Flag == 1 if the hardware has a
+   * performance counter */
+
+     HANDLE calibrationThread; /* Handle to the thread that keeps the
+   * virtual clock calibrated. */
+
+     HANDLE readyEvent; /* System event used to
+   * trigger the requesting thread
+   * when the clock calibration procedure
+   * is initialized for the first time */
+
+     /*
+      * The following values are used for calculating virtual time.
+      * Virtual time is always equal to:
+      *    lastFileTime + (current perf counter - lastCounter)
+      * * 10000000 / curCounterFreq
+      * and lastFileTime and lastCounter are updated any time that
+      * virtual time is returned to a caller.
+      */
+
+     ULARGE_INTEGER lastFileTime;
+     LARGE_INTEGER lastCounter;
+     LARGE_INTEGER curCounterFreq;
+
+     /*
+      * The next two values are used only in the calibration thread, to track
+      * the frequency of the performance counter.
+      */
+
+     LONGLONG lastPerfCounter; /* Performance counter the last time
+   * that UpdateClockEachSecond was called */
+     LONGLONG lastSysTime; /* System clock at the last time
+   * that UpdateClockEachSecond was called */
+     LONGLONG estPerfCounterFreq;
+  /* Current estimate of the counter frequency
+   * using the system clock as the standard */
+
+ } TimeInfo;
+
+ static TimeInfo timeInfo = {
+     NULL, 0, 0, NULL, NULL, 0, 0, 0, 0, 0
+ };
+
+ CONST static FILETIME posixEpoch = { 0xD53E8000, 0x019DB1DE };
+
+ /*
    * Declarations for functions defined later in this file.
    */

  static struct tm * ComputeGMT _ANSI_ARGS_((const time_t *tp));
+
+ static DWORD WINAPI     CalibrationThread _ANSI_ARGS_(( LPVOID arg ));
+
+ static void  UpdateTimeEachSecond _ANSI_ARGS_(( void ));
  ^L
  /*
   *----------------------------------------------------------------------
***************
*** 63,69 ****
  unsigned long
```

```
  TclpGetSeconds()
  {
!     return (unsigned long) time((time_t *) NULL);
  }
  ^L
  /*
--- 130,138 ----
  unsigned long
  TclpGetSeconds()
  {
!     Tcl_Time t;
!     TclpGetTime( &t );
!     return t.sec;
  }
  ^L
  /*
***************
*** 89,95 ****
  unsigned long
  TclpGetClicks()
  {
!     return GetTickCount();
  }
  ^L
  /*
--- 158,175 ----
  unsigned long
  TclpGetClicks()
  {
!     /*
!      * Use the TclpGetTime abstraction to get the time in microseconds,
!      * as nearly as we can, and return it.
!      */
!
!     Tcl_Time now; /* Current Tcl time */
!     unsigned long retval; /* Value to return */
!
!     TclpGetTime( &now );
!     retval = ( now.sec * 1000000 ) + now.usec;
!     return retval;
!
  }
  ^L
  /*
***************
*** 134,140 ****
   * Returns the current time in timePtr.
   *
   * Side effects:
!  * None.
   *
   *----------------------------------------------------------------------
   */
--- 214,226 ----
   * Returns the current time in timePtr.
   *
   * Side effects:
!  * On the first call, initializes a set of static variables to
!  * keep track of the base value of the performance counter, the
!  * corresponding wall clock (obtained through ftime) and the
!  * frequency of the performance counter.  Also spins a thread
!  * whose function is to wake up periodically and monitor these
```

```
!  * values, adjusting them as necessary to correct for drift
!  * in the performance counter's oscillator.
   *
   *----------------------------------------------------------------------
   */
***************
*** 143,153 ****
  TclpGetTime(timePtr)
      Tcl_Time *timePtr; /* Location to store time information. */
  {
      struct timeb t;

!     ftime(&t);
!     timePtr->sec = t.time;
!     timePtr->usec = t.millitm * 1000;
  }
  ^L
  /*
--- 229,342 ----
  TclpGetTime(timePtr)
      Tcl_Time *timePtr; /* Location to store time information. */
  {
+
      struct timeb t;

!     /* Initialize static storage on the first trip through. */
!
!     /*
!      * Note: Outer check for 'initialized' is a performance win
!      * since it avoids an extra mutex lock in the common case.
!      */
!
!     if ( !timeInfo.initialized ) {
!  TclpInitLock();
!  if ( !timeInfo.initialized ) {
!      timeInfo.perfCounterAvailable
!  = QueryPerformanceFrequency( &timeInfo.curCounterFreq );
!
!      /*
!       * Some hardware abstraction layers use the CPU clock
!       * in place of the real-time clock as a performance counter
!       * reference.  This results in:
!       *    - inconsistent results among the processors on
!       *      multi-processor systems.
!       *    - unpredictable changes in performance counter frequency
!       *      on "gearshift" processors such as Transmeta and
!       *      SpeedStep.
!       * There seems to be no way to test whether the performance
!       * counter is reliable, but a useful heuristic is that
!       * if its frequency is 1.193182 MHz or 3.579545 MHz, it's
!       * derived from a colorburst crystal and is therefore
!       * the RTC rather than the TSC.  If it's anything else, we
!       * presume that the performance counter is unreliable.
!       */
!
!      if ( timeInfo.perfCounterAvailable
!   && timeInfo.curCounterFreq.QuadPart != 1193182ui64
!   && timeInfo.curCounterFreq.QuadPart != 3579545ui64 ) {
!  timeInfo.perfCounterAvailable = FALSE;
!      }
!
!      /*
```

```
!       * If the performance counter is available, start a thread to
!       * calibrate it.
!       */
!
!      if ( timeInfo.perfCounterAvailable ) {
!   DWORD id;
!   InitializeCriticalSection( &timeInfo.cs );
!   timeInfo.readyEvent = CreateEvent( NULL, FALSE, FALSE, NULL );
!   timeInfo.calibrationThread = CreateThread( NULL,
!      8192,
!      CalibrationThread,
!      (LPVOID) NULL,
!      0,
!      &id );
!   SetThreadPriority( timeInfo.calibrationThread,
!      THREAD_PRIORITY_HIGHEST );
!   WaitForSingleObject( timeInfo.readyEvent, INFINITE );
!   CloseHandle( timeInfo.readyEvent );
!       }
!       timeInfo.initialized = TRUE;
!   }
!   TclpInitUnlock();
!      }
!
!      if ( timeInfo.perfCounterAvailable ) {
!
!   /*
!    * Query the performance counter and use it to calculate the
!    * current time.
!    */
!
!   LARGE_INTEGER curCounter;
!   /* Current performance counter */
!
!   LONGLONG curFileTime;
!   /* Current estimated time, expressed
!    * as 100-ns ticks since the Windows epoch */
!
!   static const LARGE_INTEGER posixEpoch = { 0xD53E8000, 0x019DB1DE };
!   /* Posix epoch expressed as 100-ns ticks
!    * since the windows epoch */
!
!   LONGLONG usecSincePosixEpoch;
!   /* Current microseconds since Posix epoch */
!
!   EnterCriticalSection( &timeInfo.cs );
!
!   QueryPerformanceCounter( &curCounter );
!   curFileTime = timeInfo.lastFileTime.QuadPart
!       + ( ( curCounter.QuadPart - timeInfo.lastCounter.QuadPart )
!   * 10000000 / timeInfo.curCounterFreq.QuadPart );
!   timeInfo.lastFileTime.QuadPart = curFileTime;
!   timeInfo.lastCounter.QuadPart = curCounter.QuadPart;
!   usecSincePosixEpoch = ( curFileTime - posixEpoch.QuadPart ) / 10;
!   timePtr->sec = (time_t) ( usecSincePosixEpoch / 1000000 );
!   timePtr->usec = (unsigned long ) ( usecSincePosixEpoch % 1000000 );
!
!   LeaveCriticalSection( &timeInfo.cs );
!
!
!      } else {
!
```

```
!  /* High resolution timer is not available.  Just use ftime */
!
!  ftime(&t);
!  timePtr->sec = t.time;
!  timePtr->usec = t.millitm * 1000;
!     }
  }
  ^L
  /*
***************
*** 439,442 ****
--- 628,843 ----
      }

      return tmPtr;
+ }
+ ^L
+ /*
+  *----------------------------------------------------------------------
+  *
+  * CalibrationThread --
+  *
+  * Thread that manages calibration of the hi-resolution time
+  * derived from the performance counter, to keep it synchronized
+  * with the system clock.
+  *
+  * Parameters:
+  * arg -- Client data from the CreateThread call.  This parameter
+  *             points to the static TimeInfo structure.
+  *
+  * Return value:
+  * None.  This thread embeds an infinite loop.
+  *
+  * Side effects:
+  * At an interval of clockCalibrateWakeupInterval ms, this thread
+  * performs virtual time discipline.
+  *
+  * Note: When this thread is entered, TclpInitLock has been called
+  * to safeguard the static storage.  There is therefore no synchronization
+  * in the body of this procedure.
+  *
+  *----------------------------------------------------------------------
+  */
+
+ static DWORD WINAPI
+ CalibrationThread( LPVOID arg )
+ {
+     FILETIME curFileTime;
+
+     /* Get initial system time and performance counter */
+
+     GetSystemTimeAsFileTime( &curFileTime );
+     QueryPerformanceCounter( &timeInfo.lastCounter );
+     QueryPerformanceFrequency( &timeInfo.curCounterFreq );
+     timeInfo.lastFileTime.LowPart = curFileTime.dwLowDateTime;
+     timeInfo.lastFileTime.HighPart = curFileTime.dwHighDateTime;
+
+     /* Initialize the working storage for the calibration callback */
+
+     timeInfo.lastPerfCounter = timeInfo.lastCounter.QuadPart;
+     timeInfo.estPerfCounterFreq = timeInfo.curCounterFreq.QuadPart;
+
```

```
+     /*
+      * Wake up the calling thread.  When it wakes up, it will release the
+      * initialization lock.
+      */
+
+     SetEvent( timeInfo.readyEvent );
+
+     /* Run the calibration once a second */
+
+     for ( ; ; ) {
+
+  Sleep( 1000 );
+  UpdateTimeEachSecond();
+
+     }
+ }
+ ^L
+ /*
+  *----------------------------------------------------------------------
+  *
+  * UpdateTimeEachSecond --
+  *
+  * Callback from the waitable timer in the clock calibration thread
+  * that updates system time.
+  *
+  * Parameters:
+  * info -- Pointer to the static TimeInfo structure
+  *
+  * Results:
+  * None.
+  *
+  * Side effects:
+  * Performs virtual time calibration discipline.
+  *
+  *----------------------------------------------------------------------
+  */
+
+ static void
+ UpdateTimeEachSecond()
+ {
+
+     LARGE_INTEGER curPerfCounter;
+  /* Current value returned from
+   * QueryPerformanceCounter */
+
+     LONGLONG perfCounterDiff; /* Difference between the current value
+   * and the value of 1 second ago */
+
+     FILETIME curSysTime; /* Current system time */
+
+     LARGE_INTEGER curFileTime; /* File time at the time this callback
+   * was scheduled. */
+
+     LONGLONG fileTimeDiff; /* Elapsed time on the system clock
+   * since the last time this procedure
+   * was called */
+
+     LONGLONG instantFreq; /* Instantaneous estimate of the
+   * performance counter frequency */
+
+     LONGLONG delta; /* Increment to add to the estimated
+   * performance counter frequency in the
```

```
+   * loop filter */
+
+    LONGLONG fuzz; /* Tolerance for the perf counter frequency */
+
+    LONGLONG lowBound; /* Lower bound for the frequency assuming
+   * 1000 ppm tolerance */
+
+    LONGLONG hiBound; /* Upper bound for the frequency */
+
+    /*
+     * Get current performance counter and system time.
+     */
+
+    QueryPerformanceCounter( &curPerfCounter );
+    GetSystemTimeAsFileTime( &curSysTime );
+    curFileTime.LowPart = curSysTime.dwLowDateTime;
+    curFileTime.HighPart = curSysTime.dwHighDateTime;
+
+    EnterCriticalSection( &timeInfo.cs );
+
+    /*
+     * Find out how many ticks of the performance counter and the
+     * system clock have elapsed since we got into this procedure.
+     * Estimate the current frequency.
+     */
+
+    perfCounterDiff = curPerfCounter.QuadPart - timeInfo.lastPerfCounter;
+    timeInfo.lastPerfCounter = curPerfCounter.QuadPart;
+    fileTimeDiff = curFileTime.QuadPart - timeInfo.lastSysTime;
+    timeInfo.lastSysTime = curFileTime.QuadPart;
+    instantFreq = ( 10000000 * perfCounterDiff / fileTimeDiff );
+
+    /*
+     * Consider this a timing glitch if instant frequency varies
+     * significantly from the current estimate.
+     */
+
+    fuzz = timeInfo.estPerfCounterFreq >> 10;
+    lowBound = timeInfo.estPerfCounterFreq - fuzz;
+    hiBound = timeInfo.estPerfCounterFreq + fuzz;
+    if ( instantFreq < lowBound || instantFreq > hiBound ) {
+  LeaveCriticalSection( &timeInfo.cs );
+  return;
+    }
+
+    /*
+     * Update the current estimate of performance counter frequency.
+     * This code is equivalent to the loop filter of a phase locked
+     * loop.
+     */
+
+    delta = ( instantFreq - timeInfo.estPerfCounterFreq ) >> 6;
+    timeInfo.estPerfCounterFreq += delta;
+
+    /*
+     * Update the current virtual time.
+     */
+
+    timeInfo.lastFileTime.QuadPart
+  += ( ( curPerfCounter.QuadPart - timeInfo.lastCounter.QuadPart )
+       * 10000000 / timeInfo.curCounterFreq.QuadPart );
+    timeInfo.lastCounter.QuadPart = curPerfCounter.QuadPart;
```

```
+
+     delta = curFileTime.QuadPart - timeInfo.lastFileTime.QuadPart;
+     if ( delta > 10000000 || delta < -10000000 ) {
+
+  /*
+   * If the virtual time slip exceeds one second, then adjusting
+   * the counter frequency is hopeless (it'll take over fifteen
+   * minutes to line up with the system clock).  The most likely
+   * cause of this large a slip is a sudden change to the system
+   * clock, perhaps because it was being corrected by wristwatch
+   * and eyeball.  Accept the system time, and set the performance
+   * counter frequency to the current estimate.
+   */
+
+  timeInfo.lastFileTime.QuadPart = curFileTime.QuadPart;
+  timeInfo.curCounterFreq.QuadPart = timeInfo.estPerfCounterFreq;
+
+     } else {
+
+  /*
+   * Compute a counter frequency that will cause virtual time to line
+   * up with system time one second from now, assuming that the
+   * performance counter continues to tick at timeInfo.estPerfCounterFreq.
+   */
+
+  timeInfo.curCounterFreq.QuadPart
+      = 10000000 * timeInfo.estPerfCounterFreq / ( delta + 10000000 );
+
+  /*
+   * Limit frequency excursions to 1000 ppm from estimate
+   */
+
+  if ( timeInfo.curCounterFreq.QuadPart < lowBound ) {
+      timeInfo.curCounterFreq.QuadPart = lowBound;
+  } else if ( timeInfo.curCounterFreq.QuadPart > hiBound ) {
+      timeInfo.curCounterFreq.QuadPart = hiBound;
+  }
+     }
+
+     LeaveCriticalSection( &timeInfo.cs );
+
   }
*** ../tcl8.3.2base/src/tcl8.3.2/test/winTime.test Mon Apr 10 13:19:08 2000
--- ./tcl8.3.2/src/tcl8.3.2/test/winTime.test Wed Sep  6 14:55:30 2000
***************
*** 33,38 ****
--- 33,64 ----
      set result
  } {1969}

+ # Next test tries to make sure that the Tcl clock stays in step
+ # with the Windows clock.  3000 iterations really isn't enough,
+ # but how many does a tester have patience for?
+
+ test winTime-2.1 {Synchronization of Tcl and Windows clocks} {pcOnly} {
+     set failed 0
+     foreach { sys_sec sys_usec tcl_sec tcl_usec } [testwinclock] {}
+     set olddiff [expr { abs ( $tcl_sec - $sys_sec
+     + 1.0e-6 * ( $tcl_usec - $sys_usec ) ) }]
+     set ok 1
+     for { set i 0 } { $i < 3000 } { incr i } {
+  foreach { sys_sec sys_usec tcl_sec tcl_usec } [testwinclock] {}
```

```
+  set diff [expr { abs ( $tcl_sec - $sys_sec
+         + 1.0e-6 * ( $tcl_usec - $sys_usec ) ) }]
+  if { ( $diff > $olddiff + 1000 )
+       || ( $diff > 11000 ) } {
+      set failed 1
+      break
+  } else {
+      set olddiff $diff
+      after 1
+  }
+     }
+     set failed
+ } {0}
+
  # cleanup
  ::tcltest::cleanupTests
  return
```

TIP #8: Add Winico support to the wm command on windows

TIP #8:	**Add Winico support to the wm command on windows**
Author:	Vince Darley ⟨vince.darley@eurobios.com⟩
Created:	Monday, 6th November 2000
Type:	Project
Tcl Version:	8.4.0
State:	Final
Vote:	Done
Version:	`$Revision: 1.7 $`
Post-History:	

Abstract

Add to *wm* the ability to do the windows-titlebar-icon manipulation that the Winico extension currently provides, without the bugs noted in that extension.

8.1 Proposal

Modify *wm* on Windows only to allow an optional *-default* argument.

```
wm iconbitmap .winpath ?-default? filename
```

And to allow a file which is of valid windows-icon format to be interpreted as such. Any file which is not correctly interpreted as an icon will be handled as before, by the *bitmap* code (which will generally either do nothing, or throw an error, thus maintaining backwards compatibility).

The *-default* argument, if given, will change not the icon of the .winpath given, but rather the default icon for all windows in the current application for which no specific icon as been set.

An implementation already exists, which fixes the basic "wrapper window" problems and which has the above syntax. The issues surrounding reference counting of icons in use has also been addressed in this patch so that icons no longer in use are released (the Winico patch required manual deletion of icons). This reference implementation is available from ftp://ftp.ucsd.edu/pub/alpha/tcl/tkWinWm.diff (documentation has been separately patched, and can also be made available).

8.2 Rationale

There have been many requests on news:comp.lang.tcl for this ability in the Tk core, and several bug reports filed against Winico, and this ability has been placed on the Tk 8.4 roadmap. http://dev.scriptics.com/software/tcltk/roadmap.tml

The choice of *wm iconbitmap* is suggested, because *wm iconbitmap* currently doesn't appear to do anything on Windows, yet is the obvious choice for the user trying to set the window's icon (e.g. many posts on news:comp.lang.tcl are actually asking why *wm iconbitmap* doesn't do anything).

In the future we may wish to extend *wm iconbitmap* on all platforms so that other image types can be accepted (e.g. .gif, .png). This proposal extends naturally to allow such future work. The primary changes required will be icon<->image conversion routines.

8.3 Alternatives

Fix the core so that Winico can work properly as an extension.

My implementation as shown that this would require a couple of patches, and also the exporting of an additional obscure function into Tk's stub table (a function which would ensure that Tk's window manager is completely initialised). It would also not help the users posting to news:comp.lang.tcl asking "why doesn't wm iconbitmap do anything?"

8.4 Objections

This is platform specific and should go in an extension

See *Alternatives* above, also see the *future suggestion* above in which this kind of code can be usefully extended in a cross-platform way.

The -default flag is weird, and it means we ignore the window name

I agree, but please suggest a better alternative rather than just moaning. The command with the -default flag is in my opinion more useful than the command without (for example it makes sure that Tk's built-in dialogs have the icon of your application). An alternative might be to use *wm iconbitmap -default filename*, but that involves more significant modifications of the semantics of *wm*. It might, however, be a good idea.

wm iconbitmap will still do nothing when given a bitmap

Yes, but there's that backwards compatibility issue. This should be properly documented with pointers to the use of valid icon file formats. When or if proper support is added to Tk for .gif, .png or even Tk images as icons, this bug can be fixed. The purpose of this TIP is not to fix that bug, but to provide a better solution.

8.5 Copyright

TIP #9: Tk Standard Library

TIP #9:	**Tk Standard Library**
Author:	Marty Backe ⟨mgbacke@usa.net⟩
	hellins ⟨hellins@263.net⟩
	Larry W. Virden ⟨lvirden@yahoo.com⟩
Created:	Tuesday, 7th November 2000
Type:	Project
Tcl Version:	8.4
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.3 $`
Post-History:	

Abstract

A Tk standard library shall be bundled with the core Tcl/Tk distribution. The library will consist of general purpose widgets and composite widgets for use in constructing Tcl/Tk applications. The library of Tk components will be written in Tcl/Tk.

9.1 Rationale

Although Tcl "ships" with a comprehensive set of native (compiled) base Tk widgets, it lacks a library of composite widgets, from which sophisticated applications can readily be built with minimal reinvention.

Although the Tcl community has created a wealth of general purpose Tk widgets, generally they are not centrally located or distributed, making their use problematic. This requires that Tcl programs which make use of such widgets must either distribute them or direct the end user on their acquisition and installation. Arguably, the success and higher visibility of other "competing" scripting languages can be attributed in some part to their extensive libraries. Tcl/Tk should continue this trend.

Tcl is perhaps unique in that it is considered both a graphical (Tk) and non-graphical (Tcl) programming language. Work has begun in implementing a standard library for Tcl. It could be argued that Tcl/Tk's largest base, and its largest growth area, is with regards to graphical applications. To this end, Tcl needs a comprehensive, and well maintained Tk standard library.

Finally, to lower the barrier of using the Tk libraries, they should be Tcl/Tk based. This helps to assure cross platform independence without requiring the user to compile code against a source distribution.

9.2 Specification

- The standard Tk library will be called "tklibX.Y", where "X.Y" will follow the version number of the Tcl/Tk distribution that it's compatible with.
- Major/minor releases of the tklib shall coincide with the major/minor releases of Tcl/Tk. That is, if Tcl/Tk version 8.5 is released, a tklib8.5 shall be released. The tklib8.5 version shall be tested and confirmed to be compatible with the release of Tcl8.5 & Tk8.5. Note that changes to tklib will not necessarily be required for it to receive a new version number, but the new version shall indicate that it has been tested and verified compatible with the new Tcl/Tk version.
- The tklib shall be considered part of the "core" of Tcl/Tk. That is, releases of major/minor versions of Tcl/Tk shall not be made independent of tklib.
- Additions to the tklib shall be made through a voting process, which is to be decided.
- Tklib components shall include a test suite. This test suite will be the means by which the library is verified as compatible with a new release of Tcl/Tk.
- Tklib components shall include documentation to the same standards as Tcl/Tk, i.e., man pages, etc. Let's continue the tradition of Tcl/Tk having the best documentation.
- The tklib components will include one or more demonstration scripts that show to best effect all of the features and options provided by the component. A picture is worth a thousand words! The Tk, BWidgets, and Iwidgets demos are prime examples to be emulated.
- Tklib components can be dependent on other tklib components. If tklib and tcllib become coordinated efforts, the tklib components can be dependent on tcllib components.
- The tklib can (and hopefully will) include megawidgets.
- Tklib components shall be written in Tcl/Tk.
- Tklib components shall be implemented in their own namespace and distributed in package form.
- Tklib components do not have to be unique with regards to other tklib components, although there shall be differentiating characteristics between them. There is more then one way to skin a cat.
- The tklib shall not contain applications, IDEs, or development tools.

9.3 Copyright

TIP #10: Tcl I/O Enhancement: Thread-Aware Channels

TIP #10: **Tcl I/O Enhancement: Thread-Aware Channels**	
Author:	Andreas Kupries ⟨a.kupries@westend.com⟩
Created:	Wednesday, 8th November 2000
Type:	Project
Tcl Version:	8.4
State:	Final
Vote:	Done
Version:	`$Revision: 1.6 $`
Post-History:	

Abstract

This TIP describes how to change the generic I/O layer in the Tcl core to make channels aware of the thread they are managed by.

10.1 Rationale

To explain the motives behind this TIP first a short look at the history of channels and threading.

In ancient times the Tcl core was not thread safe and did not employ threads. All channels belonged to a single interpreter. Later on interpreter hierarchies were introduced and the ability to move or share a channel between the interpreters in a hierarchy. When the Tcl core was made thread safe a short time after the ability to move channels between threads was added (Helper APIs in the core, main functionality in the Thread extension). The goal behind these modifications was to enable the creation of stream-like communication paths between threads to complement the message based facilities (thread send). The modifications were only a partial success because an in-depth analysis of the relevant data structures showed that the sharing of a channel between threads is not possible with the current design, only moving. This was implemented to allow at least the dispatcher- / worker-thread pattern for structuring a threaded application.

In further pursuit of the original goal the currently chosen approach is to define a channeltype where two channels are connected internally through in-memory fifo buffers where access to these shared structures is protected by mutexes.

During the implementation of fileevents for this channeltype it was discovered that an efficient implementation of this part is *not* possible because of the inability to post file events to the eventqueue of the thread the other channel of the pair resides in. An API to post such events is available (*Tcl_ThreadQueueEvent*), but not the information which thread actually manages the other channel. Because of this the current implementation of the channeltype uses polling based upon timer events posted by each side/thread to itself to manage file events in a rather inefficient way.

10.2 Reference implementation

This TIP now proposes to change the internals of the generic I/O layers in the core so that

1. Channels know the thread they are managed by, and
2. are able to deliver this information to an extension querying the core.

This then allows the two sides of the channeltype mentioned above to post events to each other, facilitating an efficient implementation of fileevents.

The changes necessary to accomplish this are:

1. Extend the structure *ChannelState* in tclIO.h with a new field of type *Tcl_ThreadId* to hold the id of the thread currently managing all channels with this state. Note: This structure is shared by all channels in a stack of transformations.
2. Modify the procedure *Tcl_CreateChannel* to store the Id of the current thread in the *ChannelState* of the new channel. This information can be obtained with *Tcl_GetCurrentThread*. It is *not* necessary to modify *Tcl_StackChannel* as the thread information is already part of the state when it is called, and won't be changed by the call.
3. If some sort of NIL/NULL value meaning "No thread" is available for *Tcl_ThreadId*, then we should modify *Tcl_CutChannel* to insert this value into the state of the channel it is called with, as this channel will not be managed by any thread afterward (the procedure removes the channel from the list of all channels managed by the current thread).
4. Modify *Tcl_SpliceChannel* in the same manner as *Tcl_CreateChannel* as the channel will be managed by the current thread afterward (The procedure adds the channel to the list of all channels managed by the current thread).
5. Declare a new API function to retrieve the Id of the managing thread from a channel. Add this declaration to generic/tcl.decls and implement the function in the file generic/tclIO.c. I propose *Tcl_GetChannelThread* as the name of this new API function.

A patch implementing all of the changes described above and additionally extending the documentation and the test-suite is available here: http://www.cs.man.ac.uk/fellowsd-bin/TIP/10.patch

10.3 Copyright

This document has been placed in the public domain.

TIP #11: Tk Menubutton Enhancement: -compound option for menubutton

TIP #11: **Tk Menubutton Enhancement: -compound option for menubutton**	
Author:	Todd Helfter ⟨tmh@purdue.edu⟩
Created:	Thursday, 16th November 2000
Type:	Project
Tcl Version:	8.4
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.4 $`
Post-History:	

Abstract

This TIP describes how to change the menubutton in the Tk core to add a -compound option to display both text and images. This behavior already exists in the button widget.

11.1 Rationale

In order to have a menubutton with both text and images, this change is needed. This change facilitates the use of an image for the menubutton face with text on top. Like the button widget, the -compound option will accept these values: none, center, left, right, top, bottom.

11.2 Reference Implementation

This TIP proposes to change the internals of the menubutton.

The changes necessary to accomplish this are:

1. Extend the structure *TkMenuButton* in *generic/tkMenubutton.h* with a new field of type *int* to hold the value of the compound setting.
2. Add an enumeration of valid -compound options in *generic/tkMenubutton.h*.
3. Modify *generic/tkMenuButton.c* and *unix/tkUnixMenubu.c* in such a way to process this new option. Note: The windows port of Tk uses the *unix/tkUnixMenubu.c* file. So this change is portable to both Unix and windows.
4. Change *tests/menubut.test* so that the test for configure options checks for 33 instead of the current 32.
5. Change *doc/menubutton.n* to show the new option under widget specific options.

11.3 Copyright

This document has been placed in the public domain.

11.4 Patch

```
Index: doc/menubutton.n
===================================================================
RCS file: /cvsroot/tk/doc/menubutton.n,v
retrieving revision 1.3
diff -c -r1.3 menubutton.n
*** menubutton.n 2000/08/25 06:58:32 1.3
--- menubutton.n 2000/11/16 14:37:15
***************
*** 26,31 ****
--- 26,39 ----
  \-disabledforeground \-padx
  .SE
  .SH "WIDGET-SPECIFIC OPTIONS"
+ .OP \-compound compound Compound
+ Specifies whether the menubutton should display both an image and text,
+ and if so, where the image should be placed relative to the text.
+ Valid values for this option are \fBbottom\fR, \fBcenter\fR,
+ \fBleft\fR, \fBnone\fR, \fBright\fR and \fBtop\fR.  The default value
+ is \fBnone\fR, meaning that the menubutton will display either an image or
+ text, depending on the values of the \fB\-image\fR and \fB\-bitmap\fR
+ options.
  .VS
  .OP \-direction direction Height
  Specifies where the menu is going to be popup up. \fBabove\fR tries to
Index: generic/tkMenubutton.c
===================================================================
```

```
RCS file: /cvsroot/tk/generic/tkMenubutton.c,v
retrieving revision 1.4
diff -c -r1.4 tkMenubutton.c
*** tkMenubutton.c 1999/04/24 01:50:49 1.4
--- tkMenubutton.c 2000/11/16 14:37:16
***************
*** 37,42 ****
--- 37,51 ----
  };

  /*
+  * The following table defines the legal values for the -compound option.
+  * It is used with the "enum compound" declaration in tkButton.h
+  */
+
+ static char *compoundStrings[] = {
+     "bottom", "center", "left", "none", "right", "top", (char *) NULL
+ };
+
+ /*
   * Information used for parsing configuration specs:
   */

***************
*** 113,118 ****
--- 122,130 ----
      {TK_OPTION_RELIEF, "-relief", "relief", "Relief",
 DEF_MENUBUTTON_RELIEF, -1, Tk_Offset(TkMenuButton, relief),
   0, 0, 0},
+     {TK_OPTION_STRING_TABLE, "-compound", "compound", "Compound",
+          DEF_BUTTON_COMPOUND, -1, Tk_Offset(TkMenuButton, compound), 0,
+          (ClientData) compoundStrings, 0},
      {TK_OPTION_STRING_TABLE, "-state", "state", "State",
 DEF_MENUBUTTON_STATE, -1, Tk_Offset(TkMenuButton, state),
 0, (ClientData) stateStrings, 0},
Index: generic/tkMenubutton.h
===================================================================
RCS file: /cvsroot/tk/generic/tkMenubutton.h,v
retrieving revision 1.5
diff -c -r1.5 tkMenubutton.h
*** tkMenubutton.h 1999/04/16 01:51:19 1.5
--- tkMenubutton.h 2000/11/16 14:37:16
***************
*** 25,30 ****
--- 25,39 ----
  #endif

  /*
+  * Legal values for the "compound" field of TkButton records.
+  */
+
+ enum compound {
+     COMPOUND_BOTTOM, COMPOUND_CENTER, COMPOUND_LEFT, COMPOUND_NONE,
+         COMPOUND_RIGHT, COMPOUND_TOP
+ };
+
+ /*
   * Legal values for the "orient" field of TkMenubutton records.
   */

***************
*** 161,166 ****
```

```
--- 170,179 ----
      /*
* Miscellaneous information:
*/
+
+     int compound;               /* Value of -compound option; specifies whether
+                                  * the button should show both an image and
+                                  * text, and, if so, how. */

      enum direction direction; /* Direction for where to pop the menu.
  * Valid directions are "above", "below",
Index: tests/menubut.test
===================================================================
RCS file: /cvsroot/tk/tests/menubut.test,v
retrieving revision 1.5
diff -c -r1.5 menubut.test
*** menubut.test 1999/04/21 21:53:29 1.5
--- menubut.test 2000/11/16 14:37:18
***************
*** 138,144 ****
  } {3}
  test menubutton-3.7 {ButtonWidgetCmd procedure, "configure" option} {
      llength [.mb configure]
! } {32}
  test menubutton-3.8 {ButtonWidgetCmd procedure, "configure" option} {
      list [catch {.mb configure -gorp} msg] $msg
  } {1 {unknown option "-gorp"}}
--- 138,144 ----
  } {3}
  test menubutton-3.7 {ButtonWidgetCmd procedure, "configure" option} {
      llength [.mb configure]
! } {33}
  test menubutton-3.8 {ButtonWidgetCmd procedure, "configure" option} {
      list [catch {.mb configure -gorp} msg] $msg
  } {1 {unknown option "-gorp"}}
Index: unix/tkUnixMenubu.c
===================================================================
RCS file: /cvsroot/tk/unix/tkUnixMenubu.c,v
retrieving revision 1.4
diff -c -r1.4 tkUnixMenubu.c
*** tkUnixMenubu.c 1999/09/21 06:43:01 1.4
--- tkUnixMenubu.c 2000/11/16 14:37:18
***************
*** 75,83 ****
      Pixmap pixmap;
      int x = 0; /* Initialization needed only to stop
  * compiler warning. */
!     int y;
      register Tk_Window tkwin = mbPtr->tkwin;
!     int width, height;

      mbPtr->flags &= ~REDRAW_PENDING;
      if ((mbPtr->tkwin == NULL) || !Tk_IsMapped(tkwin)) {
--- 75,85 ----
      Pixmap pixmap;
      int x = 0; /* Initialization needed only to stop
  * compiler warning. */
!     int y = 0;
      register Tk_Window tkwin = mbPtr->tkwin;
!     int width, height, fullWidth, fullHeight;
!     int imageXOffset, imageYOffset, textXOffset, textYOffset;
!     int haveImage = 0, haveText = 0;
```

```
      mbPtr->flags &= ~REDRAW_PENDING;
      if ((mbPtr->tkwin == NULL) || !Tk_IsMapped(tkwin)) {
***************
*** 96,101 ****
--- 98,112 ----
 border = mbPtr->normalBorder;
      }

+     if (mbPtr->image != None) {
+  Tk_SizeOfImage(mbPtr->image, &width, &height);
+  haveImage = 1;
+     } else if (mbPtr->bitmap != None) {
+  Tk_SizeOfBitmap(mbPtr->display, mbPtr->bitmap, &width, &height);
+  haveImage = 1;
+     }
+     haveText = (mbPtr->textWidth != 0 && mbPtr->textHeight != 0);
+
      /*
* In order to avoid screen flashes, this procedure redraws
* the menu button in a pixmap, then copies the pixmap to the
***************
*** 107,141 ****
     Tk_Width(tkwin), Tk_Height(tkwin), Tk_Depth(tkwin));
      Tk_Fill3DRectangle(tkwin, pixmap, border, 0, 0, Tk_Width(tkwin),
     Tk_Height(tkwin), 0, TK_RELIEF_FLAT);
-
-     /*
-      * Display image or bitmap or text for button.
-      */

!     if (mbPtr->image != None) {
!  Tk_SizeOfImage(mbPtr->image, &width, &height);
!
!  imageOrBitmap:
!  TkComputeAnchor(mbPtr->anchor, tkwin, 0, 0,
!  width + mbPtr->indicatorWidth, height, &x, &y);
!  if (mbPtr->image != NULL) {
!      Tk_RedrawImage(mbPtr->image, 0, 0, width, height, pixmap,
!      x, y);
!  } else {
!      XCopyPlane(mbPtr->display, mbPtr->bitmap, pixmap,
!      gc, 0, 0, (unsigned) width, (unsigned) height, x, y, 1);
!  }
!     } else if (mbPtr->bitmap != None) {
!  Tk_SizeOfBitmap(mbPtr->display, mbPtr->bitmap, &width, &height);
!  goto imageOrBitmap;
      } else {
!  TkComputeAnchor(mbPtr->anchor, tkwin, mbPtr->padX, mbPtr->padY,
!  mbPtr->textWidth + mbPtr->indicatorWidth,
!  mbPtr->textHeight, &x, &y);
!  Tk_DrawTextLayout(mbPtr->display, pixmap, gc, mbPtr->textLayout, x, y,
!  0, -1);
!  Tk_UnderlineTextLayout(mbPtr->display, pixmap, gc, mbPtr->textLayout,
!  x, y, mbPtr->underline);
      }

      /*
--- 118,223 ----
     Tk_Width(tkwin), Tk_Height(tkwin), Tk_Depth(tkwin));
      Tk_Fill3DRectangle(tkwin, pixmap, border, 0, 0, Tk_Width(tkwin),
     Tk_Height(tkwin), 0, TK_RELIEF_FLAT);
```

```
!     imageXOffset = 0;
!     imageYOffset = 0;
!     textXOffset = 0;
!     textYOffset = 0;
!     fullWidth = 0;
!     fullHeight = 0;
!
!     if (mbPtr->compound != COMPOUND_NONE && haveImage && haveText) {
!
!         switch ((enum compound) mbPtr->compound) {
!             case COMPOUND_TOP:
!             case COMPOUND_BOTTOM: {
!                 /* Image is above or below text */
!                 if (mbPtr->compound == COMPOUND_TOP) {
!                     textYOffset = height + mbPtr->padY;
!                 } else {
!                     imageYOffset = mbPtr->textHeight + mbPtr->padY;
!                 }
!                 fullHeight = height + mbPtr->textHeight + mbPtr->padY;
!                 fullWidth = (width > mbPtr->textWidth ? width :
!                         mbPtr->textWidth);
!                 textXOffset = (fullWidth - mbPtr->textWidth)/2;
!                 imageXOffset = (fullWidth - width)/2;
!                 break;
!             }
!             case COMPOUND_LEFT:
!             case COMPOUND_RIGHT: {
!                 /* Image is left or right of text */
!                 if (mbPtr->compound == COMPOUND_LEFT) {
!                     textXOffset = width + mbPtr->padX;
!                 } else {
!                     imageXOffset = mbPtr->textWidth + mbPtr->padX;
!                 }
!                 fullWidth = mbPtr->textWidth + mbPtr->padX + width;
!                 fullHeight = (height > mbPtr->textHeight ? height :
!                         mbPtr->textHeight);
!                 textYOffset = (fullHeight - mbPtr->textHeight)/2;
!                 imageYOffset = (fullHeight - height)/2;
!                 break;
!             }
!             case COMPOUND_CENTER: {
!                 /* Image and text are superimposed */
!                 fullWidth = (width > mbPtr->textWidth ? width :
!                         mbPtr->textWidth);
!                 fullHeight = (height > mbPtr->textHeight ? height :
!                         mbPtr->textHeight);
!                 textXOffset = (fullWidth - mbPtr->textWidth)/2;
!                 imageXOffset = (fullWidth - width)/2;
!                 textYOffset = (fullHeight - mbPtr->textHeight)/2;
!                 imageYOffset = (fullHeight - height)/2;
!                 break;
!             }
!             case COMPOUND_NONE: {break;}
!         }
!
!         TkComputeAnchor(mbPtr->anchor, tkwin, 0, 0,
!                 mbPtr->indicatorWidth + fullWidth, fullHeight,
!  &x, &y);
!
!         if (mbPtr->image != NULL) {
!             Tk_RedrawImage(mbPtr->image, 0, 0, width, height, pixmap,
```

```
!                     x + imageXOffset, y + imageYOffset);
!         }
!         if (mbPtr->bitmap != None) {
!             XCopyPlane(mbPtr->display, mbPtr->bitmap, pixmap,
!                     gc, 0, 0, (unsigned) width, (unsigned) height,
!      x + imageXOffset, y + imageYOffset, 1);
!         }
!         if (haveText) {
!             Tk_DrawTextLayout(mbPtr->display, pixmap, gc, mbPtr->textLayout,
!      x  + textXOffset, y + textYOffset ,
!                     0, -1);
!             Tk_UnderlineTextLayout(mbPtr->display, pixmap, gc,
!      mbPtr->textLayout, x + textXOffset, y + textYOffset ,
!      mbPtr->underline);
!         }
      } else {
!        if (mbPtr->image != NULL) {
!            TkComputeAnchor(mbPtr->anchor, tkwin, 0, 0,
!                    width + mbPtr->indicatorWidth, height, &x, &y);
!            Tk_RedrawImage(mbPtr->image, 0, 0, width, height, pixmap,
!                    x + imageXOffset, y + imageYOffset);
!        } else if (mbPtr->bitmap != None) {
!            TkComputeAnchor(mbPtr->anchor, tkwin, 0, 0,
!                    width + mbPtr->indicatorWidth, height, &x, &y);
!            XCopyPlane(mbPtr->display, mbPtr->bitmap, pixmap,
!                    gc, 0, 0, (unsigned) width, (unsigned) height,
!     x + imageXOffset, y + imageYOffset, 1);
!        } else {
!            TkComputeAnchor(mbPtr->anchor, tkwin, mbPtr->padX, mbPtr->padY,
!                    mbPtr->textWidth + mbPtr->indicatorWidth,
!                    mbPtr->textHeight, &x, &y);
!            Tk_DrawTextLayout(mbPtr->display, pixmap, gc, mbPtr->textLayout,
!     x  + textXOffset, y + textYOffset ,
!                    0, -1);
!            Tk_UnderlineTextLayout(mbPtr->display, pixmap, gc,
!     mbPtr->textLayout, x + textXOffset, y + textYOffset ,
!     mbPtr->underline);
!         }
      }

      /*
***************
*** 252,305 ****
      TkMenuButton *mbPtr; /* Widget record for menu button. */
  {
      int width, height, mm, pixels;

      mbPtr->inset = mbPtr->highlightWidth + mbPtr->borderWidth;
      if (mbPtr->image != None) {
 Tk_SizeOfImage(mbPtr->image, &width, &height);
!  if (mbPtr->width > 0) {
!      width = mbPtr->width;
!  }
!  if (mbPtr->height > 0) {
!      height = mbPtr->height;
!  }
      } else if (mbPtr->bitmap != None) {
 Tk_SizeOfBitmap(mbPtr->display, mbPtr->bitmap, &width, &height);
!  if (mbPtr->width > 0) {
!      width = mbPtr->width;
!  }
!  if (mbPtr->height > 0) {
```

```
!      height = mbPtr->height;
!  }
!     } else {
 Tk_FreeTextLayout(mbPtr->textLayout);
 mbPtr->textLayout = Tk_ComputeTextLayout(mbPtr->tkfont, mbPtr->text,
 -1, mbPtr->wrapLength, mbPtr->justify, 0, &mbPtr->textWidth,
 &mbPtr->textHeight);
!  width = mbPtr->textWidth;
!  height = mbPtr->textHeight;
!  if (mbPtr->width > 0) {
!      width = mbPtr->width * Tk_TextWidth(mbPtr->tkfont, "0", 1);
!  }
!  if (mbPtr->height > 0) {
!      Tk_FontMetrics fm;

!      Tk_GetFontMetrics(mbPtr->tkfont, &fm);
!      height = mbPtr->height * fm.linespace;
 }
!  width += 2*mbPtr->padX;
!  height += 2*mbPtr->padY;
      }

      if (mbPtr->indicatorOn) {
!  mm = WidthMMOfScreen(Tk_Screen(mbPtr->tkwin));
!  pixels = WidthOfScreen(Tk_Screen(mbPtr->tkwin));
!  mbPtr->indicatorHeight= (INDICATOR_HEIGHT * pixels)/(10*mm);
!  mbPtr->indicatorWidth = (INDICATOR_WIDTH * pixels)/(10*mm)
!  + 2*mbPtr->indicatorHeight;
!  width += mbPtr->indicatorWidth;
      } else {
!  mbPtr->indicatorHeight = 0;
!  mbPtr->indicatorWidth = 0;
      }

      Tk_GeometryRequest(mbPtr->tkwin, (int) (width + 2*mbPtr->inset),
--- 334,446 ----
      TkMenuButton *mbPtr; /* Widget record for menu button. */
  {
      int width, height, mm, pixels;
+     int  avgWidth, txtWidth, txtHeight;
+     int haveImage = 0, haveText = 0;
+     Tk_FontMetrics fm;

      mbPtr->inset = mbPtr->highlightWidth + mbPtr->borderWidth;
+
+     width = 0;
+     height = 0;
+     txtWidth = 0;
+     txtHeight = 0;
+     avgWidth = 0;
+
      if (mbPtr->image != None) {
 Tk_SizeOfImage(mbPtr->image, &width, &height);
!  haveImage = 1;
      } else if (mbPtr->bitmap != None) {
 Tk_SizeOfBitmap(mbPtr->display, mbPtr->bitmap, &width, &height);
!  haveImage = 1;
!     }
!
!     if (haveImage == 0 || mbPtr->compound != COMPOUND_NONE) {
 Tk_FreeTextLayout(mbPtr->textLayout);
+
```

```
 mbPtr->textLayout = Tk_ComputeTextLayout(mbPtr->tkfont, mbPtr->text,
 -1, mbPtr->wrapLength, mbPtr->justify, 0, &mbPtr->textWidth,
 &mbPtr->textHeight);
!  txtWidth = mbPtr->textWidth;
!  txtHeight = mbPtr->textHeight;
!         avgWidth = Tk_TextWidth(mbPtr->tkfont, "0", 1);
!         Tk_GetFontMetrics(mbPtr->tkfont, &fm);
!         haveText = (txtWidth != 0 && txtHeight != 0);
!     }
!
!     /*
!      * If the menubutton is compound (ie, it shows both an image and text),
!      * the new geometry is a combination of the image and text geometry.
!      * We only honor the compound bit if the menubutton has both text and
!      * an image, because otherwise it is not really a compound menubutton.
!      */

!     if (mbPtr->compound != COMPOUND_NONE && haveImage && haveText) {
!         switch ((enum compound) mbPtr->compound) {
!             case COMPOUND_TOP:
!             case COMPOUND_BOTTOM: {
!                 /* Image is above or below text */
!                 height += txtHeight + mbPtr->padY;
!                 width = (width > txtWidth ? width : txtWidth);
!                 break;
!             }
!             case COMPOUND_LEFT:
!             case COMPOUND_RIGHT: {
!                 /* Image is left or right of text */
!                 width += txtWidth + mbPtr->padX;
!                 height = (height > txtHeight ? height : txtHeight);
!                 break;
!             }
!             case COMPOUND_CENTER: {
!                 /* Image and text are superimposed */
!                 width = (width > txtWidth ? width : txtWidth);
!                 height = (height > txtHeight ? height : txtHeight);
!                 break;
!             }
!             case COMPOUND_NONE: {break;}
!         }
!         if (mbPtr->width > 0) {
!             width = mbPtr->width;
!         }
!         if (mbPtr->height > 0) {
!             height = mbPtr->height;
!         }
!         width += 2*mbPtr->padX;
!         height += 2*mbPtr->padY;
!     } else {
!  if (haveImage) {
!             if (mbPtr->width > 0) {
!                 width = mbPtr->width;
!             }
!             if (mbPtr->height > 0) {
!                 height = mbPtr->height;
!             }
!  } else {
!      width = txtWidth;
!      height = txtHeight;
!             if (mbPtr->width > 0) {
!                 width = mbPtr->width * avgWidth;
```

```
!             }
!             if (mbPtr->height > 0) {
!                 height = mbPtr->height * fm.linespace;
!             }
 }
!     }
!
!     if (! haveImage) {
!         width += 2*mbPtr->padX;
!         height += 2*mbPtr->padY;
      }

      if (mbPtr->indicatorOn) {
!         mm = WidthMMOfScreen(Tk_Screen(mbPtr->tkwin));
!         pixels = WidthOfScreen(Tk_Screen(mbPtr->tkwin));
!         mbPtr->indicatorHeight= (INDICATOR_HEIGHT * pixels)/(10*mm);
!         mbPtr->indicatorWidth = (INDICATOR_WIDTH * pixels)/(10*mm)
!          + 2*mbPtr->indicatorHeight;
!         width += mbPtr->indicatorWidth;
      } else {
!         mbPtr->indicatorHeight = 0;
!         mbPtr->indicatorWidth = 0;
      }

      Tk_GeometryRequest(mbPtr->tkwin, (int) (width + 2*mbPtr->inset),
```

TIP #12: The ”Batteries Included” Distribution

TIP #12: **The ”Batteries Included” Distribution**	
Author:	George A. Howlett ⟨gah@siliconmetrics.com⟩
	Larry W. Virden ⟨lvirden@yahoo.com⟩
Created:	Friday, 15th September 2000
Type:	Informative
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.3 $`
Post-History:	
Discussions-To:	news:comp.lang.tcl

Abstract

This document describes a comprehensive Tcl/Tk distribution. Its primary purpose is to create a standard source tree that includes Tcl, Tk, and extensions so that they can be built and installed in an simple and easy manner.

12.1 Introduction

One of the most enduring complaints about Tcl/Tk is that it lacks features, especially when compared to Perl, Python, or Java. We patiently explain that some particular feature is available in extension "XYZ" only to hear how hard it is to build and install extensions.

Frank Stajano ("The SMS server, or why I switched from Tcl to Python") describes the problem succinctly.

> "But if I had to put the finger on the single most important reason that has me now working in Python rather than in Tcl/[incr Tcl] it would not be a language issue but a library issue. I prefer Python because its standard library is a gold mine. Sure, for anything I want to do there's bound to be an extension available in the Tcl code repository on the FTP site. Now I just have to find it, fetch it, recompile the interpreter with it (Oh wait — this may mean getting and installing a C compiler for this system. Will the GNU one compile the windowing stuff properly or do I need to get VC++, or Borland? Who wants to have some fun discovering where another IDE has hidden the useful compiler flags this week?), hope that it won't clash with other extensions I've had to install, hope that it will not require a different version of the interpreter from the one I am running, and so on. Python supports the same C extension mechanism as Tcl — but the practical difference is that the stuff I want is, most of the time, already included and shipped in the standard distribution of the language!"

> "But, as a general-purpose tool, Python's single most important selling point is the richness of its standard library — an idea that Tcl is only now starting to internalise. It's all in the distribution. You can attack your practical problem using the stuff that's already installed on your system, and documented in the library manual you already printed. Python is great because it comes with batteries included."

It's true. There are too many things to know to maintain even a moderate set of extensions. There are too many different places to download extensions, too many extension-specific configuration options, etc.

My hope is that this proposal will mark the beginning of the end of the "Batteries Included" problem. One evidence of success will be that words "core" and "extension" disappear from our Tcl vocabularies. We've lived their artifical distinctions that are useful only to core developers and extension writers. It's skewed our thinking about relationship between Tcl and its parts. After all, application writers first care about whether a feature or capability is available, not how it's structured under the hood.

12.2 The "Batteries Included" Distribution.

Let's start with a very modest example. Let's imagine that the "Batteries Included" distribution is nothing more than an archive file of the source code for Tcl/Tk and several extensions.

	Unix	Windows	Mac
Tcl 8.3	x	x	x
Tk 8.3	x	x	x
[incr Tcl]	x	x	x
expect	x	?	
TclX	x		
BLT	x	x	
Trf			
Html widget			
XML			
...lots more...			

Tcl, Tk, and the packages are configured such that they can be built and installed just from a top level directory (not individually). Someone can download and try out all sorts of new features without repeating the same "configure", "make", "make install" sequences.

With this simple tar file, the following benefits are automatically generated:

- It provides a simple way for users to try out extensions. Users only have to run download, configure, compile and install, at most, once.
- It describes a clear framework for extensions. We will have established a directory structure for both source code and installed binaries. It will be much more clear how to inter-operate. This is TEA in action.
- It's better for Tcl/Tk application writers. You can count on features being universally available. Your program can again be just a Tcl script, not an array of packages that everyone needs to download and install.
- It's better for extension writers. Configuration is simpler, since you know where all the sources and the compiler-specific information will reside. You don't need to search for *tclConfig.sh* or *tkConfig.sh* files.
- It's better for Tcl/Tk distribution builders. This includes both the Linux distributors and company sysadmins that build Tcl/Tk. They don't have to fear installing extensions because of version dependencies.

  Let's give Redhat and SuSE a good reason to move off of version 8.0. One the big advantages of Linux over (let's say) Solaris is that each new Redhat or SuSE distribution comes with updated versions of utilities already built.
- It's better for the core developers. Extension writers will willing the adopt changes in exchange for the wider distribution. The core team will in turn gain better understanding of the burdens of extension writers.
- It's better for Tcl library writers. With [incr Tcl], we now have a basis for a real, extensible Tcl-code library. Library code rely on a full set of extensions being available.

12.3 Rationale

We want to create an open door procedure that makes it easy for contributors to add new features and commands to Tcl and Tk. By creating a framework for extensions to be built and distributed, the "Batteries Included" distribution will provide a path for great new features to quickly become available to the Tcl community.

The "Batteries Included" distributed is not designed to be one size that fits all. I assume there will be many distributions to suit many needs. There may be one for Tcl web servers and another for embedded systems. The goal is that the "Batteries Included" distribution will become a prototype for other distributions. Distribution creators will be able to pull code from the same CVS source tree.

What will distinguish the "Batteries Included" distribution is that it will be the most comprehensive and most up-to-date distribution. We will explicitly not choose one package or extension over another. That decision should remain with the Tcl user community. The only requirement is that the extensions are robust and/or actively maintained.

If the "Batteries Included" distribution is to become successful, it must be a cooperative effort between Tcl core developers, extension writers, and the Tcl user community. For example, we need the help of extension writers to adopt the new configuration scheme and directory structure.

12.4 Particulars

We can stage the project with small milestones while still focusing on longer range goals. For example, the first phase can be as simple as creating a tar file. It will start to address questions that were raised by TEA. For example, how do we manage documentation?

The biggest reason why this proposal will succeed is the incredible talent in the Tcl community. We can leverage the skills and experiences of the foremost experts on the core, extensions, and applications.

12.5 Tcl/Tk Version.

The distribution will be based on 8.3.2 (or 8.3.3 when it is released). While there's no assurance when 8.4 will be released and in what state, we also want to place a premium on stable, robust extensions, that have been thoroughly tested. Most extensions will be unlikely to have been tested against the 8.4 alphas.

12.6 Phase 1.

- Identify extensions.

  What extensions should be included in the near term? We need extension authors that are willing to work with us to build a directory framework, change configuration files, etc. Extensions do not need to work on all platforms. For example, there is a wealth of Windows-based extensions that should be included in a Windows specific build.

  What are the minimum requirements for extensions in the short term? Manual pages, html, tests, demos all would be nice. We need to temper this with what's practical. This is a learning process. We can adjust requirements in future phases.

- Determine build and install directory structures.

  We need to make this work with more that one release installed. Don't suppose that there only one version will ever be used.

- Setup CVS archives.

- Create configuration files.

  This will require negotiation with extension writers. We want their buy-in so they will maintain the changes.

  There may be more than one form of configuration required. One subtle but important issue is that extensions must be able to be configured without Tcl or Tk libraries already existing. This is a "trusted" configure. The extension must trust that the library will exist. Right now, most extensions work from "untrusted" configurations.

- Test builds on multiple platforms.

  For now, the Windows and Mac build files can be hand-generated. It may be too hard to create a seamless build environment. We're not trying to satisfy every Windows/Mac developer here. We can focus on creating pre-built binary distributions for these platforms.

- Create self-installing executables for Windows and the Mac.

  If we want, we can provide Linux, Solaris, etc. binaries by reviving Michael McLennan's Tclish installer.

12.7 Phase 2.

- Handle documentation issues.

  Generate platform specific doc with Richard Hipp's XML code.

- Establish Tcl code library.

- Identify more extensions.

- Determine the release schedule for "batteries included" distribution.

  How often do you release a new version? It must be more frequent than Tcl/Tk. We can start by planning for quarterly releases and then adding more frequent releases if necessary.

- Determine what core changes (if any) are needed for the distribution.

- Start looking at network-based updates.

- Start looking at selective builds. Allow builders to compile/install subsets of the distribution.

- Push on Redhat, SuSE, etc. to pick up distribution.

12.8 Phase 3.

- Network-based installs.
- Selective installations/builds.
- Include applications tree.
- Identify more extensions.

The last phases are sketchy. Feel free to add to this list, further breaking down goals into subtasks.

12.9 Open Issues

- Windows and MacIntosh sources.

  Given the dearth of configuration tools for these platforms, it's likely that only binary installations will be available for the near term.

- Documentation

  Overlap in command and widget names can be neatly handled by namespaces. Need to consider how to handle manual pages.

12.10 More Information

If anyone has interest to participate or would like to add comments to the "Batteries Included" proposal, please send mail to George Howlett ⟨gah@siliconmetrics.com⟩.

12.11 Copyright

12.12 See Also

[TIP #4] by Brent Welch ⟨welch@acm.org⟩.

TIP #13: Web Service for Drafting and Archiving TIPs

TIP #13:	**Web Service for Drafting and Archiving TIPs**
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Tuesday, 21st November 2000
Type:	Process
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.24 $`
Post-History:	

Abstract

This document proposes the TCT provide a service on the World Wide Web for drafting and archiving TIPs and for providing TIPs in a variety of formats. A reference implementation is provided, and its server requirements are outlined.

13.1 Background

It has been proposed (see [TIP #2]) that the TCT manage its projects and procedures through a set of public documents known as Tcl Improvement Proposals, or TIPs. A format for TIPs has been approved (see [TIP #3]), and although final approval of [TIP #2] is still pending, several TIPs have been submitted, discussed, and revised, and a few have been approved, so acceptance of TIPs in some form seems likely.

A prototype system has been provided by Donal Fellows ⟨fellowsd@cs.man.ac.uk⟩ at http://www.cs.man.ac.uk/fellowsd-bin/TIP/ that delivers TIPs to visitors in a variety of formats. However, that system lacks archiving of each revision of each TIP, and offers no interface (through the web or otherwise) for making revisions to TIPs.

The TIP format was inspired by the format used by the Tcl'ers Wiki (http://www.purl.org/thecliff/tcl/wiki/). The true power of the Tcl'ers Wiki, though, is not in the particular format it uses, but in the fact that it empowers the whole Tcl community to contribute to a common set of resources. The Tcl'ers Wiki shows that valuable resources can arise out of the unrestricted efforts of volunteers from the community.

13.2 Problems with Current TIP Infrastructure.

The Fellows web interface to the TIP Document Collection (http://www.cs.man.ac.uk/fellowsd-bin/TIP/) offers valuable browsing access to TIPs in a variety of formats. It accomplishes the important goal of making TIPs public. However, it suffers from two significant shortcomings:

- Revisions are only possible through the TIP Editor:

  Currently the only way to revise a TIP is to e-mail a new revision to the TIP Editor and wait for it to replace the old revision. As more TIPs are submitted, and as each TIP is more frequently revised, this bottleneck will not be tolerable.

  Discussion about TIPs currently takes place in Usenet newsgroups and on mailing lists, but because there is no easy access to revising the TIPs themselves, the new information and viewpoints arising in these discussions are not being folded back into the TIPs. This means the TIPs are failing in their intended role to present a full history of an issue to later readers. It also means newcomers to a TIP cannot receive a full briefing in one place, but must chase down discussions in mailing list and Usenet archives. Few people do that, but instead repeat points already made. The discussions about [TIP #6] reflect this problem.

- An archive of each revision of each TIP is not maintained:

  Although [TIP #2] refers to TIPs as being stored in a revision control system, probably a CVS repository, the Fellows TIP collection is not maintained in such a system. Since a TIP is an archive of a public discussion of an issue, it is important to be able to access the history of changes to each TIP.

  The ability to retrieve and restore earlier revisions of a TIP will be especially important if public revision is permitted, so that any TIP contents that are incorrectly removed, whether by accident or malice, can be restored.

13.3 Proposal

An improved system for archiving and revising TIPs is proposed:

1. TIPs will be archived in a CVS repository.

2. Anyone with the power to call a TYANNOTT vote will have commit access to this repository through either :pserver: or ssh access. With this access, they will be able to revise any part (header or body) of any TIP (whether in *State: Draft* or not). Everyone having this access will be trusted to modify TIPs only in conformance with the TIP format and the TCT procedures.

3. An enhanced version of the Fellows TIP rendering engine will display an [Edit] link at the bottom of each TIP eligible for web-editing when that TIP is rendered in HTML.

4. For any TIP in state Draft, and for which a vote is still pending, the [Edit] link will lead to an HTML form for submitting a revised TIP. For other TIPs, no [Edit] link will appear, and an attempt to directly access web-editing of such a TIP will lead to a message stating that the TIP may not be edited through the web interface.

5. The HTML editing form will display the TIP header, but will not make it available for editing. The HTML form will require that an e-mail address be entered, and will allow a name to be entered as well. A <TEXTAREA> will be initialized to hold the current TIP abstract. A second <TEXTAREA> will be initialized to hold the current TIP body. Users of the form will revise the abstract and the body, then submit the form.

6. The TIP rendering engine will receive the revisions, and will use CVS client commands to merge the revisions with other revisions and commit the revised TIP to the TIP CVS repository. If a conflict occurs during a merge, the TIP body including the conflicts will be returned to the user in another HTML form to resolve the conflict.

   Note that the CVS commit function of the TIP rendering engine implies that the CGI process in which the TIP rendering engine runs must have a user ID with commit access to the TIP CVS repository.

7. In the revised TIP checked in to CVS the submitter of the revision will be added as an Author in the header of that TIP.

13.4 Reference Implementation

The modifications to the Fellows TIP rendering engine that add the capabilities proposed above are now merged in. The TIP rendering engine is maintained at http://sourceforge.net/projects/tiprender/ . To enable the web-editing features, set the Tcl variable FEATURE(EDIT) to 1 in the file config.tcl. A working version of the proposed web service is available at http://dev.scriptics.com:8080/cgi-bin/tct/tip/ .

For what it's worth, this TIP was created primarily within a web browser, making revisions through the web interface provided by the reference implementation.

One remaining shortcoming of the reference implementation is that it provides no mechanism for uploading images to the TIP repository. Images still need to be submitted through the TIP Editor, or someone else with commit access to the TIP CVS repository.

13.5 Server Requirements

The reference implementation imposes the following requirements on a server:

1. The server provides an HTTP server that serves the public Internet, and supports the CGI interface.

2. CVS client software must be installed on the server.

3. The CVS repository containing TIPs must be on the server itself. This is due to a CVS limitation that *loginfo* scripts run on the machine housing the CVS repository, and the reference implementation uses a *loginfo* script to keep the TIPs presented through the web up to date with the commits to the repository.

4. The CVS repository must offer commit access over the Internet using either :pserver: or ssh to everyone with authority to call a TYANNOTT vote.

5. The user under which the HTTP server runs its CGI processes must have commit access to the TIP CVS repository. This may have security implications.

13.6 Future Improvements

Once the TIPs are housed in a CVS repository, other services should be easier to implement. Another browsing interface could be provided using cvsweb (http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/) to allow anyone in the community to browse TIP history. Another *loginfo* script could provide e-mail notices when a TIP is revised to users who registered their interest in that TIP.

13.7 Acknowledgments

Thanks to Donal Fellows for the original TIP rendering engine and his assistance merging in the changes. Thanks to Brent Welch for providing the server and his assistance getting it configured for use. Thanks to Mark Harrison for his assistance with managing browser caching issues.

13.8 Comments from the TCT

> It might be a good idea to make the Abstract into a seperate <textarea> and treat that specially; I've been applying the rule that a TIP's abstract should be a single paragraph (it is implicit in the way I generate XML for instance) and that would be much easier to enforce through this route. It would also have the advantage of discouraging people from placing their whole rationale in the abstract — which I've seen in several first drafts by people who shall remain nameless — and prompting the creation of TIPs more in keeping with the general concept of publishable documents.
>
> It would also be nice if each page had a way of viewing the TIP's revision history (by a link to a suitably setup CVSweb URL?) The way that SourceForge does its CVS-over-the-web is very nice indeed...
>
> *Donal K. Fellows* ⟨*fellowsd@cs.man.ac.uk*⟩

13.9 Author replies to comments

Some comments I see above, or received in e-mail:

- *Add a link to [TIP #3] on the edit page. Display [TIP #3] in a new window.*

  Link added. I don't believe in deciding to open new browser windows for the user. If the user wants to open the link in a new window, she knows how to do that.

- *Add an interface to create a new TIP*

  For now, I'm trying to stick with the TIP procedures proposed in [TIP #2], where only the TIP Editor gets to create a new TIP, so all TIPs should still be originally submitted to him. If the TCT rejects this proposal and adopts a different policy, we can revisit this question.

- *Browser caching makes it look like the edits were not made*

  Based on advice from Mark Harrison and others, I've added HTTP headers that should prevent caching. Please try it again.

- *I had the server time out on me*

  That's troubling. Can anyone seeing this problem provide any more information. What were the circumstances, in detail?

- *Add links to an interface showing revision history*

  Check out the [History] links at the bottom of the HTML rendered pages. Thanks to Donal Fellows for coding that up. (OK, so we *did* reinvent CVSWeb.)

- *A separate <TEXTAREA> for the Abstract*

  This is now implemented. Please give it a try.

13.10 Copyright

TIP #14: Access (via tkInt) to Tk Photo Image Transparency

TIP #14:	**Access (via tkInt) to Tk Photo Image Transparency**
Author:	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Wednesday, 22nd November 2000
Type:	Project
Tcl Version:	8.4.0
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.3 $`
Keywords:	Tk, photo, transparency, internal, access
Post-History:	

Abstract

It is useful for some extensions to have access to the transparency information in photo images for various reasons, but this is not currently available, even via an internal structure defined in *generic/tkInt.h*. This TIP is aimed at making the information available in a way that can be kept backwardly compatible even if the internal structure definitions change.

14.1 Rationale

I have been working for several years (on-and-off) on an extension for Tk that allows it to have non-rectangular windows (http://www.cs.man.ac.uk/~fellowsd/tcl/shapeidx.html) which is an effect that is great for all sorts of purposes but which comes particularly into its own when used in conjunction with drag-and-drop to make drag tokens that obscure only part of what lies underneath them. However, one of the most useful ways of specifying the shape of a window turns out to be via images of various kinds, and the natural way to do this is with the transparency data within the image. The problem is that this data is locked up entirely within structures that are completely private to *generic/tkImgPhoto.c*; none of it is visible at all anywhere else, even within the core. (There is code that uses colour data instead to do this sort of trick, http://www.sys.uea.ac.uk/~fuzz/tktrans/default.html, but this is a slow process and frankly a little strange if we already have transparency data available.)

To get around this problem, the data member *validRegion* of the *PhotoMaster* structure needs to be made available by some mechanism. There are two ways of doing this:

1. Placing the *PhotoMaster* structure, or some version of it, in *generic/tkInt.h*, or
2. Creating a function to access the data member.

The first way is very cheap initially, but also very inflexible and creates yet another hidden version dependency (such as is tackled in [TIP #5]) should we decide to change the structure for any reason (we also have had problems with this sort of thing in the past in relation to the *Tcl_Interp* member *result*, direct access to which has been deprecated for years, but where there is still existing code that does it and which forms one of the largest barriers for some extensions from upgrading to Tcl 8.0 or later.) It is also unnecessary since only the core needs to know how to create new instances of the structure.

The second way, by contrast, is far more flexible in the future as it will allow us to completely change the internal implementation of photo image transparency without affecting any extensions at all. The cost of doing this is that a new entry in one of the stub tables must be created. Due to the fact that the type of the *validRegion* member is (currently) internal, I propose adding the function to the *tkInt* stub interface, and I propose calling the function *TkPhotoGetValidRegion*.

This TIP is not a substitute for a more sophisticated proposal that would offer script-level and sophisticated extension-level access to photo transparency, I accept, but at the time of writing I do not see how such an interface would be structured.

14.2 Sample Implementation Patch

http://www.cs.man.ac.uk/~fellowsd/tcl/validRegion.patch

This applies a patch to *tkImgPhoto.c*, *tkInt.h*, *tkInt.decls*, *tkIntDecls.h* and *tkStubInit.c*, all within the *generic* directory of the Tk distribution. Due to the fact that the patch as it stands has no computational component, and it is an internal interface anyway, it includes no documentation or tests.

14.3 Copyright

TIP #15: Functions to List and Detail Math Functions

TIP #15:	**Functions to List and Detail Math Functions**
Author:	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Wednesday, 22nd November 2000
Type:	Project
Tcl Version:	8.4.0
State:	Final
Vote:	Done
Version:	`$Revision: 1.8 $`
Keywords:	Tcl, expr, function, introspection
Post-History:	

Abstract

Provides a way for the list of all math functions defined in the current interpreter to be discovered, and for discovering what arguments might be passed to an existing math function. This may be useful in tests as well as more general use.

15.1 Rationale

Although it is quite easy to define a new function for use in expressions, there is no public way of performing introspection on this information. Having a way to extract the arguments from an existing math function was requested by http://sourceforge.net/bugs/?func=detailbug&bug_id=119304&group_id=10894 and once you have one, it becomes trivial to also ask for a second function to list what functions are defined.

I propose the creation of two functions that fulfil this rle; *Tcl_GetMathFuncInfo* and *Tcl_ListMathFuncs*. These functions will be documented on the same manual page as *Tcl_CreateMathFunc* and implemented in the same file.

Furthermore, I also propose that the *info* command in the Tcl interpreter be extended to include a new subcommand, *functions*, which will allow Tcl scripts to discover the list of installed functions (by acting as a thin veneer over *Tcl_ListMathFuncs*.) Note that this is an extension of the *info* command because it allows for introspection of a system that affects the behaviour of several commands that form the core part of the command-set: *expr*, *for*, *if* and *while*.

15.2 Tcl_GetMathFuncInfo

This function will take an interpreter reference, a function name (as a string) and pointers to variables capable of taking each of the last four arguments to *Tcl_CreateMathFunc*, and will return a standard Tcl result (either *TCL_OK* or *TCL_ERROR*, depending on whether a function with the given name exists within the given interpreter, with an error message being left in the interpreter's result in the *TCL_ERROR* case.) The array of argument types whose reference is placed into the variable pointed to by *argTypesPtr* will be allocated by Tcl, and should be freed with *Tcl_Free*.

```
int Tcl_GetMathFuncInfo(Tcl_Interp *interp, CONST char *name,
                        int *numArgsPtr, Tcl_ValueType **argTypesPtr,
                        Tcl_MathProc **procPtr,
                        ClientData *clientDataPtr);
```

The parameter names are chosen by analogy with *Tcl_CreateMathFunc*.

In the case where a math function is defined internally by the bytecode engine and has no standard implementation (all the builtin functions in 8.4a2 are like this) the value placed in the variable indicated by the *procPtr* argument will be NULL.

15.3 Tcl_ListMathFuncs

This function will take an interpreter reference and an optional string that describes a glob-like pattern that restricts the set of math functions that the caller is interested in receiving (with a *NULL* indicating that no filtering is desired.) The function will return a pointer to a newly-allocated *Tcl_Obj* list of the names of all the math functions defined within that interpreter, or *NULL* in the case of an error (in which case a suitable message will be left in the interpreter.) The list will not be required to be sorted.

```
Tcl_Obj *Tcl_ListMathFuncs(Tcl_Interp *interp, CONST char *pattern);
```

The alternative is to pass in the addresses of variables that will be updated to contain the number of functions and an array of function names. But I prefer the *Tcl_Obj* approach as it is expressing the fact that the list of function names is really a single thing being returned (albeit one that is not a simple value.) It is not anticipated that the performance of this function will need to be crucial to too many applications.

15.4 info functions

This new subcommand will provide access from Tcl scripts to the functionality of *Tcl_ListMathFuncs*. It will take a single optional argument consisting of a pattern to pass on as the *pattern* argument (with the absence of the argument indicating that NULL is to be passed instead.)

15.5 Copyright

TIP #16: Tcl Functional Areas for Maintainer Assignments

TIP #16:	**Tcl Functional Areas for Maintainer Assignments**
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩
Created:	Tuesday, 21st November 2000
Type:	Process
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.9 $`
Post-History:	

Abstract

This document proposes a division of Tcl's source code into functional areas so that each area may be assigned to one or more maintainers.

16.1 Background

TCT procedures (see [TIP #0]) call for each *maintainer* to be responsible for a portion of Tcl's source code. Certain portions of Tcl's source code are naturally associated with certain other portions. (For example, the implementation of a command is intimately related to the documentation for that command.) Establishing a *natural* division of Tcl's source code into units needing maintainers is a useful preliminary effort toward a public call for volunteer maintainers.

16.2 Rationale

When someone reports a bug, or offers a patch, he will want to be able to determine what maintainers have oversight over his report. This implies that we seek a simple mapping from something he knows about his bug or patch to the set of maintainers.

For a patch, the submitter certainly knows what file(s) she is patching. For a bug report, the reporter is likely to know what command or C function he believes to be buggy. Fortunately, every C function or Tcl command (combined with the platform) can be associated with exactly one source code file, the file providing the definition of the C function, or the command procedure of the Tcl command. Thus, a mapping from source code file to maintainer is sufficient to complete the determination.

The source code file should not be the largest unit, however. Certain sets of files should each be gathered into a larger unit, all files in that unit with the same maintainer(s). Tcl's man pages already gather related routines and commands into one page of documentation. Using the modules implied by the man pages, and by the location of routines in particular source code files, a *natural* division of Tcl into minimal maintainer units follows in the Proposal section below.

It may be that some of these minimal units can be joined together into still larger related units. That is not necessary, though. We can just have the same maintainer(s) assigned to all the related minimal units.

16.3 Proposal

Tcl shall be divided into the following 52 functional units, each to be assigned one or more maintainers:

Events

1. *Notifier* — doc/CrtFileHdlr.3, doc/DoOneEvent.3, doc/Notifier.3, doc/Sleep.3, generic/tclNotify.c, mac/tclMacNotify.c, tests/unixNotfy.c, tests/winNotify.c, unix/tclUnixNotfy.c, unix/tclUnixEvent.c, win/tclWinNotify.c
2. *Event Loops* — doc/bgerror.n, doc/update.n, doc/vwait.n, doc/BackgdErr.3, doc/Exit.3, generic/tclEvent.c, tests/event.test
3. *Timer Events* — doc/after.n, doc/CrtTimerHdlr.3, doc/DoWhenIdle.3, generic/tclTimer.c, tests/timer.test
4. *Asynchronous Events* — doc/Async.3, generic/tclAsync.c, tests/async.test
5. *Xt Based Notifier* — unix/tclXtNotify.c, unix/tclXtTest.c
6. *Time Measurement* — compat/gettod.c, compat/strftime.c, doc/clock.n, generic/tclClock.c, generic/tclGetDate.y, mac/tclMacTime.c, tests/clock.test, tests/winTime.test, unix/tclUnixTime.c, win/tclWinTime.c

Variables

1. *Variable Commands and Interfaces* — doc/append.n, doc/array.n, doc/global.n, doc/lappend.n, doc/set.n, doc/unset.n, doc/upvar.n, doc/variable.n, doc/SetVar.3, doc/TraceVar.3, doc/UpVar.3, generic/tclVar.c, tests/append.test, tests/set.test, tests/set-old.test, tests/upvar.test, tests/var.test
2. *Environment Variables* — generic/tclEnv.c, mac/tclMacEnv.c, tests/env.test
3. *Linked C Variables* — doc/LinkVar.3, generic/tclLink.c, tests/link.test

Objects

1. *Object System and Fundamental Object Types* — doc/Backslash.3, doc/BoolObj.3, doc/Concat.3, doc/DoubleObj.3, doc/DString.3, doc/Encoding.3, doc/FindExec.3, doc/Hash.3, doc/IntObj.3, doc/Object.3, doc/ObjectType.3, doc/PrintDbl.3, doc/SplitList.3, doc/StringObj.3, doc/StrMatch.3, generic/tclEncoding.c, generic/tclHash.c, generic/tclObj.c, generic/tclStringObj.c, generic/tclUtil.c, library/encoding/*.enc tests/dstring.test, tests/encoding.test, tests/stringObj.test, tests/obj.test, tests/util.test

2. *Conversions From String* — doc/GetInt.3, generic/tclGet.c, tests/get.test
3. *bytearray Object Type* — doc/binary.n, doc/ByteArrObj.3, generic/tclBinary.c, tests/binary.test
4. *index Object Type* — doc/GetIndex.3, doc/WrongNumArgs.3, generic/tclIndexObj.c, tests/indexObj.test
5. *list Object Type* — doc/ListObj.3, generic/tclListObj.c, tests/listObj.test

Fundamental Built-in Commands

1. *A — H* — doc/break.n, doc/case.n, doc/catch.n, doc/cd.n, doc/concat.n, doc/continue.n, doc/encoding.n, doc/error.n, doc/eval.n, doc/exit.n, doc/expr.n, doc/file.n, doc/for.n, doc/foreach.n, doc/format.n, generic/tclCmdAH.c, tests/cmdAH.test tests/case.test, tests/concat.test, tests/error.test, tests/eval.test, tests/foreach.test, tests/format.test, tests/for-old.test
2. *I — L* — doc/if.n, doc/incr.n, doc/info.n, doc/join.n, doc/lindex.n, doc/linsert.n, doc/list.n, doc/llength.n, doc/lrange.n, doc/lreplace.n, doc/lsearch.n, doc/lsort.n, generic/tclCmdIL.c, tests/cmdIL.test, tests/if-old.test, tests/incr-old.test, tests/info.test, tests/join.test, tests/lindex.test, tests/linsert.test, tests/list.test, tests/llength.test, tests/lrange.test, tests/lreplace.test, tests/lsearch.test
3. *M — Z* — doc/pwd.n, doc/regexp.n, doc/regsub.n, doc/rename.n, doc/return.n, doc/split.n, doc/string.n, doc/subst.n, doc/switch.n, doc/time.n, doc/trace.n, doc/while.n, generic/tclCmdMZ.c, tests/cmdMZ.test tests/pwd.test, tests/rename.test, tests/split.test, tests/string.test, tests/subst.test, tests/switch.test, tests/trace.test, tests/while-old.test
4. *[history]* — doc/history.n, doc/RecEvalObj.3, doc/RecordEval.3, generic/tclHistory.c, library/history.tcl, tests/history.test
5. *[interp]* — doc/interp.n, doc/CrtSlave.3, generic/tclInterp.c, tests/interp.test
6. *[namespace]* — doc/namespace.n, generic/tclNamesp.c, generic/tclResolve.c, tests/namespace.test, tests/namespace-old.test
7. *[proc]* — doc/proc.n, doc/uplevel.n, generic/tclProc.c, generic/tclTestProcBodyObj.c, tests/proc.test, tests/proc-old.test, tests/uplevel.test
8. *[scan]* — doc/scan.n, generic/tclScan.c, tests/scan.test

Channels

1. *Channel Commands* — doc/close.n, doc/eof.n, doc/exec.n, doc/fblocked.n, doc/fconfigure.n, doc/fcopy.n, doc/flush.n, doc/gets.n, doc/open.n, doc/puts.n, doc/read.n, doc/seek.n, doc/socket.n, doc/tell.n, generic/tclIOCmd.c, tests/exec.test, tests/ioCmd.test, tests/remote.test, tests/socket.test
2. *Channel System* — doc/fileevent.n, doc/ChnlStack.3, doc/CrtChnlHdlr.3, doc/CrtCloseHdlr.3, doc/CrtChannel.3, doc/DetachPids.3, doc/GetStdChan.3, doc/OpenFileChnl.3, generic/tclIO.c, generic/tclIO.h, generic/tclPipe.c, tests/io.test
3. *Channel Transformations* — generic/tclIOGT.c, tests/iogt.test
4. *Built-in Channel Types* — compat/waitpid.c, doc/GetHostName.3, doc/GetOpnFl.3, doc/OpenTcp.3, doc/pid.n, generic/tclIOSock.c, mac/tclMacChan.c, mac/tclMacSock.c, tests/pid.test, tests/winConsole.test, tests/winPipe.test, unix/tclUnixChan.c, unix/tclUnixPipe.c, unix/tclUnixSock.c, win/cat.c, win/stub16.c, win/tclWinChan.c, win/tclWinConsole.c, win/tclWinPipe.c, win/tclWinSerial.c, win/tclWinSock.c

Packages

1. *dde Package* — doc/dde.n, library/dde/pkgIndex.tcl, tests/winDde.test win/tclWinDde.c
2. *http Package* — doc/http.n, library/http1.0/http.tcl, library/http1.0/pkgIndex.tcl, library/http/http.tcl, library/http/pkgIndex.tcl, tests/http.test, tests/httpd, tests/httpold.test
3. *msgcat Package* — doc/msgcat.n, library/msgcat/msgcat.tcl, library/msgcat/pkgIndex.tcl, tests/msgcat.test
4. *opt Package* — library/opt/optparse.tcl, library/opt/pkgIndex.tcl, tests/opt.test
5. *registry Package* — doc/registry.n, library/reg/pkgIndex.tcl, win/tclWinReg.c, tests/registry.test
6. *Safe Base* — doc/safe.n, library/safe.tcl, tests/safe.test
7. *tcltest Package* — doc/tcltest.tcl, library/tcltest/tcltest.tcl, library/tcltest/pkgIndex.tcl, tests/tcltest.test

File System

1. *Pathname Management* — doc/filename.n, doc/glob.n, doc/SplitPath.3, doc/Translate.3, mac/tclMacFile.c, generic/tclFileName.c, tests/fileName.test, tests/unixFile.test, tests/winFile.test, unix/tclUnixFile.c, win/tclWinFile.c
2. *File System Access* — doc/Access.3, doc/GetCwd.3, doc/SetErrno.3, generic/tclFCmd.c, generic/tclIOUtil.c, generic/tclPosixStr.c, mac/tclMacFCmd.c, tests/fCmd.test, tests/ioUtil.test, tests/macFCmd.test, tests/unixFCmd.test, tests/winFCmd.test, unix/tclUnixFCmd.c, win/tclWinError.c, win/tclWinFCmd.c

Initialization, Script Library, and Autoloader 1. doc/library.n, doc/tclvars.n, doc/unknown.n, doc/Init.3, doc/SourceRCFile.3, generic/tclInitScript.h, library/auto.tcl, library/init.tcl, library/parray.tcl, library/word.tcl, mac/tclMacInit.c, tests/autoMkindex.tcl, tests/autoMkindex.test, tests/init.test, tests/platform.test, tests/security.test, tests/unixInit.test, tests/unknown.test, unix/tclUnixInit.c, win/tclWinInit.c

Package Support 1. *Package Management* — doc/InitStubs.3, doc/package.n, doc/packagens.n, doc/pkgMkIndex.n, doc/PkgRequire.3, generic/tclPkg.c, generic/tclStubLib.c, library/package.tcl, tests/package.test, tests/pkg.test, tests/pkgMkIndex.test, tests/pkg/*.tcl

2. *Dynamic Loading* — compat/dlfcn.h, doc/load.n, doc/StaticPkg.3, generic/tclLoad.c, generic/tclLoadNone.c, library/ldAout.tcl, mac/tclMacLoad.c, tests/load.test, unix/dltest/*, unix/tclLoad*.c, win/tclWinLoad.c

Memory Management 1. *Allocation* — doc/memory.n, doc/Alloc.3, doc/TCL_MEM_DEBUG.3, doc/DumpActiveMemory.3, generic/tclAlloc.c, generic/tclCkalloc.c

2. *Preservation* — doc/Preserve.3, generic/tclPreserve.c

Regular Expressions 1. doc/re_syntax.n, doc/RegExp.3, generic/regc_color.c, generic/regc_cvec.c, generic/regc_lex.c, generic/regc_locale.c, generic/regc_nfa.c, generic/regcomp.c, generic/regcustom.h, generic/rege_dfa.c, generic/regerror.c, generic/regerrs.h, generic/regex.h, generic/regexec.c, generic/regfree.c, generic/regfronts.c, generic/regguts.h, generic/tclRegexp.c, generic/tclRegexp.h, tests/reg.test, tests/regexp.test, tools/uniClass.tcl

UTF-8 String Management 1. doc/ToUpper.3, doc/Utf.3, generic/tclUtf.c, tools/uniParse.tcl, tests/utf.test, win/tclWin32Dll.c

Fundamental Parsing and Evaluation 1. doc/AddErrInfo.3, doc/AllowExc.3, doc/AssocData.3, doc/CallDel.3, doc/CmdCmplt.3, doc/CrtCommand.3, doc/CrtObjCmd.3, doc/CrtInterp.3, doc/CrtMathFnc.3, doc/CrtTrace.3, doc/Eval.3, doc/ExprLong.3, doc/ExprLongObj.3, doc/GetVersion.3, doc/Interp.3, doc/ParseCmd.3, doc/SaveResult.3, doc/SetRecLimit.3, doc/SetResult.3, doc/Tcl.n, generic/tclBasic.c, generic/tclParse.c, generic/tclParseExpr.c, generic/tclResult.c, tests/assocd.test, tests/basic.test, tests/cmdInfo.test, tests/dcall.test, tests/expr-old.test, tests/parse.test, tests/parseExpr.test, tests/parseOld.test, tests/result.test, tests/stack.test

Bytecode 1. compat/float.h, generic/tclCompCmds.c, generic/tclCompExpr.c, generic/tclCompile.c, generic/tclCompile.h, generic/tclExecute.c, generic/tclLiteral.c tests/compExpr-old.test, tests/compExpr.test, tests/compile.test, tests/execute.test, tests/expr.test, tests/for.test, tests/if.test, tests/incr.test, tests/while.test

Threads 1. doc/Thread.3, generic/tclThread.c, generic/tclThreadJoin.c, mac/tclMacThrd.c, mac/tclMacThrd.h, tests/thread.test, unix/tclUnixThrd.c, unix/tclUnixThrd.h, win/tclWinThrd.c, win/tclWinThrd.h

Embedding Support 1. doc/AppInit.3, doc/Tcl_Main.3, doc/tclsh.1, generic/tclMain.c, generic/tclPanic.c, mac/tclMacAppInit.c, mac/tclMacPanic.c, unix/tclAppInit.c, win/tclAppInit.c

Release Engineering 1. *Release Notes* — README, changes, license.terms, */license.terms, compat/README, generic/README, mac/README, mac/bugs.doc, tests/README, tests/pkg/license.terms, tools/README, unix/README, win/README

2. *Portability Support* — compat/dirent.h, compat/dirent2.h, compat/tclErrno.h, compat/unistd.h, generic/tclMath.h, generic/tclPort.h, mac/tclMacPort.h, unix/tclMtherr.c, unix/tclUnixPort.h, win/tclWinMtherr.c, win/tclWinPort.h

3. *Configuration and Build Tools* — tests/all.tcl, tools/configure.in, tools/cvtEOL.tcl, tools/genStubs.tcl, tools/index.tcl, tools/Makefile.in, tools/man2help.tcl, tools/man2help2.tcl, tools/man2tcl.c, tools/tcl.hpj.in, tools/tcl.wse.in, tools/tclSplash.bmp, tools/tcltk-man2html.tcl, tools/white.bmp, unix/Makefile.in, unix/aclocal.m4, unix/configure.in, unix/install-sh, unix/ldAix, unix/mkLinks, unix/mkLinks.tcl, unix/tcl.m4, unix/tcl.spec, unix/tclConfig.sh.in, win/Makefile.in, win/aclocal.m4, win/configure.in, win/makefile.vc, win/mkd.bat, win/rmd.bat, win/tcl.hpj.in, win/tcl.m4, win/tcl.rc win/tclConfig.sh.in, win/tclsh.ico, win/tclsh.rc

4. *Other Tools* — tools/checkLibraryDoc.tcl, tools/genWinImage.tcl, tools/man2html.tcl, tools/man2html1.tcl, tools/man2html2.tcl, tools/regexpTestLib.tcl

Macintosh Stuff 1. *[resource]* — doc/resource.n, doc/source.n, mac/tclMacResource.c, mac/tclMacResource.r, tests/resource.test, tests/source.test

2. *Mac-Specific Files* — mac/AppleScript.html mac/Background.doc mac/libmoto.doc mac/morefiles.doc mac/MW_TclAppleScriptHeader.h mac/MW_TclAppleScriptHeader.pch mac/MW_TclHeader.h mac/MW_TclHeader.pch mac/MW_TclTestHeader.h mac/MW_TclTestHeader.pch mac/porting.notes mac/tclMac.h mac/tclMacAETE.r

mac/tclMacAlloc.c mac/tclMacApplication.r mac/tclMacBOAAppInit.c mac/tclMacBOAMain.c mac/tclMacCommonPch.h mac/tclMacDNR.c mac/tclMacExit.c mac/tclMacInterupt.c mac/tclMacLibrary.c mac/tclMacLibrary.r mac/tclMacMath.h mac/tclMacMSLPrefix.h mac/tclMacOSA.c mac/tclMacOSA.exp mac/tclMacOSA.r mac/tclMacProjects.sea.hqx mac/tclMacShLib.exp mac/tclMacTclCode.r mac/tclMacUnix.c mac/tclMacUtil.c tests/osa.test

16.4 Shared Files

The following files are shared by all of Tcl. Any maintainer may modify them as necessary to complete changes they are making to their portion of Tcl. Some of the following files define Tcl's API and should be changed only in accordance with TCT approval.

- ChangeLog, compat/limits.h, compat/fixstrtod.c, compat/memcmp.c, compat/opendir.c, compat/stdlib.h, compat/string.h, compat/strncasecmp.c, compat/strstr.c, compat/strtod.c, compat/strtol.c, compat/strtoul.c, compat/tmpnam.c, doc/man.macros, generic/tcl.decls, generic/tcl.h, generic/tclInt.decls, generic/tclInt.h, generic/tclTest.c, generic/tclTestObj.c, mac/tclMacInt.h, mac/tclMacTest.c, tests/misc.test, unix/tclUnixTest.c, win/tclWinInt.h, win/tclWinTest.c

16.5 Generated Files

The following files are generated, so they don't need maintainers.

- generic/tclDate.c, generic/tclUniData.c, generic/tclDecls.h, generic/tclIntDecls.h, generic/tclIntPlatDecls.h, generic/tclPlatDecls.h, generic/tclStubInit.c, library/tclIndex, unix/configure, win/configure

16.6 Copyright

TIP #17: Redo Tcl's filesystem

TIP #17:	**Redo Tcl's filesystem**
Author:	Vince Darley ⟨vince@santafe.edu⟩
Created:	Friday, 17th November 2000
Type:	Project
Tcl Version:	8.4.0
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.17 $`
Post-History:	

Abstract

Many of the most exciting recent developments in Tcl have involved putting virtual file systems in a file (e.g. Prowrap, Freewrap, Wrap, TclKit) but these have been largely *ad hoc* hacks of various internal APIs. This TIP seeks to replace this with a common underlying API that will, in addition, make porting of Tcl to new platforms a simpler task as well.

17.1 Overview

There are two current drawbacks to Tcl's filesystem implementation:

- virtual filesystems are not properly supported.
- it is all string-based, rather than Tcl_Obj-based.

Prowrap (http://sourceforge.net/projects/tclpro), Freewrap (http://home.nycap.rr.com/dlabelle/freewrap/freewrap.html), Wrap (http://members1.chello.nl/~j.nijtmans/wrap.html), TclKit (http://www.equi4.com/jcw/wiki.cgi/19.html), ... are all attempts to provide an ability to place Tcl scripts and other data inside a single file (or just a small number of files). The best and simplest way to achieve that task (and many other useful tasks) is to let Tcl handle the contents of a single 'wrapped document' as if it were a filesystem: the contents may be opened, sourced, stat'd, copied, globbed, etc. Also note that at the European Tcl/Tk meeting, the (equal) second-ranked request for Tcl was support for standalone executables (http://mini.net/cgi-bin/wikit/837.html).

This TIP suggests that Tcl's core be modified to allow non-native filesystems to be plugged in to the core, and hence allow *perfect* virtual filesystems to exist. The implementations provided by all of the above tools are very far from perfect. The most obvious types of virtual filesystem which should be supported are:

- wrapped/archived document 'bundles' such as TclKits, .zip files, etc.
- remote filesystems (e.g. an FTP site).

but the main point is that all filesystem access should occur through a hookable interface, so that Tcl neither knows nor cares what type of filesystem it is dealing with.

Furthermore this hookable interface should be Tcl_Obj based, providing a new 'Path' object type, which should be designed with two goals in mind:

- allow caching of 'native path representations' (all native Tclp... filesystem calls involve various Utf->Native conversions). For example, quick testing for 'file exists' shows that a 20% speed up can be achieved by caching the native representation (Windows 2000sp1).
- allow virtual filesystems to operate very efficiently — this will probably require caching of the filesystem to use for a particular file.

If all of these goals are achieved, Tcl will have a new filesystem which is both more efficient and more powerful than the existing implementation.

17.2 Technical discussion

1. Virtual filesystems

   An examination of the core shows that a very limited support was added to tclIOUtil.c in June 1998 (presumably by Scriptics to support prowrap) so that TclStat, TclAccess and Tcl_OpenFileChannel commands could be intercepted. (See http://cvs.sourceforge.net/cgi-bin/cvsweb.cgi/tcl/generic/tclIOUtil.c?rev=1.2&content-type=text/x-cvsweb-markup&cvsroot=tcl)

   This TIP seeks to provide a *complete* implementation of virtual file system support, rather than these piecemeal functions.

   Fortunately, since Tcl is already abstracted across three different filesystem types (through the Tclp...) functions, it is not that big a task to abstract away to any generic filesystem.

   One goal of this TIP is to allow an *extension* to be written so that one can implement a virtual filesystem entirely in Tcl: i.e. to provide sufficient hooks into Tcl's core so that an extension can capture all filesystem requests and divert them if desired. The goal is not to provide Tcl-level hooks in Tcl's core. Such hooks will only be at the C level, and an extension would be required to expose them to the Tcl level.

2. Objectified filesystem interface.

   Every filesystem access in Tcl's core usually involves several calls to 'access', 'stat', etc.

   For example 'file atime $path' requires two calls to 'stat' and one call to 'utime', all with the same $path argument. Each of these requires a conversion from the same Utf path to the same native string representation. No caching is performed, so each of these goes through Tcl_UtfToExternal. Often Tcl code will use the same $path objects for an entire sequence of Tcl 'file' operations. Clearly a representation which cached the native path would speed up all of these operations (except the first).

   The second reason why objectification is desirable is that in a pluggable-fs environment we must determine, for each file operation, which filesystem to use (whether native, a mounted .zip file, a remote FTP site, etc.). If this information can be cached for a particular path, again we will not need to recalculate it at every step. A similar technique to that used by Tcl's bytecode compilation will be used: each cached object will have a "filesystemEpoch" counter, so that we can tell with each access whether the filesystem has been modified (and we must discard the cached information). Mounting/unmounting filesystems will obviously modify the filesystemEpoch.

   A relatively complete implementation of this TIP, and a sample "vfs" extension now exist, and have been tested through TclKit. On both the "virtual" and "objectification" parts of this tip, the implementation is known to be stable and complete (at least on Windows): TclKit can operate through this new vfs implementation without the need to override a single Tcl core command at the script level, and all reasonable filesystem tests (cmdAH.test, fCmd.test, fileName.test, io.test) pass in a scripted document. Commands which operate on files (image, source, etc.) and extensions like Image, Winico can be made to work in a TclKit automatically! There is still some room for optimisation in some parts of the new objectification code (which wasn't possible in the old string-based API). The current implementation has great efficiency gains for vfs's implemented at the script level, since the same Path objects can be passed through the entire process, without an intermediate conversion (and string duplication which would otherwise be required). The combination of caching and objectification changes the existing list of steps from

   ```
   Tcl_Obj -> string -> filesystem -> convert-to-native -> native-call
   ```

   or (with vfs hooked in)

   ```
   Tcl_Obj -> string -> vfilesystem -> pick-filesystem
                             -> convert-to-native -> native-call
   ```

   and

   ```
   Tcl_Obj -> string -> vfilesystem -> pick-filesystem
                             -> Tcl_NewStringObj -> Tcl-vfs-call
   ```

   to

   ```
   Tcl_Obj -> vfilesystem -> native-call
   ```

   and

   ```
   Tcl_Obj -> vfilesystem -> Tcl-vfs-call
   ```

A final side-benefit of this proposal would be that it further modularises the core of Tcl, so that one could, in principle:

- remove the native filesystem support entirely from Tcl (perhaps useful for embedded devices etc), since there will be a clean layer separating Tcl from its native filesystem functionality.
- use Tcl's filesystem for other purposes (outside of Tcl).

However these final two points are explicitly *not* the goal of this TIP! I simply want to improve Tcl to add vfs support, and the best way to do that seems (to me) to be along the lines of this TIP.

17.3 Proposal

The changes to Tcl's core for virtual filesystem support are actually very minor. Every occurrence of a Tclp-filesystem call must be replaced by a call to a hookable procedure. The current filesystem structure (defined in tclInt.h) and hookable procedure list is as follows (for documentation on this structure, see Documentation section below):

```
/*
 * struct Tcl_Filesystem:
 *
 * One such structure exists for each type (kind) of filesystem.
 * It collects together in one place all the functions that are
 * part of the specific filesystem.  Tcl always accesses the
 * filesystem through one of these structures.
 *
 * Not all entries need be non-NULL; any which are NULL are simply
 * ignored.  However, a complete filesystem should provide all of
 * these functions.
 */

typedef struct Tcl_Filesystem {
    CONST char *typeName;   /* The name of the filesystem. */
    int structureLength;    /* Length of this structure, so future
                             * binary compatibility can be assured. */
    Tcl_FilesystemVersion version;
                            /* Version of the filesystem type. */
    TclfsPathInFilesystem_ *pathInFilesystemProc;
                            /* Function to check whether a path is in this
                             * filesystem */
    TclfsDupInternalRep_ *dupInternalRepProc;
                            /* Function to duplicate internal fs rep */
    TclfsFreeInternalRep_ *freeInternalRepProc;
                            /* Function to free internal fs rep */
    TclfsInternalToNormalizedProc_ *internalToNormalizedProc_;
                            /* Function to convert internal representation
                             * to a normalized path */
    TclfsConvertToInternalProc_ *convertToInternalProc_;
    /* Function to convert object to an
     * internal representation */
    TclfsStatProc_ *statProc; /* Function to process a 'Tcl_Stat()' call */
    TclfsAccessProc_ *accessProc;
                            /* Function to process a 'Tcl_Access()' call */
    TclfsOpenFileChannelProc_ *openFileChannelProc;
                            /* Function to process a 'Tcl_OpenFileChannel()' call */
    TclfsMatchInDirectoryProc_ *matchInDirectoryProc;
                            /* Function to process a 'Tcl_MatchInDirectory()' */
    TclfsGetCwdProc_ *getCwdProc;
                            /* Function to process a 'Tcl_GetCwd()' call */
    TclfsChdirProc_ *chdirProc;
                            /* Function to process a 'Tcl_Chdir()' call */
    TclfsLstatProc_ *lstatProc;
                            /* Function to process a 'Tcl_Lstat()' call */
    TclfsCopyFileProc_ *copyFileProc;
                            /* Function to process a 'Tcl_CopyFile()' call */
    TclfsDeleteFileProc_ *deleteFileProc;
                            /* Function to process a 'Tcl_DeleteFile()' call */
    TclfsRenameFileProc_ *renameFileProc;
                            /* Function to process a 'Tcl_RenameFile()' call */
    TclfsCreateDirectoryProc_ *createDirectoryProc;
                            /* Function to process a 'Tcl_CreateDirectory()' call */
    TclfsCopyDirectoryProc_ *copyDirectoryProc;
                            /* Function to process a 'Tcl_CopyDirectory()' call */
```

```
    TclfsRemoveDirectoryProc_ *removeDirectoryProc;
                            /* Function to process a 'Tcl_RemoveDirectory()' call */
    TclfsLoadFileProc_ *loadFileProc;
                            /* Function to process a 'Tcl_LoadFile()' call */
    TclfsUnloadFileProc_ *unloadFileProc;
                            /* Function to unload a previously successfully
                             * loaded file */
    TclfsReadlinkProc_ *readlinkProc;
                            /* Function to process a 'Tcl_Readlink()' call */
    TclfsListVolumesProc_ *listVolumesProc;
                            /* Function to list any filesystem volumes added
                             * by this filesystem */
    TclfsFileAttrStringsProc_ *fileAttrStringsProc;
                            /* Function to list all attributes strings which
                             * are valid for this filesystem */
    TclfsFileAttrsGetProc_ *fileAttrsGetProc;
                            /* Function to process a 'Tcl_FileAttrsGet()' call */
    TclfsFileAttrsSetProc_ *fileAttrsSetProc;
                            /* Function to process a 'Tcl_FileAttrsSet()' call */
    TclfsUtimeProc_ *utimeProc;
                            /* Function to process a 'Tcl_Utime()' call */
    TclfsNormalizePathProc_ *normalizePathProc;
                            /* Function to normalize a path */
} Tcl_Filesystem;
```

Once that is done, almost no more *changes* need be made to Tcl's core. We must simply add code (to tclIOUtil.c and declarations to tclInt.h) to implement the hookable functions and to provide a simple API by which extensions can hook into the new filesystem support.

This gives us the simplest level of vfs. Most remaining changes are objectifying the way Tcl's core uses filesystems. Many of these changes actually simplify the core, for example, we replace:

```
        case FILE_COPY: {
            int result;
            char **argv;

            argv = StringifyObjects(objc, objv);
            result = TclFileCopyCmd(interp, objc, argv);
            ckfree((char *) argv);
            return result;
        }
```

with

```
        case FILE_COPY: {
            return TclFileCopyCmd(interp, objc, objv);
        }
```

and the Unix versions of stat, access, chdir are as simple as:

```
int TclpObjStat(Tcl_Obj *pathPtr, struct stat *buf) {
    return stat(Tclfs_GetNativePath(pathPtr), buf);
}
int TclpObjChdir(Tcl_Obj *pathPtr) {
    return chdir(Tclfs_GetNativePath(pathPtr));
}
int TclpObjAccess(Tcl_Obj *pathPtr, int mode) {
    return access(Tclfs_GetNativePath(pathPtr));
}
```

There are a few other small changes required, some which are absolutely necessary, and some which make the implementation of a Tcl-level vfs much simpler and more robust:

- Cross-filesystem copy and rename operations will fail. A patch was added so that Tcl can fall back on 'open r/open w/fcopy/file mtime' as a copying method for files, and a new function ::tcl::copyDirectory for directories. These techniques are only used if the source/dest are in different filesystems, or if the filesystem Tcl tries to use returns the EXDEV posix result in 'errno'. This is a natural extension of Tcl's current way of falling back from 'rename' to 'copy and delete'.
- Add *-tails* flag to *glob* (and internally to *TclGlob*) to indicate that we only want the tails of the files to be returned.
- Add *file normalize path* subcommand to *file*, which returns an absolute path in which all "..", "." sequences have been removed, and the file is a platform-normalized path (e.g. the longname is used on windows).
- Modify the TclDoGlob implementation so it handles recursion itself, rather than passing it on to the various TclpMatchFilesTypes functions. This simplifies the platform-specific code, and makes vfs support of *glob* much more robust, easy, complete, etc. This has the side- benefit that TclpMatchFilesTypes need not operate directly on the interpreter's result. The simpler function with a different signature has been named *TclpMatchInDirectory*.
- Modify the implementation of *encoding names* to use the TCL_GLOBMODE_TAILS flag to TclGlob, simplifying that code.
- Add an API to tclIO.c to allow us to Unregister a channel *without* deleting it. We need this to be able to take a channel created in Tcl (registered and with refcount of 1) and turn it into a "pristine channel with refcount 0" as returned by Tcl_OpenFileChannel. This is called *Tcl_DetachChannel*.
- Add a function to tclInt.decls called 'TclpVerifyInitialEncodings' which is required when all of Tcl is packaged in a virtual filesystem (e.g. TclKit), since Tcl's very early call to TclpSetInitialEncodings fails to achieve anything useful.
- the perfect vfs support can have some weird side-effects. For instance, if I embed all of tcltest and tests/ inside a TclKit, and try to source 'all.tcl', I get errors that each file does not exist. This is because the test code tries to pipe each file in turn to a newly created tcl process (open "| tclsh foo.test r"), but the files don't really exist (as far as the OS is concerned). We therefore add an introspection command 'file system $path' which returns a list of two items: the name of the filesystem and the type of the file within that filesystem. For example it might be 'native local', 'native networked', 'vfs ftp', 'vfs zip' etc.
- vfs systems on different platforms may require different directory separator characters (different to the native characters), especially on MacOS (in which a valid file might be *HD:Tcl:tcl.zip/lib/tcl8.4/history.tcl*), and we therefore add 'file separator ?filename?' command to retrieve the correct separator to use.

As mentioned above an implementation of all of this now exists. The modified Tcl core passes all Tcl tests, and works with a new version of TclKit (which can itself pass all reasonable tests when executing the test suite inside a scripted document). It has been tested with a variety of wrapped demos (tclhttpd, bwidgets, widgets, alphatk), and performs very well. The patch is available from the author of this TIP, and a version (possibly not the most recent) can be downloaded from: ftp://ftp.ucsd.edu/pub/alpha/tcl/tclobjvfs.diff

17.4 Documentation: vfs-aware extensions

All calls to filesystem functions in Tcl's core and in extensions should preferably be made through the new API defined in tclInt.decls. All these functions have the prefix 'Tclfs_'. Of course extensions which call older string-based APIs (e.g. Tcl_OpenFileChannel) will still work, but will not benefit from the efficiency of the cached object representation. Most of these functions are not commonly used by extensions (e.g. Tclfs_CopyDirectory), so only the most common are listed here:

```
Tcl_Channel Tclfs_OpenFileChannel(Tcl_Interp *interp, Tcl_Obj *pathPtr,
                                  char *modeString, int permissions)
int         Tclfs_EvalFile(Tcl_Interp *interp, Tcl_Obj *fileName)
int         Tclfs_Stat(Tcl_Obj *pathPtr, struct stat *buf)
int         Tclfs_Access(Tcl_Obj *pathPtr, int mode)
```

These replace the equivalent string-based APIs.

```
int      Tclfs_ConvertToPathType(Tcl_Interp *interp, Tcl_Obj *pathPtr)
```

Attempts to convert the given object to a path type. This is a little more than a simple wrapper around *Tcl_ConvertToType(interp, pathPtr, &tclFsPathType)*. If it returns TCL_ERROR, the object is not a valid path. In this sense it is very similar to Tcl_TranslateFileName for the existing string-based API. It should be called before attempting to pass an object to any of the other filesystem APIs (again in much the same way as Tcl_TranslateFileName was used in the core).

```
int      Tclfs_EqualPaths(Tcl_Obj* firstPtr, Tcl_Obj* secondPtr)
Tcl_Obj* Tclfs_SplitPath(Tcl_Obj* pathPtr, int *lenPtr)
Tcl_Obj* Tclfs_JoinPath(Tcl_Obj *listObj, int elements)
Tcl_Obj* Tclfs_JoinToPath(Tcl_Obj *basePtr, int objc, Tcl_Obj *CONST objv[])
```

These all manipulate paths. They return Tcl_Obj* with refCounts of zero.

```
Tcl_Obj* Tclfs_GetNormalizedPath(Tcl_Interp *interp, Tcl_Obj* pathObjPtr)
char*    Tclfs_GetTranslatedPath(Tcl_Interp *interp, Tcl_Obj* pathPtr)
```

and finally:

```
char*    Tclfs_GetNativePath(Tcl_Obj* pathObjPtr)
```

which is used by native filesystems only, and is a shorthand for getting at the cached native representation for MacOS, Windows or Unix (as appropriate). This is always a string based representation, but may really be of type TCHAR* on Windows, for example.

17.5 Documentation: writing a new filesystem

The *objPtr->internalRep.otherValuePtr* field is a pointer to one of these structures, for objects of "path" type.

```
typedef struct FsPath {
    char *translatedPathPtr;    /* Name without any ~user sequences */
    Tcl_Obj *normPathPtr;       /* Normalized absolute path, without .
                                 * or .. sequences, and without ~user
                                 * sequences. */
    Tcl_Obj *cwdPtr;            /* If null, path is absolute, else
                                 * this points to the cwd object used
                                 * for this path.  We have a refCount
                                 * on the object. */
    ClientData nativePathPtr;   /* Native representation of this path,
                                 * which is filesystem dependent. */
    int filesystemEpoch;        /* Used to ensure the path representation
                                 * was generated during the correct
                                 * filesystem epoch.  The epoch changes
                                 * when filesystem-mounts are changed. */
    struct Tcl_FilesystemRecord *fsRecPtr;
                                /* Pointer to the filesystem record
                                 * entry to use for this path. */
} FsPath;
```

Path to filesystem mapping:

```
int TclfsPathInFilesystem_ (Tcl_Obj *pathPtr, ClientData *clientDataPtr)
```

Is the given path in this filesystem? This function should return either -1, or it should return TCL_OK. If it returns TCL_OK, it may wish to set the clientData parameter to point to a filesystem specific representation of the path. (The native filesystem actually postpones the calculation of the native representation until it is requested, but TclKit's vfs immediately allocates a structure containing an int and Tcl_Obj* which it uses as an internal representation).

Internal representation manipulation:

```
void TclfsFreeInternalRep_ (ClientData clientData)
ClientData TclfsDupInternalRep_ (ClientData clientData)
```

These two are called to duplicate and free the clientData field of the FsPath structure. If they are NULL, they are ignored (and on duplication, the new object's clientData field is set to NULL).

Path normalization:

```
int TclfsNormalizePathProc_ (Tcl_Interp *interp, Tcl_Obj *pathPtr,
                             int nextCheckpoint)
```

This function should check the string representation of pathPtr, starting at character index 'nextCheckpoint', and convert it from that point onwards (if possible) to a filesystem-specific unique form. It should return the character index one beyond where is could no longer apply (e.g. pointing to a directory separator or end of string). That index is then passed on to the next filesystem to continue. Most filesystems do not support path ambiguity, in which case the function need not be implemented at all (a NULL entry in the lookup table is acceptable). For example on Windows, the path "c:/PROGRA~1/tcl/tclkit.exe/lib" would be modified to "C:/Program Files/Tcl/tclkit.exe/lib" by the core's normalization procedure, which would return '31', pointing to the '/' in-between .exe and lib.

File manipulation:

For each filesystem function which is implemented, these procs should be declared:

```
TclfsAccessProc_ *accessProc;
TclfsStatProc_ *statProc;
TclfsOpenFileChannelProc_ *openFileChannelProc;
TclfsMatchFilesTypesProc_ *matchFilesTypesProc;
TclfsLstatProc_ *lstatProc;
TclfsCopyFileProc_ *copyFileProc;
TclfsRenameFileProc_ *renameFileProc;
TclfsCopyDirectoryProc_ *copyDirectoryProc;
TclfsDeleteFileProc_ *deleteFileProc;
TclfsCreateDirectoryProc_ *createDirectoryProc;
TclfsRemoveDirectoryProc_ *removeDirectoryProc;
TclfsLoadFileProc_ *loadFileProc;
TclfsReadlinkProc_ *readlinkProc;
TclfsListVolumesProc_ *listVolumesProc;
TclfsFileAttrStringsProc_ *fileAttrStringsProc;
TclfsFileAttrsGetProc_ *fileAttrsGetProc;
TclfsFileAttrsSetProc_ *fileAttrsSetProc;
TclfsUtimeProc_ *utimeProc;
```

In fact, copy/rename file need not be implemented, because Tcl will fallback: from rename to copy and delete, and from copy to open-r/open-w/fcopy/mtime when necessary. However a filesystem may well be able to implement these more efficiently than that.

Cd/pwd support:

```
TclfsChdirProc_ *chdirProc;
TclfsGetCwdProc_ *getCwdProc;
```

the chdir proc need only return TCL_OK if the path is a valid directory, and TCL_ERROR otherwise. There is no need to remember the path in any way. Native filesystems will of course want to make the appropriate system calls to change the real cwd. Most filesystems will not implement the 'Cwd' proc, since Tcl keeps track of the cwd for you. However, the native filesystem should implement it.

Unload file support:

```
TclfsUnloadFileProc_ *unloadFileProc;
```

This function is called automatically by Tcl's core to unload a file, *if* this filesystem was the one which successfully loaded the file initially.

17.6 Philosophy

This TIP is influenced by the thoughts behind the TkGS project (http://sourceforge.net/projects/tkgs/). Whereas TkGS provides a general and efficient graphics system, the aim of this TIP is to provide a similarly general and efficient filesystem.

17.7 Alternatives

1. Alternatives to adding vfs support

   TclKit manages a pretty good job of vfs support. It is limited by the inadequacy of overriding at the Tcl level. Prowrap is limited by the inability to glob, load, cd, pwd, etc.

   There are currently no better alternatives if Tcl's C core calls C functions directly (as it does), or if extensions call C functions directly (as they do), then complete vfs support requires a patch like this to Tcl's core.

2. Alternatives to objectification

   A previous patch added string-based vfs support to Tcl's core, and required very few core changes at all. It could be adopted instead of an objectified filesystem. This would make Tcl's filesystem more complete, but would not make it any more efficient. Also it is much harder to implement complete 'glob' emulation without the newer API.

17.8 Objections

Won't all these hooks slow down Tcl's core a lot?

There are actually remarkably few changes required, so the only slowdown would occur if additional filesystems are hooked into the core. This is similar to the impact of the 'stacked channels' implementation. With the objectified filesystem, this does actually speed up Tcl's core (as remarked above, 'file exists' is 20% faster on Windows 2000).

Won't this break backwards compatibility ("The Tcl question")?

Not at all. With the current vfs patch, the entire test suite passes as before, even with an extra 'reporting' filesystem activated. Most reasonable tests now pass even when the test suite is embedded in a wrapped document.

Won't this make Tcl's core more complex??

Adding a Tcl_Obj interface is definitely a bit more complex in some areas than the existing string-based system, but in other areas it cleans things up a lot. Indeed, one result will be that Tcl's filesystem is properly abstracted away, which conceptually simplifies the core (there will be 10-15 functions which are called for *all* filesystem access, whether it is native or virtual).

17.9 Future thoughts

This section contains items which are outside the scope of this TIP, but it was thought useful to raise and have documented for the record.

- Should we remove the native 'Tclpxx' filesystem functions from Tcl's API? Or perhaps require a new #define TCL_PROVIDE_NATIVE_FILESYSTEM to allow an extension to access these calls? They are all inside tclInt.h, so we could easily protect them with such a define.

This patch still places the native filesystem in a preferential position, and it is hard-coded as the tail of the fs-lookup list. There are two changes which could be made in the future:

- Move the native-fs support to a static extension which is loaded on startup. This would ensure the layer now separating Tcl from the native FS is not violated, and might let others use Tcl or pieces of Tcl in new ways.
- By incorporating some pieces of the 'vfs' extension into the core in the future, and probably making some changes to some of the Tclp native-fs functions, we could make Tcl entirely filesystem-agnostic (e.g. we could do weird things like mount the native filesystem inside a virtual filesystem). Alternatively, if the native filesystem is not loaded at all, that makes for a very good way to ensure a wrapped executable is 'safe', because it cannot even access the local disk.

Also,

- Once prowrap is updated to use the new APIs, we should probably remove the primitive vfs hooks it currently uses, this will remove some obsolete stuff from Tcl's core without affecting anything else (I think — any extensions out there use those APIs?). Prowrap simply needs to register a Tcl_Filesystem with the stat, access and openfilechannel fields set to its existing procedures; all other fields can be NULL. (They would also need to be objectified).
- file copy can now potentially copy across filesystems, which could be both very slow (across the internet) and may even want different eol conventions on each end. We could add a *-command* flag to *file copy* (and perhaps *file rename*), and we could perhaps add optional ways of specifying the encoding/translation of the transfer? (The main issue is to distinguish between text and binary files, which require automatic and binary *-translation* respectively).

17.10 Copyright

TIP #18: Add Labels to Frames

TIP #18:	**Add Labels to Frames**
Author:	Peter Spjuth ⟨peter.spjuth@space.se⟩
Created:	Tuesday, 12th December 2000
Type:	Project
Tcl Version:	8.4
State:	Accepted
Vote:	Done
Version:	`$Revision: 2.2 $`
Post-History:	

Abstract

This TIP proposes to add a labelled frame widget to Tk.

18.1 Introduction

Labelled frames are a common thing in a GUI and the need for them are rather clear by the fact that practically every widget package implements some version of it.

This proposal wants to add simple labelled frames to Tk. Even though a labelled frame can be built by three frames and label, this requires some skill and a bit work. I believe such a basic thing should be easier and this change would make creating a labelled frame as simple as it deserves to be.

Below is an example of what I mean with a labelled frame.

Figure 18.8: Example of labelled frame

18.2 Specification

A new widget class, labelframe, is added. It works like a frame, with the following changes.

These options are added:

-text Standard option. Default value ””.

-font Standard option. Default value same as Label widget.

-fg Standard option. Default value same as Label widget.

-labelwidget Specify a widget to use as label. Default value ””. This option overrides any -text, -font and -fg setting. The widget used must exist before using it as -labelwidget, and if it is not a descendant of the frame it is automatically raised in the stacking order to be visible.

-labelanchor Sets where to place the label. Takes the values nw, n, ne, en, e, es, se, s, sw, ws, w and wn, listing them clockwise. Default value “nw”.

-padx, -pady Standard options. Adds some “air” between the border and the interior of the frame. Default value 0.

These options changes default values:

-borderwidth, new default value 2. -relief, new default value groove.

-padx and -pady are useful in frames and toplevels too, and since it is easy and cheap to add them at the same time, this TIP proposes to add them there too.

18.3 Rationale

My main approach has been to make a simple but still general solution. The most typical usage should be really easy, more advanced usage possible, and more features should be possible to add later if needed.

Trying to mimic all the abilities of a label widget is rather futile. It leads to code duplication and future updates to the label widget would need to be copied too to keep up. Since the most common label is a simple text, the

labelframe only mimics options -text, -font and -fg to be able to handle that case in a simple manner. If you want a more advanced label, e.g. with an image or with a checkbutton, you can get it with -labelwidget.

For placement of the label I chose a style I found in IWidget's "labeledframe" widget. It's the most general solution I can see since it allows access to all twelve obvious positions in an easy way.

Options -padx and -pady does not have anything directly to do with labels, but are a generally nice addition to frames that I have missed a lot in the past. Such padding is not possible without part of the changes to geometry management (see Implementing section) that are required for displaying the label.

The thing about raising the -labelwidget in the stacking order comes from this:

With the most simple implementation, using -labelwidget could be done in two ways:

```
# Way #1
labelframe .f
label .f.l -text Mupp
.f configure -labelwidget .f.l

# Way #2
label .l -text Mupp
labelframe .f -labelwidget .l
raise .l .f
```

In the first you want the label to be a child but since it has to exist, the -labelwidget can't be used on the labelframe creation line.

In the second you try to circumvent it by creating the label first, but then you have to raise it above the labelframe to be visible.

Even though it's just one extra line of code I find it a bit awkward when it's so easy to do something about. The first can be fixed by not trying to do anything with the label widget until idle time when it has had a chance to be created. This is not a good solution though since it leads to some rather awkward things in implementation. The second can be fixed by automatically raising the label in the stacking order when used as -labelwidget. If this is documented clearly I don't have a problem with it, and that is why I chose it.

18.4 Alternatives to this TIP

An alternative way to implement a labelled frame is using mega widget style with a subframe where children are placed. This is how current widget packages do it. I think that is an awkward and unnatural way to handle such a simple thing as a labelled frame. The only reason to do so is that current limitations in geometry management prevents a simpler solution.

I believe that a labelled frame should work like a normal frame. That it displays a label should not matter more than displaying a border or a blue background. A labelled frame megawidget would be different from a frame, the most noticeable difference being that you can't pack/grid things directly into the labelled frame, instead you have to go via a subframe. Having the labelled frame work like a normal frame is more consistent and easier for the programmer at Tcl level.

18.5 Implementing

Implementing this is mostly rather straightforward. The labelframe will share most code with the frame, just like toplevel and frame share code today, and like the spinbox was built on the entry. The tricky part is that limitations in geometry management does not leave room for displaying a label. The changes needed in geometry management are simple but introduces a slight backward incompatibility.

The problem is this. Today a widget can set an internal border width. This defines a uniform width area around the edge of the widget that geometry managers should stay away from. This is not enough though, since to display a label the frame needs to get more space on one side where it will put the label. Also, there is no way for a widget

to affect its own size (anything it says is overridden by pack/grid), so the labelframe cannot make sure that enough size is requested to make room for the label.

By adding some more fields to the TkWindow structure, the information needed can be transferred to the geometry manager.

First, the present internalBorderWidth field is split into four fields, one for each side.

Second, minimum requested width/height fields are added.

This requires one macro per field for reading them and two new APIs to set the fields:

```
void Tk_SetInternalBorderWidthEx(tkwin, left, right, top, bottom)
void Tk_SetMinimumRequestedSize(tkwin, minWidth, minHeight)
```

Geometry managers would need to be updated to take the new fields into consideration, and here is where backwards compatibility comes in. Any extension implementing a geometry manager would need to be updated in the same way as grid/pack/place will be. The change is trivial, and even if not done most things will work anyway. An updated Tk plus an old extension plus an old script will still work and thus no one needs to worry about upgrading.

I consider this a minor thing since it wont break any existing applications. The only thing that will break is if someone would try to use a geometry manager that is not updated within a labelframe. And even in that case you can work around it with an extra frame.

18.6 Rejected alternatives

The ability to display a label could have been given to the normal frame by adding the options above to it. Having a new widget class has the following advantages:

The separate widget class can have its own default values, and the user can control it separately from the frame in the option database.

The normal frame does not need to store all the new options, thus saving memory.

For handling of geometry management, some other solutions was regarded.

Instead of splitting the internalBorderWidth in four, an alternative is just adding two fields. One pointing at a side and one telling how much extra border to put on that side. This only saves one field and is less general. For example, it is not possible to implement -padx and -pady with this one.

A more complex solution using callbacks which was featured in revision 1.1 of this TIP has also been discarded because it was too complex.

It would be possible to do without the minimum requested size fields if you give the responsibility to make sure the label has room to the GUI programmer. This could be rather awkward though, e.g. when making an internationalized application where labels can vary a lot.

18.7 Reference Implementation

An almost finished implementation exists, and it's just a matter of polishing the last bits to create a patch for this proposal if it is accepted.

At http://www.dtek.chalmers.se/~d1peter/labframe.tcl you can find a pure Tcl demo of labelled frames. Even though it uses sub-frames and thus do not live up to what I want to accomplish here it implements all new options as specified here and can be played with if you want to know more.

18.8 Copyright

TIP #19: Add a Text Changed Flag to Tk's Text Widget

TIP #19:	**Add a Text Changed Flag to Tk's Text Widget**
Author:	Neil McKay ⟨mckay@eecs.umich.edu⟩
Created:	Wednesday, 3rd January 2001
Type:	Project
Tcl Version:	$8.4\alpha 2$
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.5 $`
Obsoleted-By:	26
Post-History:	

Abstract

This TIP adds a *text changed* flag to the Tk text widget. The flag would initially be reset, but would be set whenever the contents of the text widget changes.

19.1 Rationale

When creating a text editor, it is often useful to know when the contents of the edit buffer have changed, e.g. in order to ask the user whether or not to save changes on exit. It is possible to create key bindings in Tk's text widget that will set a flag whenever the user changes the widget's contents; however, this is awkward, and it still requires that the programmer set the flag whenever text is changed programmatically. A better solution is to include a *text changed* flag in the code for the text widget itself; this can be accomplished with a relatively small amount of code.

19.2 Flag Behavior

The *text changed* flag should behave as follows:

- It should be reset when the text widget is created
- It should be set whenever characters are inserted into or deleted from the widget
- It must be resettable programmatically via a Tcl command

19.3 Reference Implementation

At the Tcl level, one possible implementation is to add a *changed* widget command to the text widget. One possible syntax for this command is:

```
.txt changed ?boolean?
```

where .txt is a text widget. With no *boolean* argument, the command returns the state of the text-changed flag; with an argument, it sets the state of the text-changed flag to the value of the argument.

19.4 Example

A typical sequence of commands in a text editor would be

1. Create a text widget
2. Read a file and put its contents into the text widget
3. Mark the text as unchanged
4. Edit the text
5. Write the text out, if it has changed.

This could be accomplished by the following Tcl code fragment:

```
grid [button .b -text Quit -command EndEdit]
grid [text .t]

proc EndEdit {} {
    if {[.t changed]} {
        set result [tk_messageBox -type yesno -message "Save changes?"]
        if {[string compare $result "yes"] == 0} {
            set fh [open $fileName "w"]
            puts -nonewline $fh [.t get 1.0 end-1c]
            close $fh
        }
```

```
    }
    exit
}

set fh [open $fileName "r"]
.t insert end [read $fh]
close $fh
.t changed false
```

19.5 Copyright

This document is in the public domain.

19.6 Patch

The *changed* text widget command, as described above, may be added to Tk8.4a2 by applying the patch at http://www.cs.man.ac.uk/fellowsd-bin/TIP/19.patch

TIP #20: Add C Locale-Exact CType Functions

TIP #20:	**Add C Locale-Exact CType Functions**
Author:	Jeffrey Hobbs ⟨jeff.hobbs@acm.org⟩
Created:	Monday, 8th January 2001
Type:	Project
Tcl Version:	$8.4\alpha2$
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.1 $`
Post-History:	

Abstract

This TIP adds functions to Tcl that are a subset of the standard ctype functions (isspace, isalpha, ...) that are ensured to operate only in the C locale (char $<$ 0x80).

20.1 Rationale

Tcl used to force the C locale everywhere in order to have parsing work as expected throughout Tcl, but that prevented certain i18n features from working correctly (like native character input). In enabling the i18n features, some bugs (like http://sourceforge.net/bugs/?func=detailbug&bug_id=127512&group_id=10894) were exposed that required the C locale to be enabled to function properly. Since we don't want to force that requirement, creating ctype functions that work as if they were always in the C locale is the best solution.

20.2 Reference Implementation

Add a file *generic/tclC.c* (to parallel *generic/tclUtf.c*) that contains functions following the convention C_isspace, C_isalpha, ... These functions would use character or bit maps to ensure greatest speed and efficiency of the functions.

Not all use of the ctype functions need be replaced. Those that walk over a string, especially backwards, are the ones that need replacement.

20.3 Copyright

TIP #21: Asymmetric Padding in the Pack and Grid Geometry Managers

TIP #21: **Asymmetric Padding in the Pack and Grid Geometry Managers**	
Author:	D. Richard Hipp ⟨drh@hwaci.com⟩
Created:	Sunday, 14th January 2001
Type:	Project
Tcl Version:	8.4
State:	Final
Vote:	Done
Version:	`$Revision: 1.6 $`
Post-History:	

Abstract

Proposes modifying the *pack* and *grid* geometry managers to support asymmetric padding.

21.1 Rationale

The *pack* and *grid* geometry managers allow for adding extra space to the left and right of a widget using the -padx option, and above and below the widget using -pady. But there is currently no way to add different amounts of space on opposite sides of the widget. When a layout requires differing amounts of space on opposite sides of the same widget, the usual solution is to introduce extra *frame* widgets to act as spacers. But that both complicates the code unnecessarily and obscures the intent of the programmer.

21.2 Proposed Enhancement

This TIP proposes to modify the -padx and -pady of both the *pack* and *grid* geometry managers as follows: If the argument to -padx or -pady is a screen distance, that distance is added to both sides of the widget. (This is the current behavior.) If the argument is a list of two screen distances, then the first screen distance is the extra space to add to the left or top and the second screen distance is the extra space to add to the right or bottom.

The changes to -padx and -pady occur in *pack* only if the pack geometry manager is used according to the new syntax. The older deprecated syntax (that dates from Tk3.3) will not support asymmetric padding.

21.3 Copyright

21.4 Patch

Patches to implement asymmetric padding in the pack and grid geometry managers, as described above, are available at http://www.hwaci.com/sw/asym_pad_patch_2.txt

TIP #22: Multiple Index Arguments to lindex

TIP #22:	**Multiple Index Arguments to lindex**
Author:	David Cuthbert ⟨dacut@kanga.org⟩
	Kevin Kenny ⟨kennykb@acm.org⟩
	Don Porter ⟨dgp@users.sourceforge.net⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Friday, 19th January 2001
Type:	Project
Tcl Version:	$8.4\alpha 2$
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.21 $`
Keywords:	lindex, multiple arguments, sublists
Post-History:	
Discussions-To:	news:comp.lang.tcl,mailto:kennykb@acm.org

Abstract

Obtaining access to elements of sublists in Tcl often requires nested calls to the *lindex* command. The indices are syntactically listed in most-nested to least-nested order, which is the reverse from other notations. In addition, the nesting of command substitution brackets further decreases readability. This proposal describes an extension to the *lindex* command that allows it to accept multiple index arguments, in least-nested to most-nested order, to automatically extract elements of sublists.

22.1 Rationale

The heterogeneous nature of Tcl lists allows them to be applied to a number of useful data structures. In particular, lists can contain elements that are, themselves, valid lists. In this document, these elements are referred to as *sublists.*

Extracting elements from sublists often requires nested calls to *lindex.* Consider, for example, the following Tcl script that prints the center element of a 3-by-3 matrix:

```
    set A {{1 2 3} {4 5 6} {7 8 9}}
    puts [lindex [lindex $A 2] 2]
```

When these calls are deeply nested — e.g., embedded in an *expr* arithmetic expression, having results extracted through *lrange*, etc. — the results are difficult to read:

```
# Print the sum of the center indices of two 3x3 matrices
set p [expr {[lindex [lindex $A 2] 2] + [lindex [lindex $A 2] 2]}]

# Get all but the last font in the following parsed structure:
set pstruct {text {ignored-data
                      { ... }
       }
       {valid-styles
   {justifiction {left centered right full}}
   {font {courier helvetica times}}
       }
 }
return [lrange [lindex [lindex [lindex $pstruct 1] 2] 2] 0 end-1]
```

Note that the list of indices in the latter example is listed in the reverse order of vector indices. In most other languages/domains, the last line might take on one of the following forms:

```
return list_range(pstruct[2][2][1], 0, end-1);

return pstruct[[2, 2, 1]][[0:-1]]

temp = pstruct(2, 2, 1);
result = range(temp, 0, length(temp) - 1);
```

Allowing the *lindex* command to accept multiple arguments would allow this more-natural style of coding to be written in Tcl.

22.2 Specification

1. Under this proposal, the syntax for the *lindex* command is to be modified to accept one of two forms:

```
lindex list indexList
```

or

```
lindex list index1 index2...
```

In either form of the command, the *list* parameter is expected to be a well-formed Tcl list.

In the first form of the command, the *indexList* argument is expected to be a Tcl list comprising one or more list indices, each of which must be an integer, the literal string *end*, or the literal string *end-* followed by an integer with no intervening whitespace. The existing *lindex* command is a degenerate form of this first form, where the list comprises a single index.

In the second form of the command, each of the *index* arguments is expected to be a list index, which again may be an integer, the literal string *end*, or the literal string *end-* followed by an integer with no intervening whitespace.

2. In either form of the command, once the *indexList* parameter is expanded, there is a single *list* parameter and one or more *index* parameters. If there is a single *index* parameter *N*, the behavior is identical to today's *lindex* command, and the command returns the *N*th element of *list*; if *N* is less than zero or at least the length of the list, a null string is returned.

3. If more than one *index* parameter is given, then the behavior is defined recursively; the result of

```
lindex $list $index0 $index1 ...
```

or

```
lindex $list [list $index0 $index1 ...]
```

is indentical to that of

```
lindex [lindex $list $index0] $index1...
```

or, equivalently,

```
lindex [lindex $list $index0] [list $index1...]
```

(This specification does not constrain the implementation, which may be iterative, recursive, or even expanded inline.)

4. When an invalid index is given, an error of the form, *bad index "invalid_index": must be integer or end?-integer?*, where *invalid_index* is the first invalid index encountered, must be returned.

5. If the list argument is malformed, the error resulting from an attempt to convert the list argument to a list must be returned. This behaviour is unchanged from the current implementation.

22.3 Side Effects

1. Whether the result of the *lindex* operation is successful, the underlying Tcl_Obj that represents the list argument may have its internal representation invalidated or changed to that of a list.

22.4 Discussion

Some attention must be paid to giving the *lindex* command adequate performance. In particular, the implementation should address the common case of

```
lindex $list $i
```

where *$i* is a "pure" integer (that is, one whose string representation has not yet been formed). If the above specification is followed naively, the flow will be as follows.

Since *objc* is three, *objv[2]* is expected to be a list. Since it is not, it must be converted from its string representation. It does not have one yet, so the string representation must be formed. Now the string representation is parsed as a list. An array (of length one) of Tcl_Obj pointers is allocated to hold the list in question, and a Tcl_Obj is allocated to hold the single element. Memory is allocated to hold the element's string representation. Now the *lindex* command converts the first element of the list to an index (in this case an integer).

This elaborate ballet of type shimmering requires converting the integer to a string and back again. It also requires four calls to *ckalloc:*

1. Allocate a buffer for the string representation.
2. Allocate the array of Tcl_Obj pointers for the list representation.
3. Allocate the Tcl_Obj that represents the first (and only) element of the list.
4. Allocate a buffer for the string representation of that element.

And at the end, the result is the same integer that was passed as a parameter originally.

To avoid all this overhead in the common case, the proposed implementation shall (in the case where *objc==3*)

1. Test whether *objv[2]* designates an object whose internal representation holds an integer. If so, simply use it as an index.
2. Test whether *objv[2]* designates an object whose internal representation holds a list. If so, perform the recursive extraction of indexed elements from sublists described above.
3. Form the string representation of *objv[2]* and test whether it is *end* or *end-* followed by an integer. If so, use it as an index.
4. Attempt to coerce *objv[2]* to an integer; if successful, use the result as an integer.
5. Attempt to coerce *objv[2]* to a list; if successful, use the result as an index list.
6. Report a malformed *index* argument; the *indexList* parameter is not a well-formed list.

This logic handles all the cases of singleton lists transparently; it is effectively a simple-minded type inference that optimizes away needless conversions.

Assuming that the related [TIP #33] is approved, this logic will most likely be combined with the identical logic required in that proposal for parsing *index* arguments to the *lset* command.

22.5 Comments

Don Porter ⟨dgp@users.sourceforge.net⟩

> I agree that it would be helpful to many programmers to provide a multi-dimensional array data structure that can be accessed in the manner described in this TIP. In the *struct* module of *tcllib*, several other data structures are being developed: graph, tree, queue, stack. I would support adding another data structure to that module that provides an interface like the one described in this TIP, with the intent that all of these helpful data structures find their way into the BI distribution.
>
> I don't see any advantage to adding complexity to [lindex] as an alternative to development of a multi-dimensional array structure. Without a compelling advantage, I'm inclined against making [lindex] more complex. I like having Tcl's built-in commands provide primitive operations, and leave it to packages to combine the primitives into more useful, more complex resources.
>
> This TIP should also consider how any changes to [lindex] mesh with the whole [listx] overhaul of Tcl's [list] command that has been discussed.

Dave Cuthbert ⟨dacut@kanga.org⟩ responds

> Don makes a good point -- with a good set of data structures in tcllib, the need for this TIP is lessened or even eliminated. Nonetheless, I see this as a way of implementing the structures he describes. In other words, a more powerful primitive (which, in reality, adds fairly little complexity when measured in number of lines of code changed) would benefit these structures.

As for the [listx] overhaul, there are many competing proposals for the specification it is difficult to come up with a metric. In writing this TIP, I assumed a vacuum -- that is, a listx command would not be added to the core in the near future.

Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ points out

Although there is tcllib and [listx] to think about, they are certainly not reasons for rejecting this TIP out of hand. The availability of tcllib is not currently anything like universal (not stating whether this is a good, bad or ugly thing) and all the [listx] work will need its own TIP to make it into the core (you tend to have availability problems if it is an extension.) It is not as if the core is short of syntactic sugar right now (the [foreach] command is ample demonstration of this.)

Don Porter ⟨dgp@users.sourceforge.net⟩ follows up

I'll leave the discussion above in place so the history of this TIP is preserved, but I have withdrawn my objection.

There was quite a discussion on news:comp.lang.tcl about using lindex to return multiple arguments. For example:

```
% set list {a {b1 b2} c d e}
% lindex $list 1
b1 b2
% lindex $list 1 0
{b1 b2} a
% lindex $list {1 0}
b1
```

In other words, the list index arguments can, themselves, be lists. Only when the argument is a list would the "recursive selection" procedure of the TIP be used. For multiple arguments, the behaviour is akin to

```
lindex $list a b c  ->
     list [lindex $list a] [lindex $list b] [lindex $list c]
```

Summarised by Dave Cuthbert ⟨dacut@kanga.org⟩

Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ points out

The problems with the above version of multiple indexing are that it loses the property that [lindex] always returns a single element (making writing robust code harder) and that it forces use of the [list] constructor a lot or inefficient type handling when some of the indices must be computed. Then there is the whole question of what happens when you have indexes that are lists of lists, which is a major can of worms.

Luckily, we could always put this sort of behaviour into a separate command (e.g. called [lselect]) which addresses at least the majority of my concerns, and which (in my opinion) need not even form part of this TIP.

Dave Cuthbert ⟨dacut@kanga.org⟩ adds

I intentionally left [lselect] out of the original TIP (and it is still not present in the 02-April-2001 version). As Donal points out, it is a major can of worms and, though there was general agreement on c.l.t that such a command would be useful, people had differing opinions on what form it should take.

Perhaps my view on TIPs is incorrect, but I try to include only sure-fire "yeah, we ought to have done that a few versions ago" items.

22.6 Notes on History of this TIP

This TIP was originally written by Dave Cuthbert ⟨dacut@kanga.org⟩, but ownership has passed (beginning of April) to Kevin Kenny ⟨kennykb@acm.org⟩.

This TIP underwent substantial revision in May of 2001, to add the syntax where all the *index* parameters could be grouped as a list rather than placed inline on the command line.

22.7 See Also

[TIP #33].

22.8 Copyright

TIP #23: Tk Toolkit Functional Areas for Maintainer Assignments

TIP #23:	**Tk Toolkit Functional Areas for Maintainer Assignments**
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
	Jim Ingham ⟨jingham@apple.com⟩
	Don Porter ⟨dgp@users.sourceforge.net⟩
Created:	Monday, 22nd January 2001
Type:	Process
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.19 $`
Post-History:	

Abstract

This document proposes a division of the Tk toolkit's source code into functional areas so that each area may be assigned to one or more maintainers.

23.1 Background

TCT procedures (see [TIP #0]) call for each *maintainer* to be responsible for a portion of the Tk toolkit's source code. Certain portions of the Tk toolkit's source code are naturally associated with certain other portions. (For example, the implementation of a command is intimately related to the documentation for that command.) Establishing a *natural* division of the Tk toolkit's source code into units needing maintainers is a useful preliminary effort toward a public call for volunteer maintainers.

23.2 Rationale

[TIP #16] provides a convincing rationale for establishing a simple mapping from source files to maintainers. It also breaks out maintainers' functional areas for the Tcl core. This document attempts to develop a similar mapping for the Tk toolkit.

Just as with [TIP #16], this document attempts to divide the Tk toolkit into a set of the smallest sensible functional units.

One other factor, which was not addressed in [TIP #16], is that there is considerably more platform dependent code in Tk than in Tcl, and it is unreasonable to expect people to take ownership for pieces of code that run on platforms they don't have access to. However, we want to make sure that the maintainer structure supports the cross-platform nature of Tk.

To that end, in any area where there is both generic code, and platform specific code, we propose that maintainers can sign up for the generic code *and* code for one or more platforms. By overlapping the generic code, we ensure that the public interfaces to Tk will stay consistent among the platforms, while not forcing maintainers to presume expertise in code they can't even compile, much less test or understand fully.

23.3 Proposal

The Tk toolkit shall be divided into the following eighty-six functional units, each to be assigned one or more maintainers. Each area will also be a Category in the SourceForge Tracker for Tk:

Widgets

1. *Bindings* — library/tk.tcl
2. *Appearance* — generic/default.h, mac/tkMacDefault.h, unix/tkUnixDefault.h, win/tkWinDefault.h
3. *[*button] and [label]* — doc/button.n, doc/checkbutton.n, doc/label.n, doc/radiobutton.n, generic/tkButton.c, generic/tkButton.h, library/button.tcl, mac/tkMacButton.c, unix/tkUnixButton.c, tests/butGeom.tcl, tests/butGeom2.tcl, tests/button.test, tests/unixButton.test, tests/winButton.test, win/rc/buttons.bmp, win/tkWinButton.c
4. *Canvas Basics* — doc/CanvPsY.3, doc/CanvTxtInfo.3, doc/CanvTkwin.3, doc/CrtItemType.3, doc/GetDash.3, doc/canvas.n, generic/tkCanvUtil.c, generic/tkCanvas.c, generic/tkCanvas.h, tests/canvas.test
5. *Canvas Items* — generic/tkCanvArc.c, generic/tkCanvBmap.c, generic/tkCanvImg.c, generic/tkCanvLine.c, generic/tkCanvPoly.c, generic/tkCanvText.c, generic/tkCanvWind.c, generic/tkRectOval.c, tests/arc.tcl, tests/canvImg.test, tests/canvRect.test, tests/canvText.test, tests/canvWind.test
6. *Canvas PostScript* — generic/prolog.ps, generic/tkCanvPs.c, library/prolog.ps, tests/canvPs.test, tests/canvPsArc.tcl, tests/canvPsBmap.tcl, tests/canvPsGrph.tcl, tests/canvPsImg.tcl, tests/canvPsText.tcl
7. *[entry]* — doc/entry.n, generic/tkEntry.c, library/entry.tcl, tests/entry.test
8. *[frame] and [toplevel]* — doc/frame.n, doc/toplevel.n, generic/tkFrame.c, tests/frame.test
9. *[listbox]* — doc/listbox.n, generic/tkListbox.c, library/listbox.tcl, tests/listbox.test
10. *Generic Menus* — doc/menu.n, doc/menubutton.n, doc/popup.n, generic/tkMacWinMenu.c, generic/tkMenu.c, generic/tkMenu.h, generic/tkMenuDraw.c, generic/tkMenubutton.c, generic/tkMenubutton.h, library/menu.tcl, library/tearoff.tcl, tests/macWinMenu.test, tests/menu.test, tests/menuDraw.test, tests/menubut.test
11. *Mac Menus* — mac/tkMacMDEF.c, mac/tkMacMDEF.r, mac/tkMacMenu.c, mac/tkMacMenu.r, mac/tkMacMenubutton.c, mac/tkMacMenus.c, tests/macMenu.test

12. *Unix Menus* — tests/unixMenu.test, unix/tkUnixMenu.c, unix/tkUnixMenubu.c
13. *Win Menus* — tests/winMenu.test, win/tkWinMenu.c
14. *[message]* — doc/message.n, generic/tkMessage.c, tests/message.test
15. *[scale]* — doc/scale.n, generic/tkScale.c, generic/tkScale.h, library/scale.tcl, mac/tkMacScale.c, tests/scale.test, unix/tkUnixScale.c
16. *[scrollbar]* — doc/scrollbar.n, generic/tkScrollbar.c, generic/tkScrollbar.h, library/scrlbar.tcl, mac/tkMacScrlbr.c, tests/macscrollbar.test, tests/scrollbar.test, unix/tkUnixScrlbr.c, win/tkWinScrlbr.c
17. *[spinbox]* — doc/spinbox.n, library/spinbox.tcl, tests/spinbox.test
18. *[text]* — doc/text.n, generic/tkText.c, generic/tkText.h generic/tkTextBTree.c, generic/tkTextDisp.c, generic/tkTextImage.c, generic/tkTextIndex.c, generic/tkTextMark.c, generic/tkTextTag.c, generic/tkTextWind.c, library/text.tcl, tests/text.test, tests/textBTree.test, tests/textDisp.test, tests/textImage.test, tests/textIndex.test, tests/textMark.test, tests/textTag.test, tests/textWind.test
19. *Menubars (obsolete)* — doc/menubar.n, library/obsolete.tcl
20. *[tk_optionMenu]* — doc/optionMenu.n, library/optMenu.tcl

Widget Options
1. *Option Parsing* — doc/ConfigWidg.3, doc/SetOptions.3, generic/tkConfig.c, generic/tkOldConfig.c, mac/tkMacConfig.c, unix/tkUnixConfig.c, tests/config.test, win/tkWinConfig.c
2. *Relief* — doc/3DBorder.3, doc/GetRelief.3, generic/tk3d.c, generic/tk3d.h, unix/tkUnix3d.c, tests/bevel.tcl, tests/border.tcl, win/tkWin3d.c
3. *Built-in Bitmaps* — bitmaps/error.bmp, bitmaps/gray12.bmp, bitmaps/gray25.bmp, bitmaps/gray50.bmp, bitmaps/gray75.bmp, bitmaps/hourglass.bmp, bitmaps/info.bmp, bitmaps/questhead.bmp, bitmaps/question.bmp, bitmaps/warning.bmp, doc/GetBitmap.3, generic/tkBitmap.c, mac/tkMacBitmap.c, tests/bitmap.test
4. *Conversions From String* — doc/GetAnchor.3, doc/GetCapStyl.3, doc/GetJoinStl.3, doc/GetJustify.3, doc/GetPixels.3, doc/GetUid.3, generic/tkGet.c, tests/get.test
5. *Objects* — generic/tkObj.c, tests/obj.test
6. *Utility Functions* — doc/DrawFocHlt.3, doc/GetScroll.3, generic/tkUtil.c, tests/util.test
7. *Colormaps and Visuals* — doc/GetClrmap.3, doc/GetVisual.3, generic/tkVisual.c, tests/visual.test
8. *Color Names* — doc/GetColor.3, doc/colors.n, generic/tkColor.c, generic/tkColor.h, mac/tkMacColor.c, unix/tkUnixColor.c, tests/cmap.tcl, tests/color.test, win/tkWinColor.c, xlib/colors.c
9. *Cursor Names* — doc/GetCursor.3, doc/cursors.n, generic/tkCursor.c, mac/tkMacCursor.c, mac/tkMacCursors.r, mac/tkMacXCursors.r, unix/tkUnixCursor.c, tests/cursor.test, win/rc/cursor*.cur, win/tkWinCursor.c, xlib/X11/cursorfont.h
10. *Key Symbols* — doc/keysyms.n, mac/tkMacKeyboard.c, unix/tkUnixKey.c, win/tkWinKey.c, xlib/X11/keysym.h, xlib/X11/keysymdef.h

Standard Dialogs
1. *Generic Dialog Support* — library/comdlg.tcl
2. *[tk_chooseColor]* — doc/chooseColor.n, library/clrpick.tcl, tests/clrpick.test
3. *[tk_dialog]* — doc/dialog.n, library/dialog.tcl, mac/tkMacDialog.c, tests/winDialog.test, unix/tkUnixDialog.c, win/tkWinDialog.c
4. *[tk_chooseDirectory]* — doc/chooseDirectory.n, library/choosedir.tcl, tests/choosedir.test
5. *[tk_get*File]* — doc/getOpenFile.n, generic/tkFileFilter.c, generic/tkFileFilter.h, library/tkfbox.tcl, library/xmfbox.tcl, tests/filebox.test, tests/xmfbox.test
6. *[tk_messageBox]* — doc/messageBox.n, library/msgbox.tcl, tests/msgbox.test

Images
1. *Image Basics* — doc/CrtImgType.3, doc/DeleteImg.3, doc/GetImage.3, doc/ImgChanged.3, doc/NameOfImg.3, doc/image.n, generic/tkImage.c, generic/tkImgUtil.c, generic/tkStubImg.c, tests/image.test
2. *Bitmap Images* — doc/bitmap.n, generic/tkImgBmap.c, tests/imgBmap.test
3. *Photo Images* — doc/CrtPhImgFmt.3, doc/FindPhoto.3, doc/photo.n, generic/tkImgPhoto.c, tests/imgPhoto.test
4. *Photo Image|GIF* — generic/tkImgGIF.c
5. *Photo Image|PPM* — generic/tkImgPPM.c, tests/imgPPM.test

Fonts

1. *Generic Fonts* — doc/FontId.3, doc/GetFont.3, doc/MeasureChar.3, doc/TextLayout.3, doc/font.n, generic/tkFont.c, generic/tkFont.h, tests/font.test
2. *Mac Fonts* — mac/tkMacFont.c, tests/macFont.test
3. *Unix Fonts* — tests/unixFont.test, unix/tkUnixFont.c
4. *Win Fonts* — tests/winFont.test, win/tkWinFont.c

Geometry management

1. *Geometry Management* — doc/GeomReq.3, doc/MaintGeom.3, doc/ManageGeom.3, generic/tkGeometry.c, tests/geometry.test
2. *[grid]* — doc/grid.n, generic/tkGrid.c, tests/grid.test
3. *[pack]* — doc/pack-old.n, doc/pack.n, generic/tkPack.c, tests/oldpack.test, tests/pack.test
4. *[place]* — doc/place.n, generic/tkPlace.c, tests/place.test

Selection and Clipboard

1. *[clipboard]* — doc/Clipboard.3, doc/clipboard.n, generic/tkClipboard.c, mac/tkMacClipboard.c, tests/clipboard.test, tests/unixSelect.test, tests/winClipboard.test, unix/tkUnixSelect.c, win/tkWinClipboard.c
2. *[selection]* — doc/ClrSelect.3, doc/CrtSelHdlr.3, doc/GetSelect.3, doc/OwnSelect.3, doc/selection.n, generic/tkSelect.c, generic/tkSelect.h, tests/select.test

Other Tk commands

1. *[console]* — doc/console.n generic/tkConsole.c, library/console.tcl
2. *[focus]* — doc/focus.n, generic/tkFocus.c, tests/focus.test
3. *[grab]* — doc/Grab.3, doc/grab.n, generic/tkGrab.c, tests/grab.test
4. *[option]* — doc/AddOption.3, doc/GetOption.3, doc/option.n, generic/tkOption.c, tests/option.file1, tests/option.file2, tests/option.test
5. *[send]* — doc/SetAppName.3, doc/send.n, mac/tkMacSend.c, tests/send.test, tests/unixSend.test, tests/winSend.test, unix/tkUnixSend.c, win/tkWinSend.c
6. *[tk_focus*]* — doc/focusNext.n, library/focus.tcl, tests/focusTcl.test
7. *[tk_setPalette]* — doc/palette.n, library/palette.tcl
8. *Safe Tk* — doc/loadTk.n, library/safetk.tcl, tests/safe.test

Low-level Tk functions

1. *Geometry Functions* — generic/tkTrig.c
2. *Tk_Win Functions* — doc/ConfigWind.3, doc/CrtWindow.3, doc/IdToWindow.3, doc/MainWin.3, doc/MapWindow.3, doc/Name.3, doc/Restack.3, doc/SetClass.3, doc/SetClassProcs.3, doc/SetVisual.3, doc/StrictMotif.3, doc/Tk_Init.3, doc/WindowId.3, generic/tkWindow.c, tests/window.test
3. *Graphic Contexts* — doc/GetGC.3, generic/tkGC.c
4. *Generic Window Operations* — doc/CoordToWin.3, doc/FreeXId.3, doc/GetHINSTANCE.3, doc/GetHWND.3, doc/GetPixmap.3, doc/GetRootCrd.3, doc/GetVRoot.3, doc/HWNDToWindow.3, doc/MoveToplev.3, doc/SetGrid.3, doc/bell.n, doc/bind.n, doc/bindtags.n, doc/destroy.n, doc/lower.n, doc/raise.n, doc/tk.n, doc/tkwait.n, doc/winfo.n, doc/wm.n, generic/tkCmds.c, generic/tkPointer.c, tests/bell.test, tests/cmds.test, tests/id.test, tests/raise.test, tests/tk.test, tests/winfo.test, xlib/xgc.c
5. *Mac Window Operations* — mac/tkMacAppearanceStubs.c, mac/tkMacDraw.c, mac/tkMacEmbed.c, mac/tkMacHLEvents.c, mac/tkMacRegion.c, mac/tkMacSubwindows.c, mac/tkMacWindowMgr.c, mac/tkMacWm.c, mac/tkMacXStubs.c, tests/macEmbed.test
6. *Unix Window Operations* — tests/unixEmbed.test, tests/unixWm.test, unix/tkUnix.c, unix/tkUnixDraw.c, unix/tkUnixEmbed.c, unix/tkUnixEvent.c, unix/tkUnixFocus.c, unix/tkUnixWm.c, unix/tkUnixXId.c
7. *Win Window Operations* — tests/WinWm.test, win/stubs.c, win/tkWinDraw.c, win/tkWinEmbed.c, win/tkWinImage.c, win/tkWinPixmap.c, win/tkWinPointer.c, win/tkWinRegion.c, win/tkWinWindow.c, win/tkWinWm.c, win/tkWinX.c
8. *Events* — doc/BindTable.3, doc/event.n, generic/tkBind.c, tests/bind.test
9. *Event Loop* — doc/CrtCmHdlr.3, doc/CrtGenHdlr.3, doc/EventHndlr.3, doc/HandleEvent.3, doc/MainLoop.3, doc/QWinEvent.3, doc/RestrictEv.3, generic/tkEvent.c, tests/event.test
10. *Error Handling* — doc/CrtErrHdlr.3, doc/tkerror.n, generic/tkError.c, library/bgerror.tcl, tests/bgerror.test
11. *Atoms* — doc/InternAtom.3, generic/tkAtom.c, xlib/X11/Xatom.h

Shells
1. *Argv Parsing* — doc/ParseArgv.3, generic/tkArgv.c
2. *Application Embedding* — doc/Tk_Main.3, generic/tkInitScript.h, generic/tkMain.c, mac/tkMacInit.c, unix/tkUnixInit.c, win/tkWin32DLL.c, win/tkWinInit.c, tests/main.test,
3. *wish* — doc/wish.1, mac/tkMacAppInit.c, unix/tkAppInit.c, win/winMain.c
4. *Mac DND Tclets* — mac/tclets.r, mac/tclets.tcl

Demonstrations
1. *Widget Tour* — library/demos/arrow.tcl, library/demos/bind.tcl, library/demos/bitmap.tcl, library/demos/button.tcl, library/demos/check.tcl, library/demos/clrpick.tcl, library/demos/colors.tcl, library/demos/cscroll.tcl, library/demos/ctext.tcl, library/demos/dialog1.tcl, library/demos/dialog2.tcl, library/demos/entry1.tcl, library/demos/entry2.tcl, library/demos/filebox.tcl, library/demos/floor.tcl, library/demos/form.tcl, library/demos/hscale.tcl, library/demos/icon.tcl, library/demos/image1.tcl, library/demos/image2.tcl, library/demos/items.tcl, library/demos/label.tcl, library/demos/menu.tcl, library/demos/menubu.tcl, library/demos/msgbox.tcl, library/demos/plot.tcl, library/demos/puzzle.tcl, library/demos/radio.tcl, library/demos/ruler.tcl, library/demos/sayings.tcl, library/demos/search.tcl, library/demos/states.tcl, library/demos/style.tcl, library/demos/text.tcl, library/demos/twind.tcl, library/demos/vscale.tcl, library/demos/widget, library/demos/images/earth.gif, library/demos/images/earthris.gif, library/demos/images/face.bmp, library/demos/images/flagdown.bmp, library/demos/images/flagup.bmp, library/demos/images/gray25.bmp, library/demos/images/letters.bmp, library/demos/images/noletter.bmp, library/demos/images/pattern.bmp, library/demos/images/tcllogo.gif, library/demos/images/teapot.ppm
2. *Square Demo* — generic/tkSquare.c, library/demos/square
3. *Other Demos* — library/demos/browse, library/demos/hello, library/demos/ixset, library/demos/rmt, library/demos/rolodex, library/demos/tcolor, library/demos/timer

Localization
1. *L10N* — library/msgs/de.msg, library/msgs/el.msg, library/msgs/en.msg, library/msgs/es.msg, library/msgs/fr.msg, library/msgs/it.msg, library/msgs/nl.msg

Release Engineering
1. *Release Notes* — README, */README, */*/README, changes, license.terms, tk4.0.ps, doc/options.n mac/bugs.doc, tests/bugs.tcl
2. *Portability* — compat/limits.h, compat/stdlib.h, compat/unistd.h
3. *X11 Emulation* — xlib/X11/X.h, xlib/X11/Xfuncproto.h, xlib/X11/Xlib.h, xlib/X11/Xutil.h, xlib/xbytes.h, xlib/xdraw.c, xlib/ximage.c, xlib/xutil.c
4. *Mac Build* — mac/MW_TkHeader.h, mac/MW_TkHeader.pch, mac/MW_TkOldImgHeader.h, mac/MW_TkTestHeader.h, mac/MW_TkTestHeader.pch, mac/tkMacApplication.r, mac/tkMacLibrary.r, mac/tkMacProjects.sea.hqx, mac/tkMacResource.r, mac/tkMacShlib.exp
5. *Unix Build* — unix/Makefile.in, unix/aclocal.m4, unix/configure.in, unix/install-sh, unix/tcl.m4, unix/tk.spec, unix/tkConfig.sh.in
6. *Win Build* — win/Makefile.in, win/aclocal.m4, win/configure.in, win/makefile.vc, win/mkd.bat, win/rc/tk.rc, win/rc/tk_base.rc, win/rc/wish.rc, win/rmd.bat, win/tcl.m4, win/tkConfig.sh.in, win/tkWin.h
7. *Test Tools* — generic/tkTest.c, mac/tkMacTest.c, win/tkWinTest.c, tests/all.tcl, tests/defs.tcl, tests/visual_bb.test
8. *Logos* — library/images/logo.eps, library/images/logo100.gif, library/images/logo64.gif, library/images/logoLarge.gif, library/images/logoMed.gif, library/images/pwrdLogo.eps, library/images/pwrdLogo100.gif, library/images/pwrdLogo150.g library/images/pwrdLogo175.gif, library/images/pwrdLogo200.gif, library/images/pwrdLogo75.gif, library/images/taiku.gif, win/rc/tk.ico, win/rc/wish.ico

23.4 Shared Files

The following files are shared by all of Tk. Any maintainer may modify them as necessary to complete changes they are making to their portion of Tk. Some of the following files define Tk's API and should be changed only with TCT approval.

- ChangeLog, doc/tkvars.n, doc/TkInitStubs.3, doc/man.macros, generic/tk.decls, generic/tk.h, generic/tkInt.decls, generic/tkInt.h, generic/tkPort.h, generic/tkStubLib.c, mac/tkMac.h, mac/tkMacInt.h, mac/tkMacPort.h, unix/tkUnixInt.h, unix/tkUnixPort.h, win/tkWinInt.h, win/tkWinPort.h

23.5 Generated Files

The following files are generated, so they don't need maintainers.

- generic/ks_names.h, generic/tkDecls.h, generic/tkIntDecls.h, generic/tkIntPlatDecls.h, generic/tkIntXlibDecls.h, generic/tkPlatDecls.h, generic/tkStubInit.c, library/demos/tclIndex library/tclIndex unix/configure, unix/mkLinks, win/configure

23.6 Platform Dependencies

In addition to the division into functional areas, responsibility for a given area can also be qualified by one or more *platform* specifiers. Some areas, like *Windows Configuration and Build Tools* are obviously platform specific, so the qualification is unnecessary. Others, like *Canvas Items*, are wholly generic. But others, like *Button*, *Scale* or *Scrollbar* contain code for all platforms.

A maintainer can sign up for one of these latter areas, but specify support for only one platform. This means that that person will be responsible for the generic code in this area, in conjunction with the other platform maintainers in this area, and the platform specific code in that area.

The point behind sharing the generic code among all the maintainers is so that any changes to the Tk visible face of the widget be designed in concert for all platforms. Therefore, it is the responsibility of a platform maintainer for one platform who is sponsering a new feature for that area to work with the other platform maintainers to ensure that the feature is implemented on all platforms. One of the strengths of Tk is its cross-platform nature, and one of the maintainer's jobs is to ensure that this continues.

Procedurally, the maintainer will be listed as *Button Widget — Macintosh*, etc. A maintainer for a given area can sign up for one or more platforms. Due to the good design of the Tk's platform dependencies, determining which files are generic, and which are platform specific is trivial. The generic ones are in the *generic* directory, the Mac ones in the *mac* directory, etc. Similarly, an area which has NO files in the mac, win, or unix directories is a generic area, and no qualifiers are needed.

23.7 Copyright

TIP #24: Tcl Maintainer Assignments

TIP #24:	**Tcl Maintainer Assignments**
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
	Kevin B KENNY ⟨kennykb@acm.org⟩
	⟨dgp@user.sourceforge.net⟩
Created:	Monday, 29th January 2001
Type:	Informative
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.21 $`
Post-History:	

Abstract

This document keeps a record of who maintains each functional area of Tcl ([TIP #16]).

24.1 Assignments

Listed below are Tcl's 52 functional units, in the same order as in [TIP #16]. See [TIP #16] for the precise definition of what code belongs to what area. The area names are changed to match the Categories in Tcl's SourceForge Bug Manager (http://sourceforge.net/bugs/?group_id=10894).

Note that an area can have more than one maintainer. When the maintenance of the entire area requires several types of expertise, it is desirable to have more than one maintainer.

In several of the areas below, there are maintainers who have volunteered to provide special expertise (for example, assistance with programming and testing for the Mac platform) to assist in maintaining an area, but who have not taken on the whole area. These maintainers are indicated by a parenthesized designation of their expertise.

For each of Tcl's functional units, the following maintainers are assigned:

1. *Notifier* — Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Kevin Kenny ⟨kennykb@acm.org⟩ (Win32, Solaris, HP-UX)
2. *Event Loops* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
3. *Timer Events* — Kevin Kenny ⟨kennykb@acm.org⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
4. *Async Events* —
5. *XT Notifier* —
6. *Time Measurement* — Kevin Kenny ⟨kennykb@acm.org⟩, Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩
7. *Variables* — Miguel Sofer ⟨mig@utdt.edu⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
8. *Environment Variables* — Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac) Jeff Hobbs ⟨JeffH@ActiveState.com⟩
9. *Linked C Variables* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
10. *Objects* — Miguel Sofer ⟨mig@utdt.edu⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
11. *Conversions from String* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
12. *ByteArray Objects* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
13. *Index Object* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
14. *List Object* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
15. *Commands A-H* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
16. *Commands I-L* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
17. *Commands M-Z* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
18. *[history]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
19. *[interp]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
20. *[namespace]* — Miguel Sofer ⟨mig@utdt.edu⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
21. *[proc] and [uplevel]* — Miguel Sofer ⟨mig@utdt.edu⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
22. *[scan]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
23. *Channel Commands* — Andreas Kupries ⟨a.kupries@westend.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
24. *Channel System* — Andreas Kupries ⟨a.kupries@westend.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
25. *Channel Transforms* — Andreas Kupries ⟨a.kupries@westend.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

26. *Channel Types* — Andreas Kupries ⟨a.kupries@westend.com⟩, Rolf Schroedter ⟨Rolf.Schroedter@dlr.de⟩ (WinSerial), Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩

27. *dde Package* — Kevin Kenny ⟨kennykb@acm.org⟩

28. *http Package* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

29. *msgcat Package* —

30. *opt Package* —

31. *registry Package* — Kevin Kenny ⟨kennykb@acm.org⟩

32. *Safe Base* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

33. *tcltest Package* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

34. *Pathname Management* — Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Vincent Darley ⟨vincentdarley@users.sourceforge.net⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

35. *File System* — Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Vincent Darley ⟨vincentdarley@users.sourceforge.net⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

36. *Init — Library — Autoload* — Don Porter ⟨dgp@users.sourceforge.net⟩, Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩

37. *Package Manager* — Don Porter ⟨dgp@users.sourceforge.net⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

38. *Dynamic Loading* — Kevin Kenny ⟨kennykb@acm.org⟩, Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

39. *Memory Allocation* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

40. *Memory Preservation* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

41. *Regexp* -

42. *UTF-8 Strings* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

43. *Parsing and Eval* — Miguel Sofer ⟨mig@utdt.edu⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

44. *Bytecode Compiler* — Miguel Sofer ⟨mig@utdt.edu⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

45. *Threading* — Andreas Kupries ⟨a.kupries@westend.com⟩, Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩

46. *Embedding Support* — Don Porter ⟨dgp@users.sourceforge.net⟩, Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩

47. *Release Notes* — Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩

48. *Portability Support* — Mo DeJong ⟨mdejong@cygnus.com⟩, Daniel Steffen ⟨das@users.sourceforge.net⟩ (Mac), Jim Ingham ⟨jingham@apple.com⟩ (Mac), Jeff Hobbs ⟨JeffH@ActiveState.com⟩

49. *Configure and Build Tools* — Mo DeJong ⟨mdejong@cygnus.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩

50. *Other Tools* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

51. *[resource]* — Daniel Steffen ⟨das@users.sourceforge.net⟩ Jim Ingham ⟨jingham@apple.com⟩

52. *Macintosh* — Daniel A. Steffen ⟨steffen@ics.mq.edu.au⟩, Jim Ingham ⟨jingham@apple.com⟩

24.2 Orphaned Categories

The following Categories in Tcl's SourceForge Bug Tracker should be mapped to new Categories corresponding to a maintained area of Tcl, when seeking the appropriate maintainer:

1. *Other* — Used for reports that span several categories. Also includes many closed old reports from before the time the current categories were established.

24.3 Sections Without Maintainers

Those sections without a maintainer are maintained by the Tcl Core Team with each change requiring TYANNOTT review.

24.4 Copyright

TIP #25: Native tk_messageBox on Macintosh

TIP #25:	**Native tk_messageBox on Macintosh**
Author:	Mats Bengtsson ⟨matben@privat.utfors.se⟩
Created:	Wednesday, 7th February 2001
Type:	Project
Tcl Version:	$8.4\alpha 2$
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.1 $`
Post-History:	

Abstract

This is a replacement for the *tk_messageBox* on the Macintosh with a native implementation which is compliant with the Appearance Manager in Mac OS 8 and later.

25.1 Rationale

The present (in 8.3.2p1 and earlier) *tk_messageBox* on the Macintosh is non-movable, and lacks many features that are required to be compliant with Mac OS 8 and later. Non-movable dialogs should be abandoned in a multitasking environment. This TIP presents a step to extend the native appearance on the Macintosh.

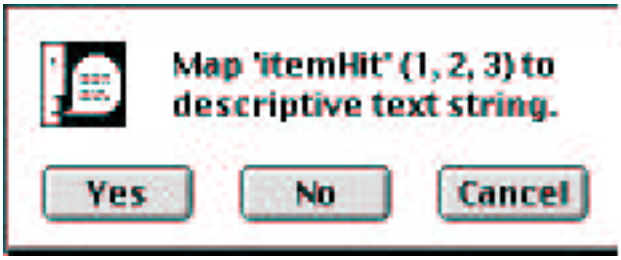

Figure 25.9: This is the present tk_messageBox.

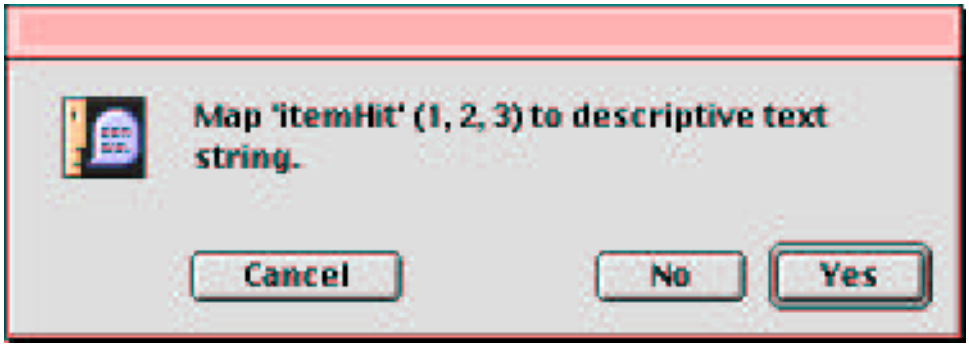

Figure 25.10: This is the native tk_messageBox.

25.2 Reference Implementation

The proposed change is now implemented as a loadable extension (in C) on Macintosh, and can be downloaded at http://hem.fyristorg.com/matben/download/MovableAlerts.sit . This extension requires Tk 8.3.2p1 or later due to the changed stub loading mechanism. The core of the code should go in the *tkMacDialog.c* file. Some additional changes are necessary in order to load the new *tk_messageBox* and not the old, script based *tk_messageBox*. Also, need to check for the presence of the Appearance manager:

```
if (Appearance Manager)
    use native (new) messageBox
else
    use present script based messageBox
```

All functionality from the documentation that are applicable are implemented, with some exceptions:

- There is a *-finemessage* option to support the native text message set in a smaller font below the main message.
- Both *-message* and *-finemessage* option are truncated at 255 characters.
- Buttons appear from right to left instead of vice versa.
- There is always a default button.

All these deviations are consistent with the look-and-feel of Mac OS 8.0 and on. Existing scripts using *tk_messageBox* are compatible with the new *tk_messageBox*.

Open questions:

- Name of extra option *-finemessage*
- Name of the two C functions in the implementation
- How to make the core code call the new code instead of the original, script implemented

25.3 Copyright

TIP #26: Enhancements for the Tk Text Widget

TIP #26:	**Enhancements for the Tk Text Widget**
Author:	Ludwig Callewaert ⟨ludwig_callewaert@frontierd.com⟩
	Ludwig Callewaert ⟨ludwig.callewaert@belgacom.net⟩
Created:	Tuesday, 20th February 2001
Type:	Project
Tcl Version:	8.4
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.5 $`
Obsoletes:	19
Post-History:	
Discussions-To:	news:comp.lang.tcl

Abstract

This TIP proposes several enhancements for the Tk text widget. An unlimited undo/redo mechanism is proposed, with several user available customisation features. Related to this, a text modified indication is proposed. This means that the user can set, query or receive a virtual event when the content of the text widget is modified. And finally a virtual event is added that is generated whenever the selection changes in the text widget.

26.1 Rationale

The text widget provides a lot of features that make it ideally suited to create a text editor from it. The vast number of editors that are based on this widget are a proof of this. Yet some basic features are missing from the text widget and need to be re-invented over and over again by the authors of the various editors. This TIP adds a number of the missing features.

A first missing feature is an undo/redo mechanism. The mechanism proposed here is simple yet powerful enough to accommodate a very reasonable undo/redo strategy. It also provides sufficient user control, so that the actual strategy can be refined and tailored to the users need.

A second missing feature is a notification if the text in the widget has been modified with respect to a reference point. [TIP #19] deals partly with this. This implementation takes it some steps further. First of all, there is a link with the undo/redo mechanism, since undoing or redoing changes can take you to or away from the reference point, and as such changes the modified state of the widget. Secondly, with this implementation, a virtual event is generated whenever the modified state of the widget changes, allowing the user to bind to that event and for instance give a visual indication of the modified state of the widget.

Finally, a virtual event has been added that is triggered whenever the selection in the widget changes. At first is may seem not so useful, but there are a number of situations where this functionality is needed. A couple of examples where I ran into the need for this may clarify this. On Windows, if the text widget does not have the focus, the selection tag is not visible. This is consistent with other Windows applications. However, when implementing a search mechanism, the found string needs to be tagged with the selection tag. (You want it to be selected). The search (and replace) dialog box has the focus however, so this selection tag is invisible. To make it visible, another tag was used to duplicate the selection tag. This is very easy when the functionality described here is available. Otherwise it is very difficult to do this consistently. Another occasion was when I was implementing a rectangular cut and paste for the text widget. This was based on adding spaces on the fly, while selecting the rectangle. If for some reason the selection changes (for instance on Unix another application gets the selection) these spaces need to be removed again. Doing this is virtually impossible without this functionality. With it, it becomes trivial. The functionality itself adds little or no overhead to the text widget.

26.2 Specification

The undo/redo mechanism operates by adding two stacks of edit actions to the text widget. Every insert or delete operation is added to the undo stack in normal operation. At certain times a semaphore (a marker if you will) is added onto the stack. All insert and delete actions in between two semaphores are considered to be one edit action, and will be undone or redone as one. The insertion of the semaphores is under user control. There is a default operation however. This will insert semaphores whenever the mode changes from insertion to deletion, or vise versa. Semaphores are also inserted when the cursor moves. By turning the autosemaphores off and inserting them at the desired points, compound actions can be creates, such as search and replace. The default paste function is an example of such an action.

Undoing an action, will re-apply in reverse order all inserts and deletes in between two semaphores. These inserts and deletes will now move to the redo stack. Redoing a change re-applies the inserts and deletes, and moves them again to the undo stack. Normal insertions or deletions will clear the redo stack.

It is also possible to clear the undo stack, giving the user control over the depth of the stack.

The modified state of the widget is implemented using a counter. Every insert or delete action increments this counter also when redone. Every undone insert or delete decrements this counter. The widget is considered to be modified if the counter is not zero. A virtual event *<<Modified>>* is generated whenever this counter changes from zero to non-zero or vice versa. A mechanism is provided to reset the counter to zero. The modified state can also be explicitely set by the user. In that case, the counter mechansim is not operational until the modified state has been reset again.

1. *pathName configure -undo 0|1* — this enables or disables the undo/redo mechanism. The default is zero.
2. *pathName configure -autosemaphores 0|1* — when one inserts semaphore automatically whenever insert changes to delete or vice versa. When off, no semaphores are inserted, except by the user (See 6). The default is one.

3. *pathName edit undo* — undoes the last edit action if undo is enabled (See 1). Raises an exception if there is nothing to undo. Does nothing otherwise.
4. *pathName edit redo* — redoes the last edit action if undo is enabled (See 1). Raises an exception if there is nothing to redo. Does nothing otherwise.
5. *pathName edit reset* — resets the undo and redo stacks (clears them).
6. *pathName edit semaphore* — inserts a semaphore (marker) on the undo stack, indicating an undo boundary. If a semaphore is already present, this will do nothing. This means that it is safe to issue the command several times, without any inserts or deletes occurring in between.
7. *pathName edit modified ?booelan?* — If boolean is not specified returns the modified state of the widget (either 1 or zero). If boolean is specified, sets the modified state of the widget to that value.
8. <<*Modified*>> — this virtual event is generated whenever the modified state of the widget changes from modified to not modified or vice versa.
9. <<*Selection*>> — this virtual event is generated whenever the range tagged with the selection tag changes.
10. <<*Undo*>> — this virtual event calls pathName edit undo. Issues a bell signal if there is nothing to undo.
11. <<*Redo*>> — this virtual event calls pathName edit redo. Issues a bell signal if there is nothing to redo.
12. <*Control-z*> — is bound to the <<Undo>> virtual event.
13. <*Control-Z*> — is bound to the <<Redo>> virtual event.

26.3 Example

The following code illustrates how the new features are intended to be used.

```
global fileName
global modState
global undoVar

set fileName "None"
set modState ""
set undoVar  0


text .t -background white -wrap none
# Example 1: The Modified event will update a text label
bind .t <<Modified>>  updateState
# Example 2: The Selection event will create a tag that
#            duplicates the selection
bind .t <<Selection>> duplicateSelection
# Pressing the return key should also mark a boundary
# in the undo stack
bind .t <Return> ".t edit semaphore"

frame .l
label .l.l -text "File: "
label .l.f -textvariable fileName
label .l.m -textvariable modState

grid .l.l -sticky w   -column 0 -row 0
grid .l.f -sticky w   -column 1 -row 0
grid .l.m -sticky e   -column 2 -row 0

grid columnconfigure .l 1 -weight 1
```

```
frame .b
button .b.l -text "Load"   -width 8 -command loadFile
button .b.s -text "Save"   -width 8 -command saveFile
button .b.i -text "Indent" -width 8 -command blockIndent

checkbutton .b.e -text "Enable Undo" -onvalue 1 -offvalue 0 -|   |   variable undoVar
trace variable undoVar w setUndo
button .b.u -text "Undo"     -width 8 -command "undo"
button .b.r -text "Redo"     -width 8 -command "redo"
button .b.m -text "Modified" -width 8 -command ".t edit modified on"

grid .b.l -row 0 -column 0
grid .b.s -row 0 -column 1
grid .b.i -row 0 -column 2
grid .b.e -row 0 -column 3
grid .b.u -row 0 -column 4
grid .b.r -row 0 -column 5
grid .b.m -row 0 -column 6

grid columnconfigure .b 0 -weight 1
grid columnconfigure .b 1 -weight 1
grid columnconfigure .b 2 -weight 1
grid columnconfigure .b 3 -weight 1
grid columnconfigure .b 4 -weight 1
grid columnconfigure .b 5 -weight 1


grid .l -sticky ew   -column 0 -row 0
grid .t -sticky news -column 0 -row 1
grid .b -sticky ew   -column 0 -row 2

grid rowconfigure    . 1 -weight 1
grid columnconfigure . 0 -weight 1



proc updateState {args} {
   global modState

   # Check the modified state and update the label
   if { [.t edit modified] } {
      set modState "Modified"
   } else {
      set modState ""
   }
}


proc setUndo {args} {
   global undoVar

   # Turn undo on or off
   if { $undoVar } {
      .t configure -undo 1
   } else {
      .t configure -undo 0
   }
}

proc undo {} {
   # edit undo throws an exception when there is nothing to
```

```
   # undo. So catch it.
   if { [catch {.t edit undo}] } {
      bell
   }
}

proc redo {} {
   # edit redo throws an exception when there is nothing to
   # undo. So catch it.
   if { [catch {.t edit redo}] } {
      bell
   }
}

proc loadFile {} {

   set file [tk_getOpenFile]
   if { ![string equal $file ""] } {
      set fileName $file
      set f [open $file r]
      set content [read $f]
      set oldUndo [.t cget -undo]

      # Turn off undo. We do not want to be able to undo
      # the loading of a file
      .t configure -undo 0
      .t delete 1.0 end
      .t insert end $content
      # Reset the modified state
      .t edit modified 0
      # Clear the undo stack
      .t edit reset
      # Set undo to the old state
      .t configure -undo $oldUndo
   }
}

proc saveFile {} {
   # The saving bit is not actually done
   # So the contents in the file are not updated

   # Saving clears the modified state
   .t edit modified 0
   # Make sure there is a semaphore on the undo stack
   # So we can get back to this point with the undo
   .t edit semaphore
}

proc blockIndent {} {
   set indent "   "

   # Block indent should be treated as one operation from
   # the undo point of view

   # if there is a selection
   if { ![catch {.t index sel.first} ] } {
      scan [.t index sel.first] "%d.%d" startline startchar
      scan [.t index sel.last]  "%d.%d" stopline  stopchar
      if { $stopchar == 0 } {
         incr stopline -1
      }
```

```
      # Get the original autosemaphores state
      set oldSema [.t cget -autosemaphores]
      # Turn of automatic insertion of semaphores
      .t configure -autosemaphores 0
      # insert a semaphore before the edit operation
      .t edit semaphore
      for {set i $startline} { $i <= $stopline} {incr i} {
         .t insert "$i.0" $indent
      }
      .t tag add sel $startline.0 "$stopline.end + 1 char"
      # insert a semaphore after the edit operation
      .t edit semaphore
      # put the autosemaphores back in their original state
      .t configure -autosemaphores $oldSema
   }
}

proc duplicateSelection {args} {
   .t tag configure dupsel -background tomato
   .t tag remove dupsel 1.0 end

   if { ![catch {.t index sel.first} ] } {
      eval .t tag add dupsel [.t tag ranges sel]
   }
}
```

26.4 Reference Implementation

http://www.cs.man.ac.uk/fellowsd-bin/TIP/26.patch

The patch has received little testing so far, so any testing is encouraged.

26.5 Copyright

TIP #27: CONST Qualification on Pointers in Tcl API's

TIP #27:	**CONST Qualification on Pointers in Tcl API's**
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
Created:	Sunday, 25th February 2001
Type:	Project
Tcl Version:	8.4
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.5 $`
Post-History:	
Discussions-To:	news:comp.lang.tcl,mailto:kennykb@acm.org

Abstract

Many of the C and C++ interfaces to the Tcl library lack a CONST qualifier on the parameters that accept pointers, even though they do not, in fact, modify the data that the pointers designate. This lack causes a persistent annoyance to C/C++ programmers. Not only is the code needed to work around this problem more verbose than required; it also can lead to compromises in type safety. This TIP proposes that the C interfaces for Tcl be revised so that functions that accept pointers to constant data have type signatures that reflect the fact. The new interfaces will remain backward-compatible with the old, except that a few must be changed to return pointers to CONST data. (Changes of this magnitude, in the past, have been routine in minor releases; the author of this TIP does not see a compelling reason to wait for Tcl 9.0 to clean up these API's.)

27.1 Rationale

When the Tcl library was originally written, the ANSI C standard had yet to be widely accepted, and the *de facto* standard language did not support a *const* qualifier. For this reason, none of the older Tcl API's that accept pointers have CONST qualifiers, even when it is known that the objects will not be modified.

In interfacing with other systems whose API's were designed after the ANSI C standard, this limitation becomes annoying. Code like:

```
const char* const string = " ... whatever ... ";
Tcl_SetStringObj( Tcl_GetObjResult( interp ),
                  (char*) string, /* Have to cast away
                                   * const-ness here
                                   * even though the string
                                   * will only be copied
                                   */
                  -1 );
```

is more verbose than necessary. It is also unsafe: the cast allows a number of unsafe type conversions (the author of this TIP has had to debug at least one extension where an integer was cast to a character pointer in this context).

In an C++ environment where engineering practice forbids using C-style cast syntax, the syntax gets even more annoying, although it provides improved safety. C++ code analogous to the above snippet looks like:

```
const char* const string = "...whatever...";
Tcl_SetStringObj( Tcl_GetObjResult( interp ),
                  const_cast< char* >( string ), -1 );
```

This code is hardly a paragon of readability.

The popular Gnu C compiler also has a problem with the *char ** declaration of so many of the parameters. With the default set of compilation options, a call like:

```
Tcl_SetStringObj( Tcl_GetObjResult( interp ),
                  "Hello world!", -1 );
```

results in an error; suppressing this message requires either using the obscure option *-fwritable-strings* on the compiler command line, or else applying awkward (and unsafe) cast syntax:

```
Tcl_SetStringObj( Tcl_GetObjResult( interp ),
                  const_cast< char* >( "Hello, world!" ), -1 );
```

Introducing CONST on parameters, however, does not bring in any incompatibility; as long as there is a prototype in scope, any ANSI-compliant compiler will implicitly cast non-CONST arguments to be type-compatible with CONST formal parameters.

27.2 Specification

This TIP proposes that, wherever possible, Tcl API's that accept pointers to constant data have their signatures in *tcl.decls* and the corresponding source files adjusted to add the CONST qualifier.

The change introduces a potential incompatibility in that code compiled on a (hypothetical) architecture where pointers to constant data have a different representation from those to non-constant data will not load against the revised stub table. This incompatibility is, in fact, not thought to be a problem, since no known port of Tcl has encountered such an architecture.

If we confine the scope of this TIP to adding CONST only to parameters, we preserve complete compatibility with existing implementations. It is neither possible nor desirable, however, to preserve drop-in compatibility across all the API's. The earliest example in the stub table is the *Tcl_PkgRequireEx* function. This function is declared to return *char **; the pointer it returns, however, is into memory managed by the Tcl library. Any attempt by

an extension to scribble on this memory or free it will result in corruption of Tcl's internal data structures; it is therefore safer and more informative to return *CONST char **. (This particular example is also highly unlikely to break any existing extension; the author of this TIP has yet to see one actually use the return value.)

Some of the API's, such as *Tcl_GetStringFromObj*, will continue to return pointers into writable memory inside the Tcl library. *Tcl_GetStringFromObj*, for instance, deals with memory that is managed co-operatively between extensions and the Tcl library; one simply must trust extensions to do the right thing (for instance, not overwrite the string representation of a shared object).

Some of the API's will not be modified, even though they appear to accept constant strings. For instance, *Tcl_Eval* modifies its string argument while it is parsing it, even though it restores its initial content when it returns. This behavior has sufficient impact on performance that it is probably not desirable to change it. The cases where the Tcl library does this sort of temporary modification, however, must be documented in the programmers' manual. They affect thread safety and positioning of data in read-only memory. One can foresee that cleaning up the API's that do not suffer from this problem will mean that programmers will be less tempted to use unsafe casts on the ones that remain.

Finally, there are a handful of API's that are essentially impossible to clean up portably; the ones that accept variable arguments come to mind. These will be left alone. One particular case in point is *Tcl_SetResult*: its third argument determines whether its second argument is constant or non-constant. In an environment without writable strings, a call like:

```
Tcl_SetResult( interp, "Hello, world!", TCL_STATIC );
```

or

```
Tcl_SetResult( interp, "Hello, world!", TCL_VOLATILE );
```

cannot be handled without unsafe casting. Fortunately, several alternatives are available. The most attractive appears to be:

```
Tcl_SetObjResult( interp,
                  Tcl_NewStringObj( "Hello, world!", -1 ) );
```

which is also more informative about what is really going on. Note that *TCL_STATIC* no longer actually carries the static pointer around. Although *Tcl_SetResult* appears to do so, as soon as the command returns, code in *tclExecute.c* converts the string result into an object result by calling *Tcl_GetObjResult*. The code using *Tcl_SetObjResult* therefore carries no greater performance cost than the original *Tcl_SetResult*.

27.3 Reference Implementation

The changes described in this TIP cut across too many functional areas to be implemented effectively all at once. Several people have pointed out that implementing this cleanup all at once appears to be necessary to avoid "CONST pollution," where the library becomes full of code that casts away the CONST qualifier. To study this issue, the author has conducted the experiment of imposing CONST strings on the first API in the stubs table: *Tcl_PkgProvideEx*.

The first concern that arose was that several other functions used the CONST strings passed as parameters, and these functions also needed to be updated. Fortunately, all were static within *tclPkg.c*. Next, when updating the documentation, the author discovered that five other functions were documented in the same man page, and shared a common defintion of the *package* and *version* parameters. They, too, were included in the change, and once again, the change was propagated forward into the functions that they called. (This activity is where the issue of replacing *Tcl_SetResult* with *Tcl_SetObjResult* was detected.)

When replacing *Tcl_SetResult* with *Tcl_SetObjResult*, the author discovered that the *file* parameter to *Tcl_DbNewStringObj* was also a constant string. With more enthusiasm than caution, he decided to attack the corresponding parameter in all the *TCL_MEM_DEBUG* interfaces. (In retrospect, it would probably have been easier to tackle this issue separately.) This change wound up cutting across virtually all of the external interfaces to *tclStringObj.c* and *tclBinary.c* and the associated documentation.

The author expects that many of the other API's will be much less closely coupled than the one studied. In particular, now that the interfaces of *tclStringObj.c* have been done once, they don't need to be done again! In fact, starting with the interfaces, like *tclStringObj.c*, that are used pervasively throughout the library and working outward would certainly have been a better course of action than tracing the dependencies forward from one function chosen almost at random.

The result of the experimental change was that twenty-eight external APIs, plus about a dozen static functions, needed to have the CONST qualifier added to at least one pointer. After these changes were made, the test suite compiled, linked, and passed all regression tests with all combinations of the NODEBUG and TCL_MEM_DEBUG options. It was nowhere necessary to cast away CONST-ness.

Possible incompatibility with existing extensions was present only in that the return values from the four functions, *Tcl_PkgPresent*, *Tcl_PkgPresentEx*, *Tcl_PkgRequire*, and *Tcl_PkgRequireEx* had the CONST qualifier added. These four functions return pointers to memory that must not be modified nor freed by the caller, so the CONST qualifier is desirable, but existing extensions may depend on storing the pointer in a variable that lacks the qualifier. This level of incompatibility in a minor release has been thought acceptable in the past; changes required to extensions are trivial, and once changed, the extensions continue to back-port cleanly to older releases.

An earlier version of these changes was uploaded to the SourceForge patch manager as patch number 404026. The revised version will be added under the same patch number as soon as the author's technical problems with uploading patches are resolved. (The major difference between the two patches is that the first patch implements the two-Stub approach described under "Rejected alternatives" below.

The success of this change has convinced the author of this TIP that the rest of the changes can be implemented in a staged manner, with little source-level incompatibility being introduced for extensions (and absolutely no incompatibility for stubs-enabled extensions compiled and linked against earlier versions of the library).

27.4 Rejected alternatives

The initial version of this TIP attempted to preserve backward compatibility of stubs-enabled extensions, even on a hypothetical architecture where pointer-to-constant and pointer-to-nonconstant have different representations.

If this level of backward compatibility is desired, it will be necessary to provide entries in the existing stub table slots corresponding to the API's that lack the CONST qualifiers.

The slots in the stub table corresponding to the non-CONST API's can be filled with wrapper functions. For example, the following function definition of *Tcl_SetStringObj_NONCONST* will use the implicit casting inherent in C to call the function with the new API.

```
void
Tcl_SetStringObj_NONCONST(Tcl_Obj* obj, /* Object to set */
                          char* bytes,  /* String value to assign */
                          int length)   /* Length of the string */
{
    Tcl_SetStringObj( obj, bytes, length );
}
```

This sort of definition is so simple that *tools/genStubs.tcl* was extended in the original patch accompanying this TIP to generate it. For example, the declaration of *Tcl_SetStringObj* that once appeared as:

```
declare 65 generic {
    void Tcl_SetStringObj( Tcl_Obj* objPtr, char* bytes, int length )
}
```

was replaced with:

```
declare 458 -nonconst 65 generic {
    void Tcl_SetStringObj( Tcl_Obj* objPtr, CONST char* bytes, int length )
}
```

declaring that slot 458 in the stubs table is to be used for the new API accepting a CONST char* for the string, while slot 65 remains used for the legacy implementation.

The difficulty with this approach, which caused it to be rejected, is that it introduces *forward* incompatibility. Any extension compiled against header files from after the change will fail to load against the stubs table from before the change. This incompatibility would require extension authors to maintain sets of header files for (at least) the earliest version of Tcl that they intend to support, rather than always being able to compile against the most current set. This problem was thought to be worse than the hypothetical and possibly non-existent problem of differing pointer representations.

27.5 Procedural note

The intent of this TIP is that, if approved, it will empower maintainers of individual modules to add *CONST* to any API where it is appropriate, provided that:

- the change does not introduce "CONST poisoning", that is, does not require type casts that remove CONST-ness;
- the documentation of the API is updated to reflect the addition of the CONST qualifier; and

Individual TIP's detailing the changes to particular APIs shall *not* be required, provided that the changes comply with these guidelines.

27.6 Change history

12 March 2001: Rejected the two-Stubs alternative and reworked the patches to use only one Stub per modified function.

27.7 Copyright

TIP #28: How to be a good maintainer for Tcl/Tk

TIP #28: **How to be a good maintainer for Tcl/Tk**	
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩ ⟨dgp@user.sourceforge.net⟩
Created:	Friday, 23rd February 2001
Type:	Informative
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.8 $`
Post-History:	

Abstract

This document presents information and advice to maintainers in the form of a Frequently Asked Questions (FAQ) list.

28.1 Preface

Notice in the header above that this is a Draft document. It won't be the *official* word of the TCT unless/until it is accepted by the TCT. Meanwhile, it should still be a helpful guide to those serving or considering service as maintainers. At the very least it's a useful straw man to revise into something better. Help us make it even more useful by using the [Edit] link at the bottom of this page (if any) to add/revise the questions and answers, or add your comments.

28.2 Background

TCT procedures (see [TIP #0]) calls for one or more *maintainers* to take responsibility for each functional area of the Tcl ([TIP #16]) or Tk ([TIP #23]) source code. Every source code patch to Tcl or Tk will be committed to the official branches of the appropriate CVS repository only after approval by an appropriate set of maintainers.

28.3 Can I be a Tcl/Tk maintainer?

Most likely. To be a maintainer, you should have...

- ...an interest in Tcl/Tk.
- ...access to the Internet (Web and e-mail).
- ...some volunteer time to contribute.
- ...the ability and the support software to code in C and/or Tcl, use CVS, use SourceForge facilities, and familiarity with a portion of the Tcl/Tk source code to be maintained, or the willingness to acquire these things.

For the most part, if you are reading this document, you probably have what it takes to be a Tcl/Tk maintainer.

28.4 What can I maintain?

The Tcl Core Team (TCT) has divided up the Tcl/Tk source code into functional areas as described in [TIP #16] and [TIP #23]. You can volunteer to help maintain as many areas as you think you can handle. Select those you have experience with or an interest in.

28.5 What does a maintainer do?

Maintainers are the people who make changes to the files that make up the source code distribution of Tcl or Tk -- code, documentation, and tests. That's what a maintainer does: check in changes to the official source code in the area he/she maintains.

The source code can be changed for several reasons: to correct a bug, to add a new feature, or to re-implement an existing feature in a new way. The reason for a change controls how much oversight the maintainer must have while making the change. More on this below.

28.6 How do I prepare to be a maintainer?

The official repositories of Tcl and Tk source code are kept at SourceForge, so you need to register for a SourceForge account (https://sourceforge.net/account/register.php). As part of the registration, you will select a login name. When you volunteer as a maintainer, the administrators of the Tcl or Tk projects will need that name to give you write access to the appropriate repository.

Once you have a SourceForge account, get familiar with the tools it provides. Most important is that you get set up to use CVS over SSH to access the repository. This can be difficult. There are some notes on how other Developers on the Tcl and Tk projects have been able to successfully get this done at http://tcltk.org/sourceforge

This document does not include instructions on how to use CVS. See the following references for assistance with learning CVS.

http://cvsbook.red-bean.com/cvsbook.html

Add more references here please.

28.7 How do I volunteer to be a maintainer?

Send a message to ⟨tcl-core@lists.sourceforge.net⟩ telling the TCT your SourceForge login name and what area(s) you want to help maintain. Someone will add you to the list of *Developers* on the Tcl or Tk projects and enable your access to SourceForge features like the Bug Tracker and Patch Manager. As a Developer, you will have write access to the appropriate repository of official source code.

28.8 Write access! So I can just start changing Tcl/Tk?!

For some purposes, yes. For others, you'll need to get approval from the TCT first. Read on...

28.9 What Internet resources does a maintainer use?

A maintainer uses the SourceForge Bug Tracker for Tcl or Tk to learn what bugs are reported in his area (browse by Category).

http://sourceforge.net/bugs/?group_id=10894

http://sourceforge.net/bugs/?group_id=12997

A maintainer uses the SourceForge Patch Manager for Tcl or Tk to learn what patches make changes in his area (browse by Category).

http://sourceforge.net/patch/?group_id=10894

http://sourceforge.net/patch/?group_id=12997

A maintainer uses CVS via SSH to access, track, and modify the various branches of development in the repository of official Tcl or Tk source code.

```
cvs -d :ext:username@cvs.tcl.sourceforge.net:/cvsroot/tcl \
   checkout -r $BRANCH_TAG -d $LOCAL_DIR tcl

cvs -d :ext:username@cvs.tktoolkit.sourceforge.net:/cvsroot/tktoolkit \
   checkout -r $BRANCH_TAG -d $LOCAL_DIR tk
```

A maintainer examines the state of Tcl Improvement Proposals (TIPs) and adds his comments to them at the TIP Document Collection.

http://dev.scriptics.com:8080/cgi-bin/tct/tip/

A maintainer may follow and participate in TCT discussions about TIPs and other matters concerning Tcl/Tk development on the TCLCORE mailing list.

http://lists.sourceforge.net/lists/listinfo/tcl-core

A maintainer may receive e-mail notification every time any change is made to any entry in Tcl's or Tk's Bug Tracker or Patch Manager by subscribing to the TCLBUGS mailing list.

http://lists.sourceforge.net/lists/listinfo/tcl-bugs

28.10 There are multiple maintainers in my area. What do I do?

The maintainer tasks are the same; you just have more hands to get the job done. It is up to the maintainers of an area to decide among themselves how they will divide the tasks. They might each take on a particular subset of files. Or they might let some maintainers fix bugs while others review new features. Or they might appoint one maintainer as the *lead* and let him assign tasks to the others. Whatever works for you, and gets the work done.

28.11 I found a bug in my area. What do I do?

Bug finding and reporting is a job for the whole community, so when you find a bug, take off your maintainer hat. Report it to the Bug Tracker just like anyone would. If you recognize that the bug is in your area, go ahead and assign it to the Category for your area and to yourself or one of the other maintainers who share responsibility for that area.

28.12 Why do I report the bug to myself?

So that the bug appears in the database. Someone else may find it too, and when they go to report it to the Bug Tracker, they should discover that it's an already reported problem. A registered bug report is also the place where progress on fixing the bug can be recorded for all to see.

28.13 There's a bug reported in the Category for the area I maintain. What do I do?

First, understand the bug report. The best bug reports are clear and come with a demonstration script, but not all reports are so well crafted. You may need to exchange messages with the person who reported the bug. If the reporter logged in to SourceForge as *username* before submitting a report, then you can write back to *username@users.sourceforge.net*. If the bug was reported by *nobody*, the best you can do is post a followup comment to the bug asking for more information, and hope the reporter comes back to check.

Next, confirm that the bug report is valid, original, and that it belongs in your area. Does it correctly assert that some public interface provided by your area behaves differently from its documented behavior? If not, then you should take the appropriate action:

1. If the bug report notes a problem in another project, assign it to a Developer who is an Admin of the other project. Add a comment asking them to reassign to the correct project. Assigned To: *an Admin of the other project*.

   If no Developer is an Admin of the other project, or the other project isn't hosted by SourceForge, note the error in a comment, and mark the report invalid. Resolution: Invalid; Status: Closed; Assigned To: *yourself*.

2. If the bug report notes a problem due to a bug in another area, reassign it to the appropriate Category. Category: *correct category*

3. If the reporter's expectations are incorrect, point them to the documentation. You may also want to revise the documentation if it is not clear. Resolution: Invalid; Status: Closed; Assigned To: *yourself*.

4. If the bug report notes a problem already noted by another bug report, note the duplication. Resolution: Duplicate; Status: Closed; Assigned To: *yourself*.

5. If the bug report acknowledges that the code is behaving as documented, but argues that the documented behavior should be revised, then the report is a feature request rather than a bug report. More on handling feature requests below. Group: Feature Request.

Valid, original bug reports in your area should be assigned to a maintainer of your area. If you are the only maintainer of your area, assign the bug to yourself. If there are multiple maintainers, you should decide among yourselves how to divide up the bug report assignments.

28.14 There's a bug assigned to me. What do I do?

Now we get the the heart of what a maintainer does. This is where you unleash the energies and talents you bring to the table. So, the best answer is "Do what works best for you." The rest of this answer should be read as additional guidelines and tips that have worked well for others and might help you, but not as a mandatory checklist you must follow. If some advice below seems more burdensome than helpful, fall back to "Do what works best for you." The goal is to register a patch that fixes the bug with the SourceForge Patch Manager. Do whatever helps you accomplish that goal.

Try to enlist the assistance of the person who reported the bug. This is especially important if the problem is platform-specific on a platform you do not have access to. Gaining the participation of the person who reported the bug can have many other benefits too. They see that progress is being made. They can offer additional insights they have, but left out of their original report. They can see how better bug reports lead to faster, better solutions, so their next reports may be of higher quality. They may even gain enough experience that their next report may come with the correction already attached. Eventually, they may even become maintainers themselves.

First, try to develop a test that demonstrates the bug and add it to the section of the test suite for your area. If the original bug report contained a demonstration script, perhaps you can adapt that. The new test will help you verify when you have fixed the bug.

If a fix for the bug is offered with the report, give it a try. Otherwise develop a fix yourself. Take care that while fixing the bug, you do not create new bugs by changing the correct behavior of other parts of the code in your section. The test suite for your area is very helpful. Use it.

It may become apparent that the best fix for your bug can only be accomplished after another bug is fixed first, or perhaps after a new feature is added to Tcl/Tk. In those cases, add a comment to the original bug report so those interested will know what is causing the delay. SourceForge may offer a way to denote these dependencies as well.

If you have trouble fixing the bug, ask for help. Try the other maintainers of your area first. Then try posting comments attached to the original bug report. Using *cvs log*, you can get a list of developers who've recently made changes to the files you maintain. They might be able to offer advice, or explanations about why the code is the way it is. If none of these focused searches for help bears fruit, then try broader requests to the TCLCORE mailing lists, or the news:comp.lang.tcl newsgroup.

At any time, you may have several bugs assigned to you. It will help guide the expectations of the Tcl community if you can assign priority values to the bugs indicating the importance you assign to them. Try to work on fixing higher priority bugs before lower priority bugs. Some reasons you might give a bug a higher priority include:

1. The bug causes a panic or core dump.
2. Documentation is missing or incorrect.
3. Other bug fixes are waiting on this bug fix.
4. Several duplicate reports or "me too" comments about the bug are coming in from the community.

Some reasons you might give a bug a lower priority include:

1. A workaround is identified (add it as a comment attached to the bug report).
2. Feature requests tend to get lower priority since they should be handled through the TIP process.

Once you have crafted a fix for the bug, create a patch to the official source code (including the new tests that test for the fixed bug) and register it with the SourceForge Patch Manager. Note the number of the bug report fixed by the patch somewhere in the summary or comments associated with the patch. Assign the patch to yourself. Assign the Category to the area you maintain.

28.15 There's a patch registered under the Category I maintain. What do I do?

The SourceForge Patch Manager is used to review and revise patches before they are committed to the official source code. Your actions depend on what the patch does to your area, and who the patch is assigned to. The patch may change the public interface provided by your area (feature change); or the change may be completely internal (bug fix, or re-implementation) within your area. The patch may be assigned to you, to someone else, or to nobody. The person the patch is assigned to is the person who is leading the effort to integrate the patch into the official source code.

28.16 What if the patch is assigned to nobody?

The patch has probably been contributed by someone not on the list of Developers. It may be a contributed bug fix, or a contributed implementation of a TIP. Assign contributed bug fixes to the same maintainer who is assigned the corresponding bug report. If there is no corresponding bug report, add one. Assign TIP implementations to the Developer identified in the TIP as the one responsible for implementation of that TIP, or the TCT member who sponsored the TIP.

If the patch changes only your area (and shared or generated files), then leave the Category in your area. If the patch changes other areas as well as yours, change the category to None.

28.17 What if the patch is assigned to me?

Presumably you've assigned it to yourself to indicate that you're taking charge of integrating that patch into the official sources. If that's a mistake, treat the patch as if it were assigned to nobody. If you are the one leading the integration effort, see below (How do I integrate a patch into the official sources?).

28.18 What if the patch is assigned to someone else?

If the patch is assigned to another maintainer in your area, let him handle it. Leave it alone.

If the patch makes no changes in your area, change the Category of the patch to None.

If the patch makes changes in your area, and is assigned to a Developer who is not a maintainer of your area, that Developer is asking for review of the patch's changes to your area. You or one of the other maintainers of your area should review the patch and accept or reject it. Read on...

28.19 What special review does a "feature change" patch require?

Changes to the public interface of your section must be proposed to and accepted by the TCT through the TIP process before they can be added to the official Tcl source code. If the patch changes the public interface of your section, then there should be an associated TIP describing the new feature(s) that patch implements. Until there is such a TIP, and that TIP has been accepted by the TCT (check the value of the State header), you should not approve the patch.

Once there is an approved TIP corresponding to the patch, you should confirm that the patch correctly implements the accepted feature as described by the TIP. If not, you should not approve the patch.

After confirming that the patch correctly implements the feature change described in an accepted TIP, you should still review the technical merit of the patch's changes to your area before approving it.

28.20 How do I review the technical merits of a patch?

Apply the patch and run the test suites that cover your area. Check that the patch does not add any new test failures. If the patch is a bug fix, check that it actually fixes the bug. Think five times before approving a patch that causes new test failures or incompletely fixes a bug or incompletely implements an approved TIP.

Keep in mind that once the patch is integrated into the official sources, you'll be expected to maintain it. It is not in your interest to approve patches that make your job harder. Think four times before approving a patch that you do not understand.

Check that the patch keeps the features offered on different platforms consistent. If not, be certain that the documentation properly notes the platform-specific behavior. Think three times before approving a patch that causes the capabilities of Tcl/Tk to further diverge on different platforms.

Check that the patch follows Tcl's established coding conventions. See the Tcl/Tk Engineering Manual (http://dev.scriptics.com/doc/engManual.pdf) and the Tcl Style Guide (http://dev.scriptics.com/doc/styleGuide.pdf) for details. This is especially important when accepting contributed patches. Think twice before approving a patch that doesn't conform to these conventions.

Check the effect of the patch on the performance of Tcl/Tk. Use the tclbench set of benchmarks.

```
cvs -d :pserver:anonymous@cvs.tcllib.sourceforge.net:/cvsroot/tcllib \
      checkout tclbench
```

Think carefully before approving a patch that significantly degrades the performance of important operations.

Finally, while examining the patch, you may see a better way to accomplish the effect of the changes in your area. If you can provide that alternative implementation reasonably quickly, then propose it as a revision to the patch. However, be careful not to let the perfect be the enemy of the good. If a patch works, do not reject just because you can imagine a better way it could be done. Provide the better way, or accept the less good way in the patch, and leave migration to the better way for later when you have the time.

To approve the patch's changes to your area, simply note your approval in a followup comment on the patch. Indicate in your comment the Category of the area for which you approve the changes. If the patch changes multiple areas, set the Category of the patch back to None.

To reject the patch, you also indicate your rejection in a followup comment. You should explain the reasons for your rejection so that the patch can be revised with the goal of gaining your approval. If you can supply the needed revisions with reasonable effort, do so. If the patch changes multiple areas, set the Category of the patch back to None.

Unless the patch is assigned to you, do not change the Status of the patch. Leave that to the Developer assigned to the patch.

28.21 How do I integrate a patch into the official sources?

First you need the approval of at least one maintainer of each section changed by the patch.

28.22 How do I get approval for integration?

First, assign the patch to yourself to indicate that you are leading the integration effort. Next, determine the list of categories corresponding to the areas changed by the patch. It may help if you list them in a comment attached to the patch.

For each category in the list, assign the Category of the patch to that category. Then wait for a maintainer for that area to review the patch. If one approves it, then assign the next Category in the list. If maintainers for all areas on the list approve the same patch, you may integrate the patch into the official sources.

If a maintainer rejects the patch, revise the patch to address his concerns. Then start the review again. Start with the maintainer who rejected the first patch to be sure his concerns are addressed first.

Note that if the patch changes only the area you maintain, then you may immediately integrate the patch into the official sources once you are satisfied with it and it is registered in the Patch Manager.

28.23 The patch is approved. How should it be integrated?

Get a CVS working directory that is up to date with the HEAD branch of the official source repository. Apply the patch to your working directory, and then 'cvs commit' the changes to the HEAD branch.

At the same time you commit the patch, be sure to add an entry to the ChangeLog file describing the change. Follow the established format, which is derived from the GNU coding conventions. The description should be brief, but should describe the change reasonably completely. Include the SourceForge Bug and Patch ID numbers in the ChangeLog entry, but do not assume that the reader will have access to the Bug Tracker and Patch Manager to be able to understand the entry. You may assume the reader has access to the documentation.

Finally, with the patch integrated, change the Status of the patch in the Patch Manager to Accepted. If any bugs were fixed by the patch, change their Resolution to Fixed, and their Status to Closed.

28.24 I want a patch review even though the patch changes only my area.

Keep in mind that integrating a patch into the official sources is not an irreversible act. Commits to the HEAD branch will be checked out and tested by members of the Tcl community who are tracking Tcl/Tk development. Alpha and beta releases of Tcl/Tk that include your patch will also get your changes reviewed in practical settings.

That said, if you really want a pre-commit review of your patch, you can add a comment to the patch asking for review. Someone will probably respond. It's up to your judgment how long to wait, keeping in mind that you are the maintainer, so your judgment on the quality of patches in your area is implicitly trusted.

28.25 What about CVS branches?

When you integrate a patch into the official source code, you will usually 'cvs commit' the patch onto the HEAD branch. If the patch includes a feature change, it must (except in unusual circumstances approved by the TCT) be committed to the HEAD branch. The HEAD branch is the development branch from which alpha releases of Tcl/Tk are generated.

At any time, there is also one or more *stable* branches of development. As of February, 2001, the branch 'core-8-3-1-branch' indicates the sequence of revisions from which the 8.3.x releases of Tcl/Tk are generated.

Since the Tcl Core Team took over development of Tcl/Tk, no changes have been committed to a stable branch, so we really have not established procedures on how we will decide what bug fixes should and should not be applied to the stable branch. It is possible that maintainers will be involved, though. It is also possible that a special team will be appointed to update the stable branch in preparation for the next stable release. In the case that you as a maintainer are asked to commit to the stable branch, be aware that the only patches that should be committed to a stable branch are those that fix bugs. No new features should be committed here.

The other kind of branch is a *feature* branch. This is a development branch on which a sequence of several revisions may be committed as work in progress on a new feature, or re-implementation of existing features. Typically a feature branch will be created if the effort...

- ...touches on several functional areas;
- ...is worked on jointly by several Developers;
- ...is complex enough to require several revisions;
- ...needs prototyping to determine the best TIP proposal to make; or
- ...makes an incompatible change to Tcl/Tk that properly belongs on the next major version of Tcl/Tk before the HEAD branch has been designated for work toward the next major version.

As a Developer, feel free to create a feature branch if you have a reason to use one. Make a note of your branch tags in [TIP #31]. Avoid the use of a branch tag matching core-* . Save the core-* branch tags for the tags of official stable branches and releases. To avoid conflict with other Developers, consider using your SourceForge login name as a prefix on the feature branch tags you create. Try to also make the branch tag descriptive of the purpose of the branch.

One big advantage of a feature branch is that any Developer may commit changes to a feature branch without all the publication, review, and approval overhead required when committing patches to the HEAD or stable branches. On the feature branches you can go through multiple revisions reasonably quickly and spend the administrative overhead only at the end when it is time to apply the finished product to the official branches.

28.26 What other things does a maintainer do?

The tasks of fixing bugs and approving and committing patches to the official source code of Tcl and Tk are the core tasks that maintainers perform. That's all the job actually requires.

You will probably want to keep an eye on the TCT's plans for Tcl/Tk development as well. If a TIP proposes a new feature in your area, it is in your interest to know about it, and propose revisions and improvements to it. Ultimately you will be asked to approve the patch that implements the new feature, and then you will be expected to maintain it, so if you have concerns about a proposal, it's best to make them known early. TCT members will probably ask your opinion on TIPs that propose changes to your area for this reason.

28.27 Comments

Please add your comments here.

> Well, since I drafted this SourceForge has replaced the Bug Tracker and Patch Manager with a *Tracker*. This TIP *really* needs revision now.

28.28 Copyright

TIP #29: Allow array syntax for Tcl lists

TIP #29:	**Allow array syntax for Tcl lists**
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Wednesday, 7th March 2001
Type:	Project
Tcl Version:	9.0
State:	Rejected
Vote:	Done
Version:	`$Revision: 1.7 $`
Post-History:	
Discussions-To:	news:comp.lang.tcl,mailto:kennykb@acm.org

Abstract

Most popular programming languages provide some sort of indexed array construct, where array subscripts are integers. Tcl's lists are, in fact, arrays, but the existing syntax obscures the fact. Moreover, the existing list commands make it difficult to manipulate lists as arrays without running into peculiar performance issues. This TIP proposes that the syntax of *variableName(value)* be extended to function as an array selector if *variableName* designates a list. This change is upward compatible with existing Tcl scripts, because the proposed syntax results in a runtime error in every extant Tcl release.

29.1 Rationale

The implementation of lists in Tcl has evolved far beyond the original conception. While lists were originally conceived to be strings with a particular syntax that allowed them to be parsed as lists, the internal representation of a list is now an array of pointers to *Tcl_Obj* structures.

Tcl programmers, for the most part, have not taken advantage of this evolution. Code that uses hash tables for the purpose is still extremely common. Moreover, it is difficult to update lists in place, even if their internal representations are known not to be shared. One example of this difficulty is seen in the discussions (http://purl.org/thecliff/tcl/wiki/941) of how best to shuffle a list of items. The discussion began with a nave implementation of Jon Bentley's method of performing random swaps:

```
proc shuffle1 { list } {
    set n [llength $list]
    for { set i 0 } { $i < $n } { incr i } {
        set j [expr {int(rand()*$n)}]
        set temp [lindex $list $j]
        set list [lreplace $list $j $j [lindex $list $i]]
        set list [lreplace $list $i $i $temp]
    }
    return $list
}
```

Aside from the fact that the syntax obscures what the program is doing, the implementation suffers from an obscure performance problem. When the *lreplace* calls in the *shuffle1* procedure are executed, the internal representation of *list* has two references: the value of the variable, and the parameter passed to *lreplace*. The multiple references force *lreplace* to copy the list, leading to quadratic performance when large lists are shuffled.

It is possible, albeit difficult, to alleviate this problem by careful management of the lifetime of *Tcl_Obj* structures, but this change complicates the code. The simplest way to fix the performance is probably to use Donal Fellows's implementation of the *K* combinator:

```
proc K { x y } { set x }
```

which allows the caller of *lreplace* to extract the value of *list*, change the value of *list* so that the extracted value is unshared, and then pass the extracted value as a parameter to *lreplace:*

```
proc shuffle1a { list } {
    set n [llength $list]
    for { set i 0 } { $i < $n } { incr i } {
        set j [expr {int(rand()*$n)}]
        set temp1 [lindex $list $j]
        set temp2 [lindex $list $i]
        set list [lreplace [K $list [set list {}]] $j $j $temp2]
        set list [lreplace [K $list [set list {}]] $i $i $temp1]
    }
    return $list
}
```

Now the performance of the code is *O(n)* where *n* is the length of the list, but the programmer's intent has been seriously obscured!

These drawbacks have led prominent individuals such as Richard Stallman (http://www.vanderburg.org/Tcl/war/0000.html) to assert that Tcl lacks arrays.

This proposal includes the absolute minimum of functionality needed to provide array-style indexing for variables containing Tcl list objects. The reason for this limitation is that omitted functionality can be added later without breaking existing scripts. On the other hand, ill-considered extensions may turn into something that we're doomed to support forever.

29.2 Specification

This TIP's proposed change can be stated succinctly:

> Wherever the notation *a(x)* may be used to refer to an array element in the language, allow it also to refer to an element of a list, provided that the variable *a* is scalar and the value *x* is an index suitable for the *lindex* command.

Exception: Traces, *unset* and *upvar* calls designating individual list elements shall not be supported. (As a consequence of this rule, list elements may also not appear as linked variables in C code, implying that they also cannot appear as *-variable* or *-textvariable* options on Tk widgets.)

Note that this change is backward compatible with existing Tcl scripts! If a notation like *a(x)* is used to refer to a scalar variable in today's Tcl, the result is an error:

```
% set a [list foo bar grill]
foo bar grill
% set a(2)
can't read "a(2)": variable isn't array
% puts $a(2)
can't read "a(2)": variable isn't array
% set a(2) zot
can't set "a(2)": variable isn't array
```

The default behavior, if *a* is not set, and a script executes

```
set a(2) zot
```

will still be to create an associative array. If a script wishes to perform such actions on a list, it will be necessary first to initialize the variable:

```
set a [list]
set a(0) foo
```

Note that in the example above, there is no requirement that the internal representation of *a* be a list; the line,

```
set a [list]
```

could have been replaced with

```
set a {}
```

with the only impact being the run-time cost of shimmering the empty string into an empty list. Nowhere does this proposal introduce behavior that depends on a specific internal representation for any variable.

This proposal the syntax of the subscript shall be precisely those values that are accepted as the second argument to the *lindex* command. In other words, the subscript may be an integer *N*, or the string *end* or *end-N*. The value of *N* may not be less than zero nor greater than nor equal to the length of the list on any usage that reads a list element.

A usage that writes a list element may use an integer equal to the length of the list, or the string *end+1*, to designate the element one past the end. In other words,

```
  set a(end+1) foo
```

will have the same effect as:

```
  lappend a foo
```

With the proposed change in syntax, the procedure to shuffle a list becomes much more straightforward:

```
  proc shuffle1 { list } {
      set n [llength $list]
```

```
    for { set i 0 } { $i < $n } { incr i } {
        set j [expr {int(rand()*$n)}]
        set temp $list($j)
        set list($j) $list($i)
        set list($i) $temp
    }
    return $list
}
```

The given implementation copies the list only once, the first time that the line:

```
        set list($j) $list($i)
```

is executed. Thereafter, the list is an unshared object, and the replacements are performed in place.

It shall be illegal to pass a list element as the parameter to *upvar;* that is, the following usage:

```
proc increment { varName } {
    upvar 1 $varName v
    incr v
}
set x [list 1 2 3]
increment x(0)
```

will *not* be supported. However, the commoner form:

```
proc incrementElement { arrayName index } {
    upvar 1 $arrayName array
    incr array($index)
}
set x [list 1 2 3]
incrementElement x 0
```

will, of course, work as expected.

29.3 Discussion

Several reviewers expressed concern about the reuse of array syntax. In particular, the alternative syntax $a<$element> was proposed repeatedly. Alas, there is no good alternative syntax that will not break at least some existing scripts. The proposed syntax using angle-brackets is a poor choice, because Tcl scripts that generate Web pages frequently have code like:

```
puts "<$tag>$text</$tag>
```

that would be broken horribly by such a change.

There are several obvious extensions to the proposal that are not addressed, and these omissions are intentional.

- The proposal makes no attempt to deal with multiple subscripts as a means of accessing nested lists.

Use of multiple subscripts is closely related to the withdrawn [TIP #22] (which the author of this TIP intends to revive). If the related TIP is accepted, the syntax for the subscript could readily be expanded so that it could be a Tcl list giving the subscripts in lexicographic sequence. For example

```
set a(2 3) foo
```

could be used to set the fourth element of the third sublist.

- The proposal allows the *set* command (or any other use of *Tcl_SetVar2Ex*) to set only the elements that are in the list already plus the one one beyond the end.

Tcl lists are fundamentally dense arrays. Allowing non-contiguous elements, that is, sparse arrays, is a fundamental change to their semantics. Such a change is not contemplated at this time.

- The proposal does not allow the *unset* command (or any other command that arrives at *Tcl_UnsetVar2*) to delete members of a list.

Earlier versions of the proposal had proposed to permit:

```
unset a([expr { [llength $a] - 1}])
```

or equivalently:

```
unset a(end)
```

to reduce the length of the list by one. In subsequent discussions, the reviewers found it distasteful that the proposed syntax did not permit unsetting interior elements of a list. Alas, the discussion did not arrive at a consensus on what the precise semantics of such an operation ought to be. Some reviewers favored attempting to emulate sparse arrays (again, a fundamental change to the semantics of Tcl lists that is not contemplated at this time). Others preferred the semantics of shifting the remaining elements, so that

```
unset a($n)
```

would always be equivalent to

```
set a [lreplace $a $n $n]
```

except for performance. Both camps found it overly restrictive to limit the semantics of *unset* to those of the original proposal. Because the two groups failed to achieve a consensus, the author of this TIP finds it prudent to forbid *unset* altogether in the initial implementation.

- The *array* command continues to operate only on associative arrays.

Lists are a simple enough structure that the full power of the *array* command is not required to deal with them, and having it work on lists as well as arrays seems like needless effort. Moreover, existing code may well depend on a combination of *[array exists]* and *[info exists]* to distinguish associative arrays from scalar variables (including lists).

- The *upvar* command cannot address individual list elements.

Extending the syntax in this fashion would make *upvar* more consistent in its behavior, but appears to be expensive, in terms of both performance (tracking down the linked references if a list is rebuilt) and the effort required for implementation (the author of this TIP is unlikely to have the time required to implement the necessary changes to *struct Var* and the associated code).

- No traces on list elements shall be supported. List elements cannot function as linked variables in C code.

The original proposal had specified how write and unset, but not read, traces could be implemented. The original proposed functionality is described in the Appendix. The author of this TIP had proposed it primarily so that list elements could function as linked variables (for instance, in the *-variable* and *-textvariable* options of Tk widgets).

Once again, this part of the original proposal failed for lack of consensus among the reviewers. Some felt that supporting read traces in one context but not another would be overly confusing. Moreover, the proposal as written would cause write traces on the elements to fire if the internal representation of a variable shimmered between a list and something else. Some reviewers found the excess trace callbacks to be objectionable.

At least one reviewer proposed a separate *trace add element* syntax for list-element traces. This syntax would address some of the concerns about the lack of read traces (there's no reason that *trace add element* should function the same as *trace add variable*). Alas, it would not address the problem of linked variables, which was the main reason for having the traces in the first place.

Given the lack of consensus, the author of this TIP finds it prudent to withdraw or postpone this portion of the proposal.

29.4 See Also

[TIP #22] — withdrawn.

29.5 Reference Implementation

No reference implementation has yet been developed; the author of this TIP wishes to solicit the opinions of the Tcl community before spending a lot of time implementing a possibly bad idea.

29.6 Change history

12 March 2001: Added detailed discussion of the specific subscript ranges supported by read, write and unset operations. Changed the discussion to reject the alternative of padding an array when setting an index beyond the end. Added discussion of the details of write and unset traces, and rejecting read traces as being infeasible to implement. Clarified the example of creating an empty list so as to avoid any misapprehension that these changes depend on list variables' having a particular representation at any given time; in fact, every detail of this proposal is tolerant of shimmering.

13 March 2001: Fixed a copy-and-paste error in the 'incrementElement' example, and added to the discussion the fact that all operations will throw errors in the event of a malformed list.

30 March 2001: Revised yet again, in an attempt to remove as much controversial functionality as possible and reduce the TIP to the minimum useful subset, on the grounds that it is prudent to avoid supporting functionality that may later prove ill-considered.

29.7 Summary of objections

DeJong, English (non-voting), *Flynt* (non-voting), *Harrison, Ingham, Lehenbauer, Polster,* (non-voting), *Porter,* and *Sofer* (non-voting), expressed concern that the proposed syntax is confusing, since the target object could be either an associative array or a linear array (that is, a Tcl list). These objections varied in stridency from "yes, it is a risk, and I'm prepared to accept it," to "this will just be too confusing, and I can't countenance this proposal."

Hobbs found the original proposal's omission of reverse indexing distasteful. The current version of the proposal embraces his suggested change.

Cuthbert (non-voting), *Hobbs,* and *Porter* expressed concern over the semantics of *unset.* Since consensus was not achieved, the current version of the proposal defers implementation of *unset.*

Several reviewers, most notably *Ousterhout,* found the proposed *trace* semantics distasteful. The current version of the proposal eliminates *trace* on list elements.

Several reviewers appeared to labor under the misconception that this TIP introduces behavior that is dependent at run time upon the internal representation of a Tcl object. It does not; it is tolerant of shimmering in all cases.

Several reviewers objected to the proposal on the grounds that it does not specify a general object system and how such a system would allow for generic containers with array syntax. The author's intention in writing it was not to propose such a system, but only to propose a small piece of syntactic sugar, implementable here and now, that is compatible with that broader vision.

29.8 Appendix: Possible implementation of read and unset traces.

The original proposal contained the following language, which could be used as a guide if traces on list elements are contemplated at a future time.

Write and unset traces on list elements shall be supported; it shall be permissible to write:

```
trace add variable x(1) write writeCallback
```

or

```
trace add variable x(1) unset unsetCallback
```

The *write* callback shall be invoked whenever the given list element changes value; the *unset* callback shall be invoked whenever the variable is unset or when its length shrinks to the point that it no longer has a member with the given index.

Read traces on list elements shall *not* be supported. It is too difficult at this point to define what their semantics should be. For instance, if a program executes the following code:

```
trace add variable x(0) read readCallback
set x [list foo bar grill]
set y [string range $x 4 end]
```

should the callback fire? By one argument, the program has not read element zero of the list; by another, using the list as a string has read every element, and all read traces should fire. In any case, the read trace on a variable fires before its usage is known; it appears impossible in existing code to implement selective read tracing on list elements.

The implementation of write and unset traces on list elements will be done by establishing a C-level write trace on the variable as a whole. The client data of the trace will designate a structure containing the ordinal number of the element being traced, and a *Tcl_Obj* pointer designating its old value. The reference count of the *Tcl_Obj* will be incremented when this pointer is stored. Note that this increment operation makes the object shared. Any change to the designated element will thus need to copy the object.

When the write trace fires, the list representation of the variable will be extracted, reconstituting it from the string representation if necessary. If extracting the list representation fails, the trace will be considered to have failed as well, and the trace callback will return *TCL_ERROR*. If extracting the list representation succeeds, the list length will be compared with the ordinal number of the element being traced. If the element number is no longer within the list, an unset trace fires if one exists. If the element number is within the list, the two *Tcl_Obj* pointers are compared. If they are identical, the list element in question is unchanged, and nothing need be done. Otherwise, the write trace fires.

This behavior is conservative in that an operation that spoils the list representation of the object is considered to have written every element of the list. This rule is consistent with the rule that write traces on ordinary Tcl variables fire whenever the variable is set, even if it is being set to an identical value.

In any event, after the conclusion of a trace callback, the saved Tcl_Obj will have its reference count decremented and be replaced with the current element of the list (with reference count appropriately incremented, of course).

29.9 Copyright

TIP #30: Tk Toolkit Maintainer Assignments

TIP #30:	**Tk Toolkit Maintainer Assignments**
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩
	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
	Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
	Todd M. Helfter ⟨tmh@purdue.edu⟩
	Chengye Mao ⟨chengye.geo@yahoo.com⟩
	George B. Smith ⟨gbs@k9haven.com⟩
	Miguel Ban ⟨bagnonm@safelayer.com⟩
Created:	Friday, 9th March 2001
Type:	Informative
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.21 $`
Post-History:	

Abstract

This document keeps a record of who maintains each functional area of Tk ([TIP #23]).

30.1 Assignments

Listed below are Tk's 86 functional units, in the same order as in [TIP #23]. See [TIP #23] for the precise definition of what code belongs to what area, and how maintainers designate their support for platform-specific portions of the code. The area names listed below are also the Categories in the SourceForge Tracker for the Tk Toolkit (http://sourceforge.net/tracker/?group_id=12997).

For each of Tk's functional units, the following maintainers are assigned:

1. *Bindings* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
2. *Appearance* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
3. *[*button] and [label]* — Allen Flick ⟨allenflick@home.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
4. *Canvas Basics* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
5. *Canvas Items* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
6. *Canvas PostScript* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
7. *[entry]* — Allen Flick ⟨allenflick@home.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
8. *[frame] and [toplevel]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Peter Spjuth ⟨peter.spjuth@space.se⟩
9. *[listbox]* — Allen Flick ⟨allenflick@home.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
10. *Generic Menus* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Todd Helfter ⟨tmh@purdue.edu⟩
11. *Mac Menus* — George B. Smith ⟨gbs@k9haven.com⟩
12. *Unix Menus* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Todd Helfter ⟨tmh@purdue.edu⟩
13. *Win Menus* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Todd Helfter ⟨tmh@purdue.edu⟩
14. *[message]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
15. *[scale]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
16. *[scrollbar]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
17. *[spinbox]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
18. *[text]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
19. *Menubars (obsolete)* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
20. *[tk_optionMenu]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
21. *Option Parsing* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
22. *Relief* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Frdric Bonnet ⟨fredericbonnet@free.fr⟩
23. *Built-in Bitmaps* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
24. *Conversions From String* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
25. *Objects* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
26. *Utility Functions* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
27. *Colormaps and Visuals* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
28. *Color Names* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
29. *Cursor Names* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

30. *Key Symbols* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
31. *Generic Dialog Support* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
32. *[tk_chooseColor]* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ *(Unix)*, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
33. *[tk_dialog]* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ *(Unix)*, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
34. *[tk_chooseDirectory]* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ *(Unix)*, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
35. *[tk_get*File]* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ *(Unix)*, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
36. *[tk_messageBox]* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩ *(Unix)*, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
37. *Image Basics* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
38. *Bitmap Images* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Kevin Griffin ⟨vertov@artstar.com⟩
39. *Photo Images* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
40. *Photo Image|GIF* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
41. *Photo Image|PPM* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
42. *Generic Fonts* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
43. *Mac Fonts* — George B. Smith ⟨gbs@k9haven.com⟩
44. *Unix Fonts* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
45. *Win Fonts* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
46. *Geometry Management* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Chengye Mao ⟨chengye.geo@yahoo.com⟩
47. *[grid]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
48. *[pack]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
49. *[place]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
50. *[clipboard]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩ Joe English ⟨jenglish@flightlab.com⟩ (Unix)
51. *[selection]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Joe English ⟨jenglish@flightlab.com⟩ (Unix)
52. *[console]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Chengye Mao ⟨chengye.geo@yahoo.com⟩
53. *[focus]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
54. *[grab]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
55. *[option]* — Allen Flick ⟨allenflick@home.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
56. *[send]* — Allen Flick ⟨allenflick@home.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
57. *[tk_focus*]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
58. *[tk_setPalette]* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
59. *Safe Tk* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
60. *Geometry Functions* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Chengye Mao ⟨chengye.geo@yahoo.com⟩
61. *Tk_Win Functions* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
62. *Graphic Contexts* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
63. *Generic Window Operations* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
64. *Mac Window Operations* — George B. Smith ⟨gbs@k9haven.com⟩

65. *Unix Window Operations* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Joe English ⟨jenglish@flightlab.com⟩
66. *Win Window Operations* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Chengye Mao ⟨chengye.geo@yahoo.com⟩
67. *Events* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
68. *Event Loop* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Jan Nijtmans ⟨j.nijtmans@chello.nl⟩
69. *Error Handling* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
70. *Atoms* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
71. *Argv Parsing* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
72. *Application Embedding* -
73. *wish* -
74. *Mac DND Tclets* -
75. *Widget Tour* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
76. *Square Demo* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
77. *Other Demos* — Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
78. *L10N* — Jan Nijtmans ⟨j.nijtmans@chello.nl⟩, Miguel Ban ⟨bagnonm@safelayer.com⟩
79. *Release Notes* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
80. *Portability* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
81. *X11 Emulation* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩
82. *Mac Build* — George B. Smith ⟨gbs@k9haven.com⟩
83. *Unix Build* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Mo DeJong ⟨mdejong@cygnus.com⟩
84. *Win Build* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩, Mo DeJong ⟨mdejong@cygnus.com⟩
85. *Test Tools* — Allen Flick ⟨allenflick@home.com⟩, Jeff Hobbs ⟨JeffH@ActiveState.com⟩
86. *Logos* — Jeff Hobbs ⟨JeffH@ActiveState.com⟩

30.2 General Categories

The following categories in Tk's SourceForge Tracker do not refer to any specific portion of Tk. Reports in these categories should be mapped to categories corresponding to a maintained area of Tk, when seeking the appropriate maintainer:

1. *Other* — Reports that span multiple categories.

30.3 Areas Without Maintainers

Those funcational areas without a maintainer are maintained by the Tcl Core Team with each change requiring TYANNOTT review.

30.4 Copyright

TIP #31: CVS tags in the Tcl and Tk repositories

TIP #31:	**CVS tags in the Tcl and Tk repositories**
Author:	Don Porter ⟨dgp@users.sourceforge.net⟩
	miguel sofer ⟨mig@utdt.edu⟩
	Jeff Hobbs ⟨JeffH@ActiveState.com⟩
	Kevin Kenny ⟨kennykb@acm.org⟩
Created:	Monday, 12th March 2001
Type:	Informative
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.5 $`
Post-History:	

Abstract

This document keeps a record of the CVS tags used in the Tcl and Tk repositories and their meanings.

31.1 Background

CVS uses tags to collectively label a particular set of revisions of a particular set of files. With a tag, one may easily request all the revisions of all the files that correspond to something meaningful, such as an official release of a project.

There are two kinds of tags provided by CVS. First is the release tag that simply marks a set of revisions as belonging together as a unit. Each release of a project should be tagged with a release tag. Other development milestones may also receive a release tag. Release tags are useful for marking any point in development that will be useful to return to or compare against.

The second kind of tag is a branch tag. It does not mark a single revision of a file, but an entire branch of development of a file. Branch tags are the means by which different working directories can track different branches of development.

A tag may be used in a CVS repository only once, so we must keep track of what tags have already been used, and what they mean. The remaining sections of this TIP record the tags in use. This TIP should be kept up to date by adding any new tags here as they are added to the CVS repository.

31.2 Release Tags

The following tags in the Tcl and Tk CVS repositories correspond to the following releases of Tcl/Tk:

- core-8-3-3 — Tcl/Tk 8.3.3
- core-8-4-a2 — Tcl/Tk 8.4a2
- core-8-4-a1 — Tcl/Tk 8.4a1
- core-8-3-2 — Tcl/Tk 8.3.2
- core-8-3-1 — Tcl/Tk 8.3.1
- core-8-3-0 — Tcl/Tk 8.3.0
- core-8-3-b2 — Tcl/Tk 8.3b2
- core-8-3-b1 — Tcl/Tk 8.3b1
- core-8-2-3 — Tcl/Tk 8.2.3
- core-8-2-2 — Tcl/Tk 8.2.2
- core-8-2-1 — Tcl/Tk 8.2.1
- core-8-2-0 — Tcl/Tk 8.2.0
- core-8-2-b3 — Tcl/Tk 8.2b3
- core-8-2-b2 — Tcl/Tk 8.2b2
- core-8-2-b1 — Tcl/Tk 8.2b1
- core-8-1-1 — Tcl/Tk 8.1.1
- core-8-1-0 — Tcl/Tk 8.1.0
- core-8-1-b3 — Tcl/Tk 8.1b3
- core-8-1-b2 — Tcl/Tk 8.1b2
- core-8-1-b1 — Tcl/Tk 8.1b1
- core-8-0-5 — Tcl/Tk 8.0.5

- core-8-0-4 — Tcl/Tk 8.0.4
- core-8-0-3 — Tcl/Tk 8.0.3
- core-8-0-2 — Tcl/Tk 8.0p2

31.3 Branch Tags — Official Development

The following branch tags label branches of development from which releases of Tcl/Tk are generated:

- HEAD — current development of new features; spawns 8.4aX releases.
- core-8-3-1-branch — bug fix branch; spawns 8.3.X releases.

31.4 Branch Tags — Features

The following branch tags label branches on which features are being devloped and tested. No releases of Tcl/Tk will be spawned from these branches. As the features mature, they will be merged onto the HEAD branch, or they may be rejected.

- core-8-4-win-speedup (Tk) — Work on improving performance of Tk on the Windows platforms.
- dgp-privates-into-namespace (Tk) — Work on moving Tk's private commands and variables into the ::tk namespace and its children.
- kennykb-tip-22-33 (Tcl) — Work on implementing the changes described in TIP's #22 and #33.
- msofer-bcEngine (Tcl) — Work on improving performance of the bytecode engine.

31.5 Dead Branches

The following branch tags label branches that are no longer being developed. Some are old official branches from which releases are no longer being spawned. Others are feature development branches that have been merged into an official branch, or rejected.

- core-8-3-1-io-rewrite (Tcl) — Work rewriting Tcl's IO Channels to correct problems with the implementation of stacked channels. Merged into Tcl 8.3.2 and Tcl 8.4a2.
- core-8-2-1-branch — Spawned Tcl/Tk 8.2.X releases.
- core-8-1-branch-old — Spawned Tcl/Tk 8.1bX releases.
- dev-stubs-branch, dev-8-1-stubs-branch — Two branches on which the stubs interfaces were developed. Merged into Tcl 8.1.

31.6 Copyright

TIP #32: Add Tcl_Obj support to traces

TIP #32:	**Add Tcl_Obj support to traces**
Author:	David Cuthbert ⟨dacut@kanga.org⟩
	Kevin Kenny ⟨kennykb@acm.org⟩
Created:	Friday, 23rd March 2001
Type:	Project
Tcl Version:	$8.4\alpha2$
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.3 $`
Keywords:	trace, Tcl_Obj
Post-History:	
Discussions-To:	news:comp.lang.tcl

Abstract

This document proposes to add Tcl_Obj support for trace procedures written in C.

32.1 Rationale

The *Tcl_Obj* system was introduced in version 8.0, making computations (potentially) much more efficient by eliminating many type conversions to and from strings. However, the trace API continues to require character strings in both command and variable traces.

32.2 Specification

Add the following functions to the Tcl core:

- Tcl_Trace *Tcl_CreateObjTrace*(interp, level, objProc, clientData)

  Tcl_CreateObjTrace behaves in the same manner as *Tcl_CreateTrace*, except the trace procedure (objProc) should have arguments and result that match type type *Tcl_CmdObjTraceProc*:

  ```
  typedef void Tcl_CmdObjTraceProc(
      ClientData clientData,
      Tcl_Interp *interp,
      int level,
      char *command,
      Tcl_ObjCmdProc *cmdProc,
      ClientData cmdClientData,
      int objc,
      Tcl_Obj * CONST objv[] );
  ```

  Trace tokens returned by *Tcl_CreateObjTrace* can be used in *Tcl_DeleteTrace* to remove the trace.

- int *Tcl_ObjTraceVar2*(interp, part1Ptr, part2Ptr, flags, objProc, clientData)

  Tcl_ObjTraceVar2 behaves in the same manner as *Tcl_TraceVar2*, except the variable name is passed as Tcl_Obj pointers (in the same manner as *Tcl_ObjSetVar2*, q.v.), and the trace procedure (objProc) should have arguments and result that match the type *Tcl_VarObjTraceProc*:

  ```
  typedef Tcl_Obj *Tcl_VarObjTraceProc(
      ClientData clientData,
      Tcl_Interp *interp,
      Tcl_Obj *part1Ptr,
      Tcl_Obj *part2Ptr,
      int flags );
  ```

  Under normal conditions, the trace procedure should return NULL, indicating successful completion. If objProc returns a value other than NULL it signifies that an error occurred. Upon return, the reference count of the Tcl_Obj should be at least one; ownership of this reference is transferred to the Tcl interpreter.

- void *Tcl_ObjUntraceVar2*(interp, part1Ptr, part2Ptr, flags, objProc, clientData)

  Tcl_ObjUntraceVar2 behaves in the same manner as *Tcl_UntraceVar2*, except it is used to remove trace procedures registered with *Tcl_ObjTraceVar2*.

- ClientData *Tcl_ObjVarTraceInfo2*(interp, part1Ptr, part2Ptr, flags, objProc, prevClientData)

  Tcl_ObjVarTraceInfo2 behaves in the same manner as *Tcl_VarTraceInfo2*, except it is used to iterate through trace procedures registered with *Tcl_ObjTraceVar2*.

32.3 Change History

30 March 2001 — Changed return value of objProc to a Tcl_Obj * instead of int (and using the interpreter result to indicate an error). This is more consistent with the current behavior (but without the bug). -dac

32.4 See Also

Tcl manual pages *Tcl_TraceVar* and *Tcl_CreateTrace*.

32.5 Copyright

32.6 Comments

Kevin Kenny (2 April 2001):

This proposal is detailing functionality that I've wanted for quite some time. Given, however, that it allows us to make a partial break with the past, I'd like to make some minor changes to *Tcl_CmdObjTraceProc.*

In place of the type signature,

```
typedef void Tcl_CmdObjTraceProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int level,
    char *command,
    Tcl_ObjCmdProc *cmdProc,
    ClientData cmdClientData,
    int objc,
    Tcl_Obj * CONST objv[] );
```

may I suggest that since the interpreter has the *Command* structure in hand, it simply deliver a *Tcl_Command* with the command's information, in place of the command procedure and client data? Also, the command name is redundant, since the same information is present in *objv[0]*.

The signature would then be:

```
typedef void Tcl_CmdObjTraceProc(
    ClientData clientData,  /* Client data from Tcl_CreateObjTrace */
    Tcl_Interp* interp,     /* Tcl interpreter */
    int level,              /* Execution level */
    Tcl_Command cmdInfo,    /* Command information */
    int objc,               /* Parameter count */
    Tcl_Obj *CONST objv[]   /* Parameter vector */
);
```

This would allow the trace procedure to do interesting things like replace the command's *objCmdProc* and client data temporarily, before the interpreter uses them. I have a profiler that works that way, using the existing API's. It's awkward at the moment, because it needs to use *Tcl_FindCommand* to get at the command object (*Tcl_GetCommandInfo* would also work in current releases, but I'm in the position of needing bugward compatibility with 8.0). It also is a horrible performance drain because of the shimmering that's needed to support tracing currently, and the fact that tracing defeats the bytecode compiler.

If this change gets approved, and I can get *TclpGetTime* exported, I'll definitely release the profiler. (I don't care to release code that depends on tclInt.h, because I don't want to track APIs that the maintainers don't consider 'stable'.)

By the way, this change should be easier from a political standpoint than it was a year ago, when any extension that used this mechanism was presumably a competitor of the TclPro tools.

TIP #33: Add 'lset' Command to Assign to List Elements.

TIP #33:	**Add 'lset' Command to Assign to List Elements.**
Author:	Kevin Kenny ⟨kennykb@acm.org⟩
Created:	Tuesday, 15th May 2001
Type:	Project
Tcl Version:	8.4
State:	Accepted
Vote:	Done
Version:	`$Revision: 1.11 $`
Post-History:	
Discussions-To:	news:comp.lang.tcl,mailto:kennykb@acm.org

Abstract

Most popular programming languages provide some sort of indexed array construct, where array subscripts are integers. Tcl's lists are implemented internally as indexed arrays, but it is difficult to use them as such because there is no convenient way to assign to individual elements. This TIP proposes a new command, *lset*, to rectify this limitation.

33.1 Rationale

The implementation of lists in Tcl has evolved far beyond the original conception. While lists were originally conceived to be strings with a particular syntax that allowed them to be parsed as lists, the internal representation of a list is now an array of pointers to *Tcl_Obj* structures.

Tcl programmers, for the most part, have not taken advantage of this evolution. Code that uses hash tables where linear arrays would be a more appropriate structure is still extremely common. Moreover, it is difficult to update lists in place, even if their internal representations are known not to be shared. One example of this difficulty is seen in the discussions (http://purl.org/thecliff/tcl/wiki/941) of how best to shuffle a list of items. The discussion began with a nave implementation of Jon Bentley's method of performing random swaps:

```
proc shuffle1 { list } {
    set n [llength $list]
    for { set i 0 } { $i < $n } { incr i } {
        set j [expr {int(rand()*$n)}]
        set temp [lindex $list $j]
        set list [lreplace $list $j $j [lindex $list $i]]
        set list [lreplace $list $i $i $temp]
    }
    return $list
}
```

Aside from the fact that the syntax obscures what the program is doing, the implementation suffers from an obscure performance problem. When the *lreplace* calls in the *shuffle1* procedure are executed, the internal representation of *list* has two references: the value of the variable, and the parameter passed to *lreplace*. The multiple references force *lreplace* to copy the list, leading to quadratic performance when large lists are shuffled.

It is possible, albeit difficult, to alleviate this problem by careful management of the lifetime of *Tcl_Obj* structures, but this change complicates the code. The simplest way to fix the performance is probably to use Donal Fellows's implementation of the *K* combinator:

```
proc K { x y } { set x }
```

which allows the caller of *lreplace* to extract the value of *list*, change the value of *list* so that the extracted value is unshared, and then pass the extracted value as a parameter to *lreplace:*

```
proc shuffle1a { list } {
    set n [llength $list]
    for { set i 0 } { $i < $n } { incr i } {
        set j [expr {int(rand()*$n)}]
        set temp1 [lindex $list $j]
        set temp2 [lindex $list $i]
        set list [lreplace [K $list [set list {}]] $j $j $temp2]
        set list [lreplace [K $list [set list {}]] $i $i $temp1]
    }
    return $list
}
```

Now the performance of the code is *O(n)* where *n* is the length of the list, but the programmer's intent has been seriously obscured! Moreover, the performance is still rather poor: Tcl makes an atrocious showing, for instance, in Doug Bagley's 'Great Computer Language Shootout' (http://www.bagley.org/~doug/shootout/).

33.2 Specification

This TIP proposes an 'lset' command with the syntax:

```
lset varName indexList value
```

or

```
lset varName index1 index2... value
```

where:

varName is the name of a variable in the caller's scope.

If *objc==4*, then the *indexList* parameter is interpreted as a list of *index* arguments; if *objc>4*, then the *index* arguments are inline on the command line.

In either case, Each *index* argument is an index in the content of *varName* or one of its sublists (see below). The format of *index* is either an integer whose value is at least zero and less than the length of the corresponding list, or else the literal string *end*, optionally followed with a hyphen and an integer whose value is at least zero and less than the length of the corresponding list.

value is a value that is to be stored as a list element.

The return value of the command, if successful, is the new value of *varName.*

The simplest form of the command:

```
lset varName index value
```

replaces, in place, the *index* element of *varName* with the specified *value*. For example, the code:

```
set x {a b c}
lset x 1 d
```

results in *x* having the value *a d c*. The result, except for performance considerations and the details of error reporting, is roughly the same as the Tcl code:

```
proc lset { varName index value } {
    upvar 1 $varName list
    set list [lreplace $list $index $index $value]
    return $list
}
```

except that where the *lreplace* command permits indices outside the existing list elements, the proposed *lset* command forbids them.

If multiple *index* arguments are supplied to the *lset* command, they refer to successive sublists in a hierarchical fashion. Thus,

```
lset varName $i $j value
```

or, equivalently,

```
lset varName [list $i $j] value
```

asks to change the value of the *j*th element in the *i*th sublist of *varName*. Hence, the code:

```
set x {{a b c} {d e f} {g h i}}
lset x 1 1 j; # -or- lset x {1 1} j
```

changes the value of *x* to

```
{a b c} {d j f} {g h i}
```

and the code

```
set y {{{a b} {c d}} {{e f} {g h}}}
lset y 1 0 1 i; # -or- lset y {1 0 1} i
```

changes the value of *y* to

```
{{a b} {c d}} {{e i} {g h}}
```

This notation also dovetails prettily with the extension of the *lindex* command proposed in [TIP #22]. The command

```
lindex $y 1 0 1; # -or- lindex y {1 0 1}
```

will extract the element that is set by the command

```
lset $y 1 0 1 $value
```

The *lset* command will throw an error and leave the variable unchanged if it is presented with fewer than three arguments, if any of the *index* arguments is out of range or ill-formed, or if any of the data being manipulated cannot be converted to lists. It will throw an error after modifying the variable if any write trace on the variable fails.

With the proposed *lset* command, the procedure to shuffle a list becomes much more straightforward:

```
proc shuffle1b { list } {
    set n [llength $list]
    for { set i 0 } { $i < $n } { incr i } {
        set j [expr {int(rand()*$n)}]
        set temp [lindex $list $j]
        lset list $j [lindex $list $i]
        lset list $i $temp
    }
    return $list
}
```

The given implementation copies the list only once, the first time that the line:

```
        lset list $j [lindex $list $i]
```

is executed. Thereafter, the list is an unshared object, and the replacements are performed in place.

33.3 Reference Implementation

The author has implemented a simpler variant of the proposed command as an object command, and also proposes to bytecode compile it, although the implementation of bytecode compilation is incomplete. The reference implementation also does not yet expand *objv[2]* as a list in the case where *objc==4*, and is known to have memory leaks where ill-formed index arguments are presented. It is given here as *concept code* and to present its impact on performance of some common list operations. (Obviously, it will be completed and reviewed with the relevant maintainers prior to being committed to the Core.)

The core of the implementation is the following procedure:

```
int
Tcl_LsetObjCmd( clientData, interp, objc, objv )
    ClientData clientData;      /* Not used. */
    Tcl_Interp *interp;         /* Current interpreter. */
    int objc;                   /* Number of arguments. */
    Tcl_Obj *CONST objv[];      /* Argument values. */
{
```

```
Tcl_Obj* listPtr;           /* Pointer to the list being altered. */
Tcl_Obj* subListPtr;        /* Pointer to a sublist of the list */
Tcl_Obj* finalValuePtr;     /* Value finally assigned to the variable */
int index;                  /* Index of the element being replaced */
int result;                 /* Result to return from this function */
int listLen;                /* Length of a list being examined */
Tcl_Obj** elemPtrs;         /* Pointers to the elements of a
                             * list being examined */
int i;

/* Check parameter count */

if ( objc < 4 ) {
    Tcl_WrongNumArgs( interp, 1, objv, "listVar index ?index...? value" );
    return TCL_ERROR;
}

/* Look up the list variable */

listPtr = Tcl_ObjGetVar2( interp, objv[ 1 ], (Tcl_Obj*) NULL,
                          TCL_LEAVE_ERR_MSG );
if ( listPtr == NULL ) {
    return TCL_ERROR;
}

/* Make sure that the list value is unshared. */

if ( Tcl_IsShared( listPtr ) ) {
    listPtr = Tcl_DuplicateObj( listPtr );
}

finalValuePtr = listPtr;

/*
 * If there are multiple 'index' args, handle each arg except the
 * last by diving into a sublist.
 */

for ( i = 2; ; ++i ) {

    /* Take apart the list */

    result = Tcl_ListObjGetElements( interp, listPtr,
                                     &listLen, &elemPtrs );
    if ( result != TCL_OK ) {
        return result;
    }

    /* Derive the index of the requested sublist */

    result = TclGetIntForIndex( interp, objv[i], (listLen - 1), &index );
    if ( result != TCL_OK ) {
        return result;
    }

    if ( ( index < 0 ) || ( index >= listLen ) ) {

        Tcl_SetObjResult( interp,
                          Tcl_NewStringObj( "list index out of range",
                                            -1 ) );
        return TCL_ERROR;
    }
```

```
        /* Break out of the loop if we've extracted the innermost sublist. */

        if ( i >= ( objc - 2 ) ) {
            break;
        }

        /*
         * Extract the appropriate sublist, and make sure that it is unshared.
         */

        subListPtr = elemPtrs[ index ];
        if ( Tcl_IsShared( subListPtr ) ) {
            subListPtr = Tcl_DuplicateObj( subListPtr );
            result = Tcl_ListObjSetElement( interp, listPtr, index,
                                            subListPtr );
            if ( result != TCL_OK ) {
                return TCL_ERROR;
            }
        } else {
            Tcl_InvalidateStringRep( listPtr );
        }

        listPtr = subListPtr;
    }

    /* Store the result in the list element */

    result = Tcl_ListObjSetElement( interp, listPtr, index, objv[objc-1] );
    if ( result != TCL_OK ) {
        return result;
    }

    /* Finally, update the variable so that traces fire. */

    listPtr = Tcl_ObjSetVar2( interp, objv[1], NULL, finalValuePtr,
                              TCL_LEAVE_ERR_MSG );
    if ( listPtr == NULL ) {
        return TCL_ERROR;
    }

    Tcl_SetObjResult( interp, listPtr );
    return result;

}
```

The procedure depends on a new service function, *Tcl_ListObjSetElement*:

```
int
Tcl_ListObjSetElement( interp, listPtr, index, valuePtr )
    Tcl_Interp* interp;         /* Tcl interpreter; used for error reporting
                                 * if not NULL */
    Tcl_Obj* listPtr;           /* List object in which element should be
                                 * stored */
    int index;                  /* Index of element to store */
    Tcl_Obj* valuePtr;          /* Tcl object to store in the designated
                                 * list element */
{
    int result;                 /* Return value from this function */
    List* listRepPtr;           /* Internal representation of the list
                                 * being modified */
    Tcl_Obj** elemPtrs;         /* Pointers to elements of the list */
```

```
    int elemCount;              /* Number of elements in the list */

    /* Ensure that the listPtr parameter designates an unshared list */

    if ( Tcl_IsShared( listPtr ) ) {
        panic( "Tcl_ListObjSetElement called with shared object" );
    }
    if ( listPtr->typePtr != &tclListType ) {
        result = SetListFromAny( interp, listPtr );
        if ( result != TCL_OK ) {
            return result;
        }
    }
    listRepPtr = (List*) listPtr->internalRep.otherValuePtr;
    elemPtrs = listRepPtr->elements;
    elemCount = listRepPtr->elemCount;

    /* Ensure that the index is in bounds */

    if ( index < 0 || index >= elemCount ) {
        if ( interp != NULL ) {
            Tcl_SetObjResult( interp,
                              Tcl_NewStringObj( "list index out of range",
                                                -1 ) );
            return TCL_ERROR;
        }
    }

    /* Add a reference to the new list element */

    Tcl_IncrRefCount( valuePtr );

    /* Remove a reference from the old list element */

    Tcl_DecrRefCount( elemPtrs[ index ] );

    /* Stash the new object in the list */

    elemPtrs[ index ] = valuePtr;

    /* Invalidate and free any old string representation */

    Tcl_InvalidateStringRep( listPtr );

    return TCL_OK;

}
```

Even without bytecode compilation, the performance improvement of array-based applications that can be achieved by the *lset* command is substantial. The following table shows run times in microseconds (on a 550 MHz Pentium III laptop, running a modified Tcl 8.4 on Windows NT 4.0, Service Pack #6) of the three implementations of *shuffle* that appear in this TIP.

RUN TIMES IN MICROSECONDS

	Version		
	shuffle1 (Naive)	shuffle1a (K combinator)	shuffle1b (lset command)
List length			
1	26	32	27
10	108	152	101
100	1627	1462	936

```
 1000             117831            14789             9574
10000       Test stopped           152853            96912
```

Similar (30-50%) improvements are observed on many of the array related benchmarks that have been proposed. Bytecode compilation is expected to produce even greater improvements.

Another area where *lset* can achieve a major performance gain is in memory usage. The author of this TIP has benchmarked competing implementations of heapsort, one using Tcl arrays, and the other using *lset* to manipulate lists as linear arrays. When sorting 80000 elements, the Tcl-array-based implementation used 12.7 megabytes of memory; the list-based implementation was faster and used only 5.6 megabytes. The explanation is simple: each entry in the hash table requires an allocated block of twenty bytes of memory, plus the space required for the hash key. The hash key is a string, and requires at least six bytes. When both of these objects are aligned and padded with the overheads imposed by *ckalloc*, they require about 80 bytes of memory on the Windows NT platform. The memory cost of an element of a Tcl list, by comparison, is four bytes to hold the pointer to the object.

33.4 Discussion

There are several objections that can be foreseen to this proposal.

- *Why implement the command in the Core and not as an extension?*

  In a word, *performance.* At the present state of Tcl development, only Core commands can be bytecoded. The cost of the hash table lookups in the *Tcl_ObjGetVar2* and *Tcl_ObjSetVar2* calls is significant, and can be eliminated from many common usages by the bytecode compiler. Since this command is likely to appear in inner loops, it is important to squeeze every bit of possible performance out of it.

- *Why a new command in the global namespace?*

  The author of this TIP feels that having a single added command that is parallel to the existing list commands is not polluting the namespace excessively. It would be a shame if this proposal founders upon the Naming of Names.

- *Why a new command, rather than including this functionality in the proposed functionality of an extensible command for list manipulation?*

  The author of this TIP has yet to see a formal proposal of any extensible list manipulation command; the closest thing appears to be Andreas Kupries's *listx* package (http://www.oche.de/~akupries/tcltk.html). Given the size and complexity of any such modification, it is unlikely that it will be available in the Core in time for an 8.4 release. The performance improvements achievable by the *lset* command are needed urgently.

- *Isn't this [TIP #29] warmed over?*

  Several objectors to [TIP #29] indicated that they were willing to consider list element assignment implemented as a new command.

- *Doesn't this proposal depend on multiple* index *arguments to* lindex″ ([TIP #22])?

  This proposal can stand alone. If multiple *index* arguments to *lindex* are also accepted, the resulting symmetry is pleasing. Having multiple *index* args to *lset* is much more important, because it is horribly difficult to implement equivalent functionality in pure Tcl without introducing excessive calls to *Tcl_DuplicateObj*. In fact, the reference implementation of *lset* presented in this TIP was motivated by the fact that its author gave up on the task and resorted to C.

33.5 Implementation Notes

Having two versions of the syntax for the *lset* command is perhaps unattractive, but neither can be left out effectively.

The syntax where the indices are packaged as a single list allows a *cursor* into complex list structure to be maintained in a single variable. The list element that the cursor designates can be altered with a single call to

the *lset* command, without needing to resort to *eval* (a command that is both expensive and dangerous) to expand the indices inline.

The syntax where each index is a first-class object is motivated by the performance of array-based algorithms. Programmers who are using lists as arrays know exactly how many subscripts they have, and in fact are generally iterating through them. A typical sort of usage might be the nave matrix multiplication shown below.

```
# Construct a matrix with 'rows' rows and 'columns' columns
# having an initial value of 'initCellValue' in each cell.

proc matrix { rows columns { initCellValue {} } } {
    set oneRow {}
    for { set i 0 } { $i < $columns } { incr i } {
        lappend oneRow $initCellValue
    }
    set matrix {}
    for { set i 0 } { $i < $rows } { incr i } {
        lappend matrix $oneRow
    }
    return $matrix
}

# Multiply two matrices

proc matmult { x y } {

    set m [llength $x];                 # Number of rows of left matrix
    set n [llength [lindex $x 0]];      # Number of columns of left matrix

    if { $n != [llength $y] } {
        return -code error "rank error: left operand has $n columns\
                            while right operand has [llength $y] rows"
    }

    set k [llength [lindex $y 0]];      # Number of columns of right matrix

    # Construct a matrix to hold the product

    set product [matrix $m $k]

    for { set i 0 } { $i < $m } { incr i } {
        for { set j 0 } { $j < $k } { incr j } {
            lset product $i $j 0.0
            for { set r 0 } { $r < $n } { incr r } {
                set term [expr { [lindex $x $i $r] * [lindex $y $r $j] }]
                lset product $i $j [expr { [lindex $product $i $j] + $term }]
            }
        }
    }

    return $product
}
```

Note how we have an [lset] operation in the innermost loop, executed (m*n*k) times.

If in this instance, we have to write:

```
                set indices [list $i $j]
                lset product $indices \
                    [expr { [lindex $product $indices] + $term }]
```

in place of the [lset] shown above, we add the cost of forming the list of indices to the cost of the inner loop. This cost is not to be sneezed at -- it's two expensive calls to *ckalloc.* (The cost can be avoided, at some cost in

readability, by maintaning a variable containing the index list, and altering its elements with other uses of [lset].)

Richard Suchenwirth suggested the compromise that appears in this proposal. This scheme will perilous to performance if implemented naively. If the implementation of [lset] simply calls *Tcl_ListObjGetElements*, look what happens to the inner loop of our *shuffle1b* procedure:

```
for { set i 0 } { $i < $n } { incr i } {
    set j [expr {int(rand()*$n)}]
    set temp [lindex $list $j]
    lset list $j [lindex $list $i]
    lset list $i $temp
}
```

- Initially, {set i 0} sets i to the constant "0"; it is a string.
- Evaluating the conditional {$i < $n} will shimmer i to an integer; now it's an integer. (We had to do a call to strtol here.)
- The [lindex $list $i] call now has to consider $i as a list of indices, and shimmers it to the list. This discards the internal rep, parses the string rep into a list, and then reconverts its first element to an integer.
- OK, now the 'lset' is happy, and no further shimmering occurs...
- ... until we get to the {incr i}. Now we go back to the string rep once again, shimmer it to an integer (yet another call to strtol), and invalidate the string rep because we've incremented the integer.
- Now we get back into the [lindex] once again, and need a list rep. This time, we have to format the integer as a string, parse it as a list, take the object representing element 0, and reparse that as an integer.

This sequence has converted the integer to and from a string, and performed four calls to *ckalloc*, but resulted in the same integer that we started with!

It is possible for a sufficiently smart compromise implementation to avoid all this shimmering. In the case where *objc==4*, the *lset* command must:

1. Test whether *objv[2]* designates an object whose internal representation holds an integer. If so, simply use it as an index.
2. Test whether *objv[2]* designates an object whose internal representation holds a list. If so, perform the recursive extraction of indexed elements from sublists described above.
3. Form the string representation of *objv[2]* and test whether it is *end* or *end-* followed by an integer. If so, use it as an index.
4. Attempt to coerce *objv[2]* to an integer; if successful, use the result as an integer.
5. Attempt to coerce *objv[2]* to a list; if successful, use the result as an index list.
6. Report a malformed *index* argument; the *indexList* parameter is not a well-formed list.

This logic handles all the cases of singleton lists transparently; it is effectively a simple-minded type inference that optimizes away needless conversions. With it in place, none of the *lset* examples shown in this TIP will suffer from type shimmering.

In the event that the related [TIP #22] is approved, the logic for parsing an index list will likely be combined with that used in the *lindex* command.

Bytecoding variadic commands like *lset* presents some interesting technical challenges; a discussion in progress on the Tcl'ers Wiki (http://purl.org/thecliff/tcl/wiki/1604) is recording the design decisions being made for bytecoding *lset* so that they can be applied to similar commands in the future.

33.6 See Also

[TIP #22], [TIP #29].

33.7 Change History

This TIP has undergone several revisions by the original author. The most significant was made on 20 May 2001, where the syntax was revised to allow for either several indices inline on the command line or a list of indices.

33.8 Copyright

TIP #34: TEA 2.0

TIP #34: **TEA 2.0**	
Author:	Mo DeJong ⟨mdejong@cygnus.com⟩
Created:	Thursday, 3rd May 2001
Type:	Project
Tcl Version:	8.4
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.1 $`
Post-History:	

Abstract

The original TEA specification, documentation, and implementation have fallen out of date. Numerous complaints about the difficulty of creating a TEA compliant package have appeared on news:comp.lang.tcl The existing build system works but it is a pain to maintain mostly because there are two build systems, one for unix and another for windows. This document describes how some of these concerns can be addressed.

34.1 Rationale

As new software is released, existing documentation becomes obsolete. Some of the existing TEA documentation is now so badly out of date that suggested software releases are no longer available. The solution to this problem is simple, the TEA documentation and implementation must be updated.

The build system itself is in need of an update. The Unix and Windows versions of the build system are not synchronized. There are a number of features that are simply not implemented in the Windows version. The most straightforward way of dealing with this problem is to merge the two build systems. Some popular extensions have already taken this approach, Itcl for example uses a single *configure.in* script to build the Unix and Windows versions. While switching to a single *configure.in* is a big step, it will significantly simplify maintenance and make life a lot easier in the long run.

Tcl's build system does not depend on *config.guess* and *config.sub* to determine build and host triples. Instead, it depends on the output of *uname -s* and *uname -r*. That works for native builds but makes it very painful to cross compile. For example, a user might want to build Windows binaries under Linux. Tcl's existing build system makes this much harder than it needs to be. Upgrading to *autoconf 2.50* is the best way to address this problem.

Tcl's build system passes a large number of *-D* flags to the compiler instead of making use of a *config.h* file. Personal experience has shown that using a *config.h* file is a superior way of dealing with configure time defines.

34.2 Implementation Notes

Implementing this TIP is by no means an easy task. Build system changes are by far the most dangerous since a mistake that breaks something on some infrequently used configuration will not be noticed until some time in the future. One can only ask for forgiveness up front since it is a virtual certainty that these sorts of changes are going to break something. The needed documentation changes are straightforward, but the actual process make take a long time.

34.3 Alternatives

The alternative is to continue to use the existing system. Things would get no worse but they would also get no better.

34.4 Copyright

TIP #35: Enhanced Support for Serial Communications

TIP #35:	**Enhanced Support for Serial Communications**
Author:	Rolf Schroedter ⟨rolf.schroedter@dlr.de⟩
Created:	Wednesday, 6th June 2001
Type:	Project
Tcl Version:	8.4
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.6 $`
Post-History:	

Abstract

Tcl's support for RS-232 is very rudimentary. Mainly it allows to setup the communication rate [fconfigure -mode] and to read and write data with the standard Tcl functions. Real serial communications are often more complex. Therefore it is proposed to add support for hardware and software flow control, polling RS-232 (modem) status lines, and watching the input and output queue. This is all to be implemented via additional [fconfigure] options.

35.1 Rationale

There is an undamped interest in serial communications, because it's very easy to connect external hardware to a computer using the RS-232 ports.

However Tcl's support for serial communications is not complete. Real applications often need more than setting the baud rate and to read/write data bytes.

Especially if the external hardware is slow or the communication rate is high one needs support for flow-control (hard- and software). These features are provided by the operating system drivers, but Tcl's [fconfigure] doesn't support it.

On the the other hand there are cases that the external hardware makes static use of the RS-232 signals to signal external events via the modem status lines or even to be powered by the RS-232 control lines.

Additionally for non-blocking serial I/O it may be interesting for the Tcl application to know about the status of the input and output queues to read a fixed size block or to support communication timeouts.

At this opportunity it is proposed to move the documentation of the serial port fconfigure options form the *open.n* man-page to *fconfigure.n*.

35.2 Specification

It is proposed to have following set of [fconfigure] options for serial communications:

-mode baud,parity,data,stop *(Windows and Unix).* Already implemented.

-handshake mode *(Windows and Unix).* This option is used to setup automatic handshake control. Note that not all handshake modes maybe supported by your operating system. The mode parameter is case-independent.

If mode is *none* then any handshake is switched off. *rtscts* activates hardware handshake. For software handshake *xonxoff* the handshake characters can be redefined with [fconfigure -xchar]. An additional hardware handshake *dtrdsr* is available only for Windows. There is no default handshake configuration, the initial value depends on your operating system settings. The -handshake option cannot be queried, because the operating system settings may be ambiguous.

-xchar {**xonChar xoffChar**} *(Windows and Unix).* This option is used to change the software handshake characters. Normally the operating system default should be DC1 (0x11 hex) and DC3 (0x13 hex) representing the ASCII standard XON and XOFF characters. The -xchar option cannot be queried.

-ttycontrol {**signal boolean signal boolean ...**} *(Windows and Unix).* This option is used to setup the handshake output lines permanently or to send a BREAK over the serial line. The *signal* names are case-independent.

{RTS 1 DTR 0} sets the RTS output to high and the DTR output to low. For POSIX systems {BREAK 1} sends a break signal (zero-valued bits) for 0.25 to 0.5 seconds and {BREAK 0} does nothing. For Windows the break is enabled and disabled with {BREAK 1} and {BREAK 0} respectively. It's not a good idea to change the RTS (or DTR) signal with active hardware handshake *rtscts* (or *dtrdsr*). The result is unpredictable. The -ttycontrol option cannot be queried.

-ttystatus *(Windows and Unix).* The -ttystatus option can only be queried. It returns the current modem status and handshake input signals. The result is a list of signal,value pairs with a fixed order, e.g. {CTS 1 DSR 0 RING 1 DCD 0}. The *signal* names are returned upper case.

-queue *(Windows and Unix).* The -queue option can only be queried. It returns a list of two integers representing the current number of bytes in the input and output queue respectively.

-sysbuffer inSize

-sysbuffer {**inSize outSize**} *(Windows only, Unix ?).* This option is used to change the size of Windows system buffers for a serial channel. Especially at higher communication rates the default input buffer size of 4096 bytes can overrun for latent systems. The first form specifies the input buffer size, in the second form both input and output buffers are defined.

-pollinterval msec *(Windows only)*. Already implemented.

-lasterror *(Windows only, Unix?)*. Already implemented for Windows.

35.3 Implementation Details

For Unix (*termios.h*) systems the proposed changes are very straight forward, because Unix channels can be configured blocking or non-blocking. One only needs to add the serial [fconfigure] options calling the appropriate *ioctl()* functions to configure the serial port.

For Windows reading and writing files is generally blocking. Especially with activated handshake the serial communication can stop forever. Therefore the Windows implementation needs at least a writing thread preventing Tcl's main application to block. Additionally Windows provides a reach set of special APIs for serial communication which needs to be translated to [fconfigure] options.

There is one special point about Windows: For making multiple threads accessing a serial port, it needs to be opened with the OVERLAPPED flag set. Tcl detects a serial port only after opening it without the OVERLAPPED flag. Therefore this port has to be reopened, which requires a little change to *tclWinChan.c* and *tclWinPort.h*.

Macintosh systems — ?

35.4 Changed Files

tclUnixChan.c Add [fconfigure] options.

tclWinPort.h Declare a new function *TclWinSerialReopen()*

tclWinChan.h Call *TclWinSerialReopen()* after detecting the serial port.

tclWinSerial.c Partial rewrite of Tcl's serial driver. The current implementation only performs blocking output. Add [fconfigure] options.

fconfigure.n Serial [fconfigure] options should be documented here.

open.n Serial port filenames are documented here. Add a link to [fconfigure] for additional serial options.

35.5 Other Issues

It has also been proposed to add a [fconfigure -timeout] option specifying read and write timeouts. Together with a blocking read a timeout could be used to wait for an expected number of data bytes from the serial port. There are two arguments against timeouts:

1. Adding timeout to blocking I/O at the driver level radically changes the behaviour of read/write operations. This adds a lot of oddity to serial communications.
2. Timeouts can easily be implemented at Tcl level using non-blocking I/O together with Tcl's event loop. Additional support is given by [fconfigure -queue].

35.6 Copyright

TIP #36: Library Access to 'Subst' Functionality

TIP #36: **Library Access to 'Subst' Functionality**	
Author:	Donal K. Fellows ⟨fellowsd@cs.man.ac.uk⟩
Created:	Wednesday, 13th June 2001
Type:	Project
Tcl Version:	8.4
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.1 $`
Post-History:	

Abstract

Some applications make very heavy use of the *subst* command — it seems particularly popular in the active-content-generation field — and for them it is important to optimise this as much as possible. This TIP adds a direct interface to these capabilities to the Tcl library, allowing programmers to avoid the modest overheads of even *Tcl_EvalObjv* and the option parser for the *subst* command implementation.

36.1 Functionality Changes

There will be one script-visible functionality change from the current implementation; if the evaluation of any command substitution returns TCL_BREAK, then the result of the *subst* command will be the string up to that point and no further. This contrasts with the current behaviour where TCL_BREAK (like TCL_CONTINUE) just causes the current command substitution to finish early.

36.2 Design Decisions

The code should be created by effectively splitting *Tcl_SubstObjCmd* in the current *.../generic/tclCmdMZ.c* into two pieces. One of these pieces will have the same interface as the present code and will contain the argument parser. The other piece will be the implementation of the *subst* behaviour and will be separately exposed at the C level as well as being called by the front-end code.

The code should take positive flags stating what kinds of substitutions should be performed, as this is closest to the current internal implementation of the *subst* command. These flags will be named with the prefix TCL_SUBST_*. For programming convenience, the flag TCL_SUBST_ALL will also be provided allowing the common case of wanting all substitutions to be performed with a minimum of fuss.

The string to be substituted will be passed in as a *Tcl_Obj* * too, as this is both easiest to do from the point-of-view of the front-end code and permits additional optimisation of the core at some future point if it proves necessary and/or desirable. By contrast, passing in a standard C string or a *Tcl_DString* * does not permit any such optimisations in the future.

The code should return a newly-allocated *Tcl_Obj* * as this allows for the efficient implementation of the front-end involving no re-copying of the resulting string. It also allows error conditions to be represented by NULL (with an error message in the interpreter result) and does not force a Tcl_DString reference to be passed in as an *out* parameter; returning the result gives a much clearer call semantics. Another advantage of using *Tcl_Obj*s to build the result is the fact that they have a more sophisticated memory allocation algorithm that copes more efficiently with very large strings; when large and small strings are being combined together (as is easily the case in *subst*) this can make a substantial difference.

36.3 Public Interface

Added to .../generic/tcl.h

```
#define TCL_SUBST_COMMANDS    0x01
#define TCL_SUBST_VARIABLES   0x02
#define TCL_SUBST_BACKSLASHES 0x04
#define TCL_SUBST_ALL         0x07
```

Added to .../generic/tcl.decls

```
declare someNumber generic {
    Tcl_Obj * Tcl_SubstObj( Tcl_Interp *interp,
                            Tcl_Obj *objPtr,
                            int flags)
}
```

36.4 Implementation

The implementation is to be developed upon acceptance of this TIP, but will involve *Tcl_AppendToObj* and *Tcl_AppendObjToObj*.

36.5 Copyright

TIP #37: Uniform Rows and Columns in Grid

TIP #37:	**Uniform Rows and Columns in Grid**
Author:	Peter Spjuth ⟨peter.spjuth@space.se⟩
Created:	Tuesday, 19th June 2001
Type:	Project
Tcl Version:	8.4
State:	Draft
Vote:	Pending
Version:	`$Revision: 1.2 $`
Post-History:	

Abstract

This TIP proposes to add a *-uniform* option to *grid rowconfigure* and *grid columnconfigure* so as to make it easier to create layouts where cells are constrained to have identical dimensions.

37.1 Introduction

The geometry managers in Tk are very powerful and can do most things needed to layout a GUI. One thing that is tricky to do though is to put widgets in rows or columns of the same width. This would be useful for example to layout a row of buttons symmetrically. This could easily be done with the grid manager if an additional option is added.

37.2 Specification

Anywhere *column* is used below, the same applies to *row* too.

A new option, *-uniform*, is added to *grid columnconfigure*. The option takes an arbitrary string, the default value being the empty string. Any column with a non-empty value will be grouped with other columns with the same value. Each column in a group will get the size k*-*weight* (in this aspect a *-weight* value of 0 is used as 1) , where k is set so that no column becomes smaller. E.g., if all columns in a group have the same *-weight* they will all get the size of the largest member.

In the grid algorithm *-uniform* and *-weight* will be used in the calculation of the requested size, but for the distribution of extra size only *-weight* will be considered.

37.3 Rationale

To only consider *-weight* in the extra size distribution is mainly a matter of simplicity. It gives a simpler algorithm that is both easier to explain to the user and to code.

To uphold the uniform property it would be needed to force any zero *-weight* value in a group where any non-zero *-weight* exists to be set to one before doing the resize calculations. A bit complicated and the only benefit for the user would be to only have to specify *-weight* for one column in a group. But in practice this is hardly no gain at all since a typical usage looks like this:

```
grid columnconfigure . {0 1 2} -uniform a -weight 1
```

I'm not sure if someone would have a use for the effect you would get by mixing zero and non-zero weights in a group but this leaves you the freedom to do so.

37.4 Implementation

A quick try shows that this is fairly straightforward to implement. The memory cost for the change is a *Tk_Uid* field in the column slot structure to hold the option, and the CPU overhead is small for a grid that don't use the option.

37.5 Copyright

References

[TIP #0] John Ousterhout, *Tcl Core Team Basic Rules*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/0.html

[TIP #1] TIP Editor, *TIP Index*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/1.html

[TIP #2] Andreas Kupries, Donal K. Fellows, Don Porter, Mo DeJong, Larry W. Virden, *TIP Guidelines*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/2.html

[TIP #3] Andreas Kupries, Donal K. Fellows, *TIP Format*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/3.html

[TIP #4] Brent Welch, Donal K. Fellows, Larry W. Virden, Larry W. Virden, *Tcl Release and Distribution Philosophy*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/4.html

[TIP #5] Eric Melski, *Make TkClassProcs and TkSetClassProcs Public and Extensible*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/5.html

[TIP #6] Mark Harrison, *Include [Incr Tcl] in the Core Tcl distribution*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/6.html

[TIP #7] Kevin Kenny, *Increased resolution for TclpGetTime on Windows*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/7.html

[TIP #13] Don Porter, Donal K. Fellows, *Web Service for Drafting and Archiving TIPs*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/13.html

[TIP #16] Don Porter, *Tcl Functional Areas for Maintainer Assignments*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/16.html

[TIP #19] Neil McKay, *Add a Text Changed Flag to Tk's Text Widget*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/19.html

[TIP #22] David Cuthbert, Kevin Kenny, Don Porter, Donal K. Fellows, *Multiple Index Arguments to lindex*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/22.html

[TIP #23] Kevin Kenny, Jim Ingham, Don Porter, *Tk Toolkit Functional Areas for Maintainer Assignments*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/23.html

[TIP #29] Kevin Kenny, Donal K. Fellows, *Allow array syntax for Tcl lists*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/29.html

[TIP #31] Don Porter, miguel sofer, Jeff Hobbs, Kevin Kenny, *CVS tags in the Tcl and Tk repositories*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/31.html

[TIP #33] Kevin Kenny, *Add 'lset' Command to Assign to List Elements.*, on-line at http://www.cs.man.ac.uk/fellowsd-bin/TIP/33.html