# Standalone Executables with Wrap

*By Jan Nijtmans*

## Abstract

One of the main problems with the distribution of Tcl/Tk applications is that it relies on Tcl/Tk being installed at the user's site. In this paper an approach is presented which allows the generation of standalone executables which are much smaller and faster than other approaches. The generation process does not require the use of a C-compiler, which makes it a very fast procedure. The method works on any Windows or Unix machine.

## Introduction

Wouldn't it be nice to be able to pack a group of Tcl/Tk scripts in a single executable, put it on a floppy, and hand it out to anyone interested to try it? Suppose one of the users is a developer who finds a bug, or simply wants to extend it. He extracts the relevant script from the executable, changes it, and places it back in the executable. Another user wants to add some more fancy dialogs which are offered by an extension (e.g. Tix or BLT). No problem, just pack the necessary dll's in the executable together with the modified scripts. The result is again a new standalone executable which can run without the need for installation. All of this can be done without the need for a compiler. Any desired change can be performed in minutes.

Someone else wants to distribute an application, but doesn't want other to be able to peek in the source code. This can be done by compiling the scripts in bytecodes, and pack the bytecodes in the executable. Further protection can be reached by encrypting the scripts. Only people who know the password can unpack the scripts. Of course, any reasonable experienced developer will be able to break this protection, because in order to run it must be decrypted anyway, but it is a barrier.

Another person wrote a windows application and needs a suitable installer. Suppose Inno Setup doesn't offer sufficient flexibility and WISE or InstallShield are too expensive. Then Wrap could be an option.

Although the above examples mainly involve Windows, the same approach works on UNIX as well. The only platform where it probably doesn't work this way is the Macintosh.

## Idea behind Wrap

Wrap is based on the assumption that executables have their headers in the beginning of the file while ZIP-files have easily distinguishable markers at the end. If data is glued after an executable, the executable can still be run without sacrificing the memory use or speed. Equally, ZIP-files can be prepended by any data still allowing the file to be handled as a ZIP-file. Many utilities are already available to handle ZIP-files, most notably Info-Zip's "zip" and "unzip" or Nico Mak Computing's "WinZip".

ZIP-files have many useful features:

- Standard compression library available (zlib), which can easily be incorporated in any application. Many platforms (e.g. Linux) already have this library installed by default.
- Encryption support available. Although this doesn't give the most protection you can think of, in combination with compression and bytecode compilation it can be made very hard to break.

It seems that the ZIP format is ideal to be used as file format, in combination with a platform-specific executable format.

## How to create a Wrapped executable

The first step in creating a wrapped executable is to create the executable itself. Generally, this executable is tclsh or wish, linked together with the Wrap extension. Then a 22-byte footer is attached to the executable, basically just an empty zip-file. This attachment activates the executable such that it starts to behave like a zip-file. This opens the way to start attaching any other file to the executable. The only problem left is how to make those extra zip-entries useful. That's where the Wrap extension comes in.

The Wrap extension contains all functionality needed to unpack ZIP entries. The public domain zlib library is contained into Wrap. Apart from the zip-entries from the executable, Wrap can also be used to open and read entries from any other zip-file.

## How is this different from other methods

Other known ways to create standalone executables are:

- mktclapp
- freewrap
- TclPro Wrapper

Freewrap and TclPro Wrapper work in about the same way as Wrap. They, too, consist of an executable followed by multiple entries of other types of data. The main two differences are that they don't use the zip-format for the entries and they are not usable as a separate extension. Freewrap currently doesn't compress the attached data (although that could be implemented in a future version).

Mktclapp deviates from the other methods, in that a C-compiler is needed to create the final executable. Data is attached to the executable by compiling it as C-string's. The advantage of this is that it makes it a lot easier to compile C-code and Tcl-code in a single executable.

TclPro Wrapper is most advanced of all, but what else would you expect from a commercial piece of software. Being not open-source, TclPro is most ideal for people who want to hide there source-code from users. Because of this, it is intentionally made almost impossible to unpack an executable, thereby retrieving the original data. Further on, TclPro contains a bytecode compile which makes it even more complicated to retrieve the original source code.

Two features unique to Wrap which are not found in other solutions are:

Cross-creation Because the generation process doesn't require a compiler, you can create Windows executables on unix or unix executables on Windows. Assuming that you pre-built the Wrap executables first on the target machine and that you have the zip utility available on the development machine. After that, all different platform executables can be built on one machine regardless which system it runs.

Loadable extension Al functionality of Wrap is available from Tcl/Tk as well using Wrap as a loadable extension. This allows Wrap to handle scripted documents.

The idea of Scripted documents comes from TclKit, which greatly motivated the Wrap development. See reference below The intention of Wrap is not to hide source-code, but to make it as easy as possible to unpack, modify and re-pack it. Although Wrap has an obfuscation function as well, anyone who knows the password can unpack it. The security in this way can never be as high as with TclPro.

## Code organisation.

Currently, Wrap consists of 6 parts:

wrapRsrc.c Contains the definition of the "rsrc" object type representing a zip-file. This object cashes the entry table, which is optimised for maximum speed. Any application wanting to open a zip-entry can quickly search for it in a hash-table. Opening it merely means seeking to the specified location and then starting to read the content.

wrapChan.c This module comes in view after the zip-entry is opened. It contains the implementation of a read-only Tcl channel.

wrapInit.c This file is responsible for the initialization, and also contains the Tcl commands supplied by Wrap.

wrapMain.c This file supplied the replacement for Tcl_Main() or Tk_Main(), meant for standalone executables.

wrapComp.c In development. The interface to the Tcl ByteCode compiler. Wrap will in the future provide a ByteCode compiler similar as the one provided by TclPro.

wrapRun.c In development. This part will form a bytecode executer matching the compiler.

Wrap can be built in two ways: as static executable or as dynamically loadable library. In the first form, the result will be two executables "tclsh83s.exe" and "wish83s.exe" (Windows) or "tclsh8.3s" and "wish8.3s" (unix). In the second form the result is a loadable library "wrap04.dll" (Windows) or "wrap0.4.so" (unix) which can be loaded in a normal tclsh or wish with "package require Wrap"

## Commands

The main commands in Wrap can be divided in two groups:
::wrap::load
::wrap::open
::wrap::exec
::wrap::glob
::wrap::file
::wrap::source
These command work as much as possible identically as the corresponding Tcl commands, except they operate on zip data in stead of files from the machine's own file system.

::wrap::index load an index file into memory

::wrap::run execute bytecode

Further on, Wrap has the following internal variables:

::wrap::version contains the version number of Wrap

::wrap::patchLevel contains the patch level of Wrap

::wrap::library holds the path that should be added to auto_path. On Windows this is normally `c:\Program Files\Tcl\lib\wrap0.4`

::wrap::temp holds the name of the system temporary directory used for caching. On Windows NT this is usually `c:\winnt\temp`

## Conclusion

Wrap can be seen as a Free companion of TclPro Wrapper. However, where TclPro wrapper concentrates on hiding the source code, Wrap's goal is to make creating, modifying and unpacking Standalone Executables as easy as possible. In this way it is possible to create an appliation using Tcl/Tk, and distribute it to your customers as a single standalone executable which doesn't require installation.

## References

Wrap homepage `http://purl.oclc.org/net/nijtmans/wrap.html`

TclPro Wrapper `http://dev.scriptics.com/software/tclpro/wrapper.html`

FreeWrap `http://home.nycap.rr.com/dlabelle/freewrap/freewrap.html`

mktclapp `http://www.hwaci.com/sw.mktclapp/index.html`

Inno Setup `http://www.jordanr.dhs.org/`

WISE `http://www.wisesolutions.com/default.htm`

TclKi `http://www.equi4.com/tclkit/news.html`

---

Jan Nijtmans
CMG, Oost-Nederland B.V.
`http://purl.oclc.org/net/nijtmans/`