# A new feature of the Tcl core: "Stacked channels"

Andreas Kupries <a.kupries@westend.com>
http://www.purl.org/NET/akupries/
Aachen

March 3, 2000

**Abstract**

This paper is about a new feature which was recently added to the Tcl core called "stacked channels", the reasons why it was invented and its possible and actual applications.

## 1 Introduction

Whenever an application written in `tcl` is talking to another application in a continuous way or creating persistent information it uses pipes, sockets, files, ..., i.e. channels and the I/O system of the `tcl` core. This makes that subsystem the second most important one of the core right after the pure interpreter together with its concept of variables.

Because of this flaws in it may affect and limit its users profoundly. On the other hand this also means that any change to it should be considered carefully before its realization. The rest of this paper describes such a change to the I/O system which was recently adopted, and the reasons behind it, i.e. the problems it solves. It is structured like this:

Section 2 describes some of the problems which plagued the I/O system of the core before version 8.2. This is followed by section 3 giving an overview of the proposed and adopted solution to this problems. Here we will see examples of applications and extensions actually using this feature to solve their problems with it too. This is followed by section 4 which then goes into the technical details of the implementation of stacked channels. At last we draw some conclusions and take a look into the future of the I/O system, especially as the stacked channels as they are now by no means solve all the problems with it.

## 2 Problems of the Tcl I/O system before Tcl 8.2

Suppose

- you have an application made up of two parts communicating through a pipe, f.e. a mathematical engine and its GUI. Now move the engine to a different, possible more powerful machine.

  Is this a problem ? No. We have to change the application to use a socket instead of a pipe, but only the small part of it which deals with opening the connection between engine and GUI. Why ? Because both pipe and socket are hidden behind a single concept, the channel. The places opening it have to change, but the rest of the application can deal with it unchanged.

  Now suppose that the management decides that the communication between the two parts should be secure too and requests the implementation of some encryption protocol. This is a problem. Every part of the application involved in communication now has to be rewritten to take care of this new feature. A nightmare in maintenace just opened up, destroying any modularity the application might have had in its parts.

  Another problem of this is that the code to invoke the encryption is scattered all over the place.

- an application reading and writing configuration files faces the request to compress them for writing[1]. Again all places in the application reading/writing the configuration have to change to deal with this new requirement.

The general theme behind these two examples is that the have operations transforming streams of bytes in some way and thus similar to channels, but which have to be implemented as a separate thing despite this and thus forcing changes all over in applications using them instead of the relatively small number of places opening the channels used by the majority of the application.

## 3 Proposed solution

The solution proposed here is to crack open the encapsulation provided by the channel abstraction in a controlled way, thus giving extensions and possibly even scripts not only access to the data flowing through a channel but the ability to manipulate it as well.

The means to that end is the notion of "stacking" channels upon each other. Instead of only creating new standalone channels allow it that a new channel takes the identity of an existing channel, and additionally force it to write anything written to it into the channel it superceded and force it too to read any data it requires to satisfy read requests from the superceded channel.

Together with the existing ability to enhance the I/O system through new drivers it is now possible to wrap any transformation we like into a channel and use without

[1] and of course to decompress them while reading

disturbing the parts of any application not dealing with the creation of the channels it is using.

This means that the problems described in the last section are no such anymore. Both are solved by

1. writing an extension wrapping the desired transformations (encryption, compression),
2. creating a channel type using this extension (or actually being a part of the extension)
3. and stacking a channel of the desired type (encryption, . . . ) over the pipe, socket, file, . . . the application usually deals with. Only the places opening this channel have to be modified, but to the rest of the application the channel appears to be unchanged and is used before.

This technology of stacking channels even opens up new possibilities like

- error-correction for noisy channels,
- or, related to this, the generation and injection of authenticating hashes,
- usage of encodings like Base64 to transfer binary data over not 8bit-clean channels,
- passive taps to log the activity on a channel to
  - generate statistics
  - or debug some code implementing an internet protocol (SMTP, FTP, etc. are all human-readable).
- Two- or even multiway transfer between channels[2]

Examples of applications and extensions actually using this feature are:

- In the area of security we have `TLS`[12] by Matt Newman[6]. This extension implements the SSL protocol using `OpenSSL`[19] as its engine.

  A practical application is its use as part of the `TclHttpd`[14] so that it is able to provide the `https` schema (secure `HTTP`), a definitive requirement for e-commerce.

  Apart from eCommerce this could be used to create secure virtual private networks too, i.e. several tcl applications securely interacting over long distances.

  Related to this: Another package which could benefit from this is the `Comm`[15] package by John R. LoVerso[10] which implements the `send`-command of Tk, but using sockets instead of X properties.

  An application of all this might be the secure and trusted access to the Tcl equivalent of the Comprehensive Perl Archive Network.

[2] Sort of an extended `fcopy`. This is a new idea coming from Matt Newman[6]

- Another extension in the security area is `TrfCrypt`. As it provides only raw encryption systems and no complete security protocol around them it is more fitting in case of small-scale usage or more closed environments.
- The base framework for `TrfCrypt` is contained in and provided by `Trf`[16]. This extension additionally provides converters for several methods of encoding data (like Base64) and implementations of important authenticating hashes (like MD5).

  An application of these is `Tcl MIME`[13], a mail handling extension written by Marshall Rose[9]. It uses the stacking of channels to push and pop the encodings requested by the various "content-encoding:" headers onto the channel a MIME-compliant mail is written too.
- As `Trf` also provides a `zlib`-transformation[20] doing data compression a completely tcl-based client to CVS[3][18] is now feasible too.

The idea described above is not new. The *streams* abstraction invented by Dennis Ritchie[5, 1] and used in various Unixes to interconnect the modules handling the various network protocols[2, 3, 4] does a similar thing as stacked channels, but in kernel space instead of user space. The differences, which do exist, are something which we will talk later about, in section 5.

# 4 Technical details

Having seen the applications of the new feature we can now turn toward its implementation, i.e. the actual changes which were made to the core.

Let us first have a look at the general state of affairs before version 8.2 of the tcl core. This is shown in figure 1. A script uses a channel, and the data going through it is directed through the specific driver into the operating system.

The first attempt at implementing this feature actually had no notion of stacking channels but tried to modify the buffer management system in the I/O system instead to insert the necessary hooks for the transformations. This is shown in figure 2.

This approach soon proved to be virtually unmaintainable due to the complexity of the existing code and was given up in favor of the current implementation which mainly used the existing venues for extending the I/O system, i.e. channel types with their custom drivers. This approach delivered 100 percent of the required functionality even without actually patching the core. This is shown in figure 3.

The problem here? Any new stacked channel gets its own identity and leaves the original channel accesssible to the user, both at C- and script-level, thus leaving the door wide open for the accidential creation of an inconsistent state in the transformation and/or of corrupted data in the original channel. Closing these potential sources of gross errors requires patching the core as internal data structures[4] have to be manipulated. See figure 4.

[3]The newer versions of CVS use zlib embedded into their remote access protocol to minimize the amount of data to transfer

[4]The global list of all channels, maintained accross all interpreters

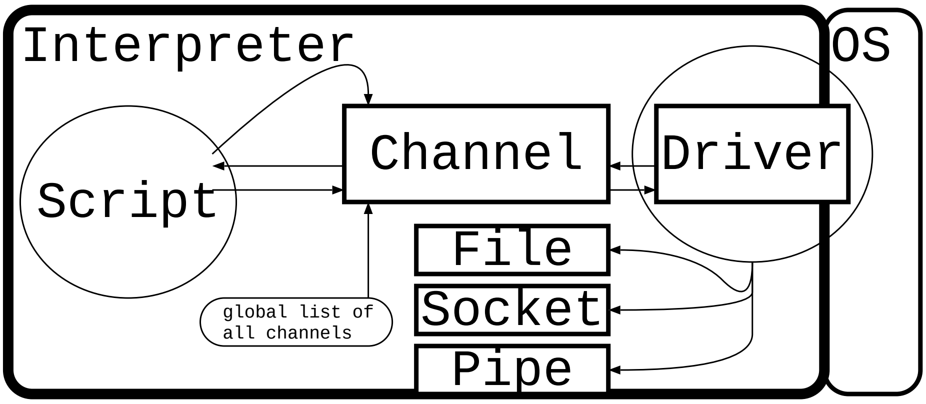


Figure 1: I/O system before Tcl 8.2

The changes in detail:

- The generic channel structure became a new field “supercedes”. For each channel it refers to the channel it has taken over. For normal channels this field contains NULL.
- Two new procedures were added to the public C-API allowing the user to
  - stack a new channel over an existing one,
  - and to remove the channel from the top of a sstack.

  Both procedures take care of manipulating the inter- and cross-interpreter data structures used to manage all channels, and setup and modify the relevant channel structures themselves. This includes
  - the manipulation of the eventsystem related structures (ChannelHandlers, etc.) for proper integration of the stacked channel into it[5] and
  - the manipulation of the original channel to switch off all its functionality which could interfere with the transformation, namely buffering, EOL-translation and UTF-8 encoding.

# 5 Conclusions & Future directions

With this new feature we now have another method to help us writing modular applications, helping us in glueing functionality together. It doesn’t solve all of our problem, though. Things we still have to consider are:

[5]To preserve working `fileevent`s was an absolute requirement for the patch.

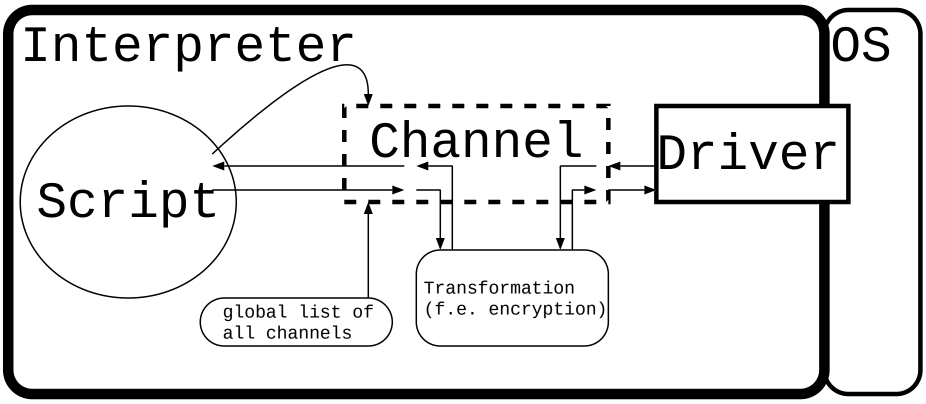


Figure 2: I/O system after 1st attempt to implement the new feature

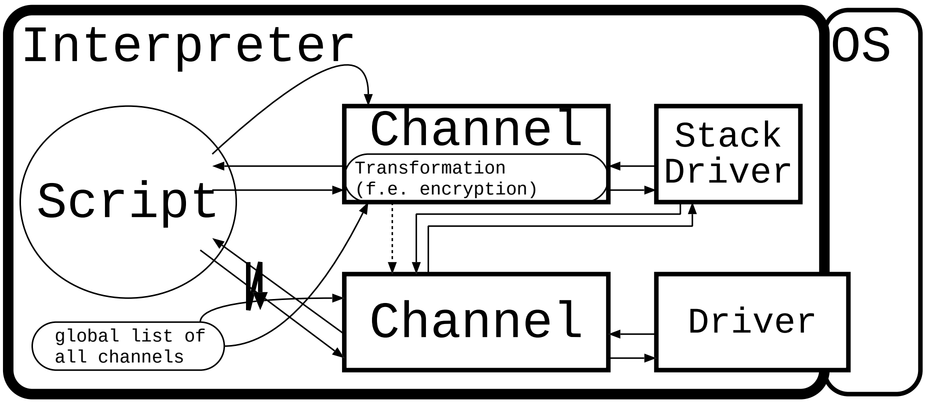


Figure 3: I/O system with new feature, without patching

- The code for the generic part of the I/O system contains three operations which are mixed together but should really be implementated as separate transformations, as shown in figure 5.

  Doing so is a major effort and although retaining backward compatibility is possible for the important parts of the I/O system it is not possible for all of them.

  It was already said that the approach of directly modifying the buffer management to insert the hooks required for this feature was given up due to the complexity of it. It is unfortunate that the Core Team decided to implement the UTF-8 encoding mechanism exactly so, thus increasing the complexity and lowering maintainability.

  It is even more unfortunate that we now have an inconsistent API for writing to a channel, as some of the public procedures go through the encoding stage and others bypass them. No way of unbundling the three internal transformations

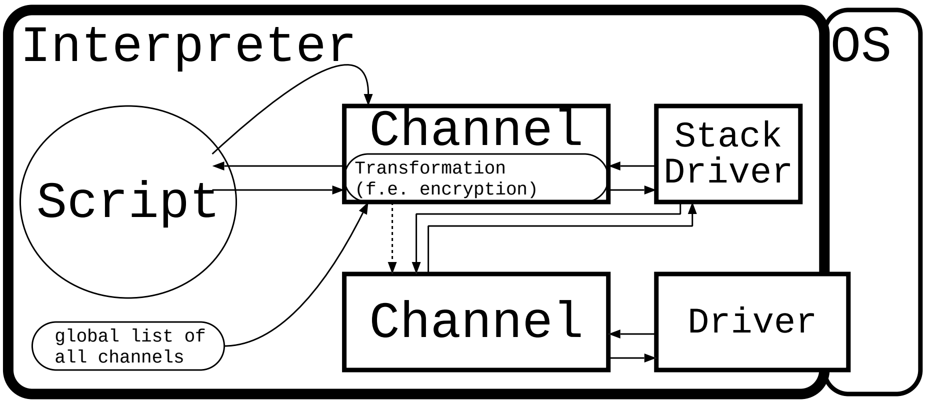


Figure 4: Current I/O system, with new feature

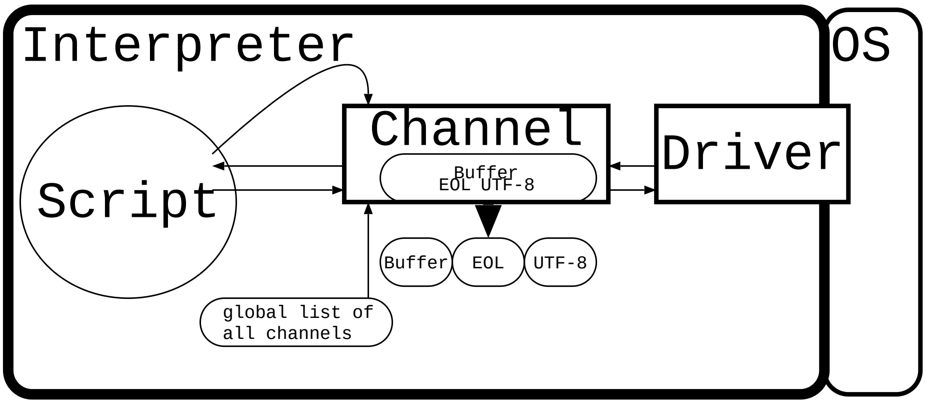


Figure 5: I/O system, internal transformations

will enable us to maintain this “feature” of the API.

- One of the differences between stacked channels and the kernel space *streams* is that the Tcl I/O system is completely stream oriented whereas *streams* allows the handling of message oriented protocols as well, for example `UDP` or `X.25` and all the other lower-level protocols below `IP`. We can emulate this by packing and unpacking messages on every level of transformations stacked upon a channel but this replicates work and wastes resources. It also means that every transformation has to exist in multiple variants for every message-based protocol which might be used below it.

  Adding message-based interfaces to channels in the future seems to be the way to avoid this.

  Usage of such interfaces is not restriced to `UDP`. Another example are the protocols used by Palm Pilots to communicate via serial interfaces.

## 6 Acknowledgements

I am indebted to Jacob Levy[8], Jan Nijtmans[7] and Matt Newman[6] for their support and ideas while this feature was in its proposal and patch phases.

## References


[1] Dennis Ritchie[5], *A Stream Input-Output system*, http://cm.bell-labs.com/cm/cs/who/dmr/st.html, http://cm.bell-labs.com/cm/cs/who/dmr/st.ps, The orriginal paper about “streams”.

[2] *Programmer’s Guide: STREAMS*, UNIX System V Release 4, UNIX Press, ISBN 0-13-02-0660-1

[3] *STREAMS Modules and Drivers*, UNIX System V Release 4.2, UNIX Press, ISBN 0-13-066879-6

[4] Stephen Rago, *UNIX System V Network Programming*, Addison-Wesley, ISBN 0-20-156318-5

[5] Dennis Ritchie, <dmr@bell-labs.com> (http://cm.bell-labs.com/cm/cs/who/dmr/)

[6] Matt Newman, <matt@sensus.org> (http://www.sensus.org)

[7] Jan Nijtmans, <j.nijtmans@chello.nl> (http://www.purl.org/net/nijtmans)

[8] Jacob Levy, <jyl@best.com>

[9] Marshall Rose, <mrose@dbc.mtview.ca.us>

[10] John R. LoVerso, <John@LoVerso.Southborough.MA.US> (http://www.schooner.com/~loverso)

[11] Brent Welch, <welch@scriptics.com> (http://www.beedub.com/)

[12] `TLS`, a binding of SSL/TLS for Tcl, uses `OpenSSL`[19], see http://www.sensus.org/tcl, by Matt Newman[6].

[13] `Tcl MIME`, a MIME-compliant mail handling package, see http://www.purl.org/NET/akupries/soft/mail/, by Marshall Rose[9].

[14] `TclHttpd`, a web server written in Tcl, see http://www.scriptics.com/products/tclhttpd/, by Brent Welch[11].

[15] `Comm`, an implementation of `send` using sockets, see http://www.schooner.com/~loverso/tcl-tk/#comm, by John R. LoVerso[10].

[16] `Trf`, see http://www.purl.org/NET/akupries/soft/trf/

[17] `TrfCrypt`, see
`http://www.purl.org/NET/akupries/soft/trfcrypt/`

[18] CVS, a tool for configuration and version management of source code. See
`http://www.cyclic.com/cvs/info.html`.

[19] `OpenSSL`, a free implementation of the SSL/TLS security protocol, see
`http://www.openssl.org`

[20] `zlib`, a free compression library, see
`http://www.cdrom.com/pub/infozip/zlib/`

[21] `bzip2`, a free compression library, see
`http://sourceware.cygnus.com/bzip2/index.html`