freeWrap

6.76

Documentation

2/12/2025

Build stand-alone TCL/TK executable files. No compiler required!

OR

Use it as a single-file WISH shell

freeWrap 6.76 Documentation

Table of Contents

freeWrap 6.76 Documentation

freeWrap 6.76 Documentation

freeWrap License

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Other license information

FreeWrap borrows existing source code from several sources. To meet license terms and Copyright notice requirements associated with some of this code the following information is provided from the original source code distributions.

Dyncall license

If not stated otherwise inside a file, all files here are distributed in terms of:

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

TCL license

This software is copyrighted by the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, ActiveState Corporation and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors

and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7014 (b) (3) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

TCLLIB license

This software is copyrighted by Ajuba Solutions and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS

PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

TK license

This software is copyrighted by the Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation, ActiveState Corporation, Apple Inc. and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (b) (3) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

TKLIB license

This software is copyrighted by Ajuba Solutions and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES,

INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, ITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227-7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

Tktable license

* COPYRIGHT AND LICENSE TERMS *

(This file blatantly stolen from Tcl/Tk license and adapted - thus assume it falls under similar license terms).

This software is copyrighted by Jeffrey Hobbs <jeff at hobbs org>. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT

OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

RESTRICTED RIGHTS: Use, duplication or disclosure by the U.S. government is subject to the restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software Clause as DFARS 252.227-7013 and FAR 52.227-19.

SPECIAL NOTES:

This software is also falls under the bourbon_ware clause v2:

This software is free, but should you find this software useful in your daily work and would like to compensate the author, donations in the form of aged bourbon and scotch are welcome by the author. The user may feel exempt from this clause if they are below drinking age or think the author has already partaken of too many drinks.

TWAPI license

Copyright (c) 2003-2012, Ashok P. Nadkarni All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- The name of the copyright holder and any other contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

winico license

1. The "Software", below, refers to the "winico" extension for Tcl, developed by Brueckner&Jarosch (in either source-code, object-code or executable-code form), and related documentation, and a "work based on the Software" means a work based on either the Software, on part of the Software, or on any derivative work of the Software under copyright law: that is, a work containing all or a portion of the "winico" extension, either verbatim or with modifications. Each licensee is addressed as "you" or "Licensee."
2. Brueckner&Jarosch holds copyrights in the Software. The copyright holder reserves all rights except those expressly granted to licensees, and German Government license rights.
3. Permission is hereby granted to use, copy, modify, and to redistribute to others. If you distribute a copy or copies of the Software, or you modify a copy or copies of the Software or any portion of it, thus forming a work based on the Software, and make and/or distribute copies of such work, you must meet the following conditions:
    1. If you make a copy of the Software (modified or verbatim) it must include the copyright notice and this license.
    2. You must cause the modified Software to carry prominent notices stating that you changed specified portions of the Software.
4. Disclaimer of warranty: Licensor provides the software on an “as is" basis. Licensor does not warrent, guarantee, or make any representations regarding the use or results of the software with respect to its correctness, accuracy, reliability or performance. The entire risk of the use and performance of the software is assumed by licensee. ALL WARANTIES INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR MERCHANTABILITY ARE HEREBY EXCLUDED.
5. Lack of maintenance or support services: Licensee understands and agrees that licensor is under no obligation to provide maintenance, support or update services, notices of latent defects, or correction of defects for the software.
6. Limitation of liability, indemnification: Even if advised of the possibility of damages, under no circumstances shall licensor be liable to licensee or any third party for damages of any character, including, without limitation, direct, indirect, incidental, consequential or special damages, loss of profits, loss of use, loss of goodwill, computer failure or malfunction. Licensee agrees to indemnify and hold harmless licensor for any and all liability licensor may incur as a result of licensee's use of the software.

Info-ZIP license

This is version 2009-Jan-02 of the Info-ZIP license. The definitive version of this document should be available at

ftp://ftp.info-zip.org/pub/infozip/license.html indefinitely and a copy at

http://www.info-zip.org/pub/infozip/license.html.

freeWrap 6.76 Documentation

freeWrap 6.76 Documentation

Overview

The freewrap program turns TCL/TK scripts into single-file binary executable programs.

The resulting program can be distributed to machines that do not have TCL/TK installed. The executable will also work on machines that have TCL/TK installed but will use its own TCL/TK "image". freeWrap itself does not need TCL/TK installed to run.

Easy, one-step wrapping.

FreeWrap consists of a single executable file. There is no setup required. Wrapping is accomplished with a single command.

Your source and data files are protected from prying eyes.

FreeWrap automatically encrypts all files you wrap into your executable application to provide a secure distribution.

freewrapTCLSH can be used to wrap TCL-only scripts.

FreewrapTCLSH creates a single executable file from a TCL script. The wrapping syntax is identical to the freewrap program. This produces a console-only type of program.

freeWrap can be used as a single file stand-alone WISH

Renaming the freeWrap program to some other file name causes freeWrap to behave as an a stand-alone, single-file WISH that can be used to run any TCL/TK script or a freeWrap package containing all the files in your application.

freewrapTCLSH can be used as a single file stand-alone TCLSH shell

Renaming the freewrapTCLSH program to some other file name causes freewrapTCLSH to behave as a stand-alone, single-file TCLSH shell that can be used to run any TCL script or a freeWrap package containing all the files in your application.

Shared libraries can be used with your wrapped programs.

FreeWrapped applications can load TCL/TK shared library extensions that have been compiled with the STUBS interface.

Your wrapped programs can be customized with your own window icons.

The Windows version of freeWrap can incorporate your own customized icon into your wrapped application.

No license fees for wrapped programs.

There are no license fees associated with freeWrap. See the freeWrap license.

Cross-platform generation of programs is supported.

The -w "wrap using" option allows cross-platform creation of wrapped applications without the use of the target computer system.

freeWrap includes several Windows-specific commands

These commands can be used to determine the location of Windows' special directories and make for easy creation of file extension associations and shortcuts.

freeWrap includes commands for ZIP file creation and extraction.

Due to freeWrap's use of the ZIP Virtual File System any ZIP archive can be opened so its contents look like a simple file subdirectory. The archive's files are automatically decompressed when read with TCL commands.

The makeZIP command allows creation and modification of ZIP archives from within your freeWrapped application.

freeWrap 6.76 is based on TCL/TK version 8.6.12

This version of freeWrap is built with threading enabled.

Availability

FreeWrap 6.76 executable files are freely available for 32-bit and 64-bit versions of Linux and Windows. Instructions and source code for building freeWrap are included in the freeWrap source code distribution.

TCL-only versions of freeWrap are also available for wrapping TCL (non-TK) scripts.

Visit http://sourceforge.net/projects/freewrap to download files.

What's Inside

TCL/TK

The freeWrap binary distributions contain TCL/TK version 8.6.13.

FreeWrap comes with its own internal file system which contains the files that normally make up a TCL/TK distribution. From within freeWrap the files normally found at ***/usr/local/lib*** can be accessed from the **/zvfs** directory. For example, the **/usr/local/lib/tdbc1.1.5** directory can be accessed as **/zvfs/tdbc1.1.5** within freeWrap.

The following TCL/TK directories are available within freeWrap:

/zvfs/itcl4.2.3
/zvfs/sqlite3.40.0
/zvfs/tcl8
/zvfs/tcl8.6
/zvfs/tcllib1.21
/zvfs/tdbc1.1.5
/zvfs/tdbcmysql1.1.5
/zvfs/tdbcodbc1.1.5
/zvfs/tdbcpostgres1.1.5
/zvfs/thread2.8.8
/zvfs/tk8.6
/zvfs/tklib0.7

TCL/TK documentation can be found at http://www.tcl-lang.org/man/tcl8.6/.

TCL/TK extensions and libraries

The freeWrap binary distributions also contain the following additional TCL/TK extensions and libraries.

Extension/Library	Version
tcllib	2.0
tklib	0.8
Tktable	2.10
TWAPI	5.0.2
winico	0.6

tcllib documentation can be found at https://core.tcl-lang.org/tcllib/doc/trunk/embedded/md/toc.md.

The **tcllib** library files are included within freeWrap at the following virtual directory.
/zvfs/tcllib1.20

tklib documentation can be found at https://core.tcl-lang.org/tklib/doc/trunk/embedded/www/toc.html.

The **tklib** library files are included within freeWrap at the following virtual directory.
/zvfs/tklib0.7

Tktable documentation can be found at http://tktable.sourceforge.net/tktable/doc/tkTable.html.

Tktable support files are included within freeWrap at the following virtual directory.
/zvfs/TkTable.

TWAPI documentation can be found at https://twapi.magicsplat.com/v4.7/index.html.

TWAPI support files are included within freeWrap at the following virtual directory.
/zvfs/twapi

winico documentation can be found at http://tktable.sourceforge.net/winico/winico.html.

freeWrap as a TCL/TK wrapper program

FreeWrap can wrap TCL/TK applications that consist of multiple script and binary image files. FreeWrap combines all the files into a single executable file. The syntax for wrapping an application is described below.

Calling Syntax:	freewrap mydir/prog.tcl [-debug] [-f FileLoadList] [-forcewrap] [-i ICOfile] [-o OutFile] [-p] [-w freewapProg] File1 ... FileN	
where:	**mydir/prog.tcl**	file path to main TCL/TK program script
	File1 ... FileN	A list of space-separated text or binary files to include in the wrapped application.
	-debug	Opens a console window so user can see debug messages while wrapping.

Calling Syntax:	freewrap mydir/prog.tcl [-debug] [-f FileLoadList] [-forcewrap] [-i ICOfile] [-o OutFile] [-p] [-w freewapProg] File1 ... FileN	
	-f	Specifies that the following named file (FileLoadList) contains a list of files to wrap
	-forcewrap	Force freeWrap to act as a wrapping program even if it has been renamed.
	-i	Substitute the following named Windows ICO file (ICOfile) as the program application icon.
	-o	Indicates that the name of the produced executable program should be set to OutFile. This may be a full or relative file path.
	-p	Create a freeWrap program package instead of an executable program.
	-w	Specifies that the following named freeWrap program should be used to create the wrapped application. This option allows building an application targeted for execution on a different operating system. Example usage: /apps/freewrap myprog.tcl -w /winapps/win64/freewrap.exe In this example a Linux version of freeWrap is used to produce a wrapped application for 64-bit Windows.
	-0 **-1** **-2** **-3** **-4** **-5** **-6** **-7** **-8** **-9**	These ten options control the amount of compression performed on the wrapped files in a manner similar to the ZIP application. **-0** performs no compression, **-1** performs the least but is the fastest, while **-9** performs the most compression and takes the longest time.
output:	prog (Linux) or prog.exe (Windows) **Note**: the output file will be placed in the directory from which freeWrap is called.	

The names of the files being wrapped may include either relative or full paths. The resulting executable program can access the wrapped files by either referring to them by their full path as they existed at the time of wrapping or adding the paths to TCL's ***auto_path*** variable. If the ***auto_path*** method is used, the appropriate *tclIndex* or *pkgIndex.tcl* files should also be wrapped into the application. Please see the information on how to wrap a package extension. Please see Naming and referring to wrapped files for more details on how to refer to wrapped files within the application.

Both text and binary files can be wrapped.

-debug Option

Use of the -debug option will display the freeWrap console window so any debug warning messages can be viewed while wrapping.

-f Option

For larger wrap projects, the user may wish to use freeWrap's -f option to specify a file which contains a list of files to wrap. The specified text file must contain one file name per line. Each file name listed in the file will be added to the wrapping. Use of the -f option does not preclude the specification of individual files on the freeWrap command line. The -f option may also be used several times on the same command line.

Example: *freewrap myprog.tcl logo.gif -f projlist.txt code2.tcl -f special.txt*

-forcewrap Option

The -forcewrap option can be used to force freeWrap to act as a wrapping program even if it has been renamed. Without this command line option, freeWrap behaves like a WISH shell when it has been renamed.

-i Option

This option will replace the freeWrap icon within your wrapped application with the contents of the specified ICO file. Use the -i option to specify the icon you wish to use for your wrapped application. This option is only relevant when wrapping an application for the Windows operating system.

Example: *freewrap myprog.tcl -i myprog.ico*

In this example, *myprog.ico* must be an ICO formatted file. Each icon in this file must be stored as a BMP image. Icons in the PNG format will not be replaced. The -i option will only replace an icon if the new version is exactly the same size as the original icon in freeWrap. PNG files are compressed and, therefore, a new PNG formatted icon is unlikely to be the same size as the original freeWrap icon of the same dimensions and color depth.

When creating your ICO file, keep in mind that freeWrap contains the following versions of the freeWrap icon in a BMP format.

Size	Colors
16x16	16
16x16	256
32x32	4-bit
32x32	8-bit
32x32	24-bit
48x48	4-bit
48x48	8-bit
48x48	24-bit
128x128	24-bit

Size	Colors
256x256	24-bit

Icons, of these resolutions, found in your ICO file will be used to replace the freeWrap icons. If your ICO file doesn't contain at least these versions of your icon then only the matching icons will be replaced. This would leave a freeWrap icon that could be displayed by Windows at some time.

-o Option

The -o Option allows you to specify the name of the executable program you are creating/wrapping. The option value may be either single file name, a full file path, or a relative file path.

-p Option

Using the -p option creates a wrapped application without the freeWrap executable component. This file is called a freeWrap program package. A freeWrap program package can be run using freeWrap as a single-file shell By default, freeWrap program packages are given a file extension of *fwp*.

Example wrapping: freewrap myapp.tcl -p

Example execution: freewish myapp.fwp

-w Option

By default, freeWrap attaches the wrapped files to a copy of the freeWrap program you use to do the wrapping. The -w option allows attaching the wrapped files to a different copy of freeWrap. Since freeWrap is available for multiple operating systems, this feature is useful for assembling freeWrapped applications for other operating systems while on a single computer.

Example (assembling a Windows version while running freeWrap on Linux):

freewrap myprog.tcl -w freewrap.exe

Example (assembling a Linux version while running freeWrap on Windows):

freewrap myprog.tcl -w freewrap

Remember, the argument following the -w option must be the file path to a version of the freeWrap program that can execute on the other operating system.

freeWrap as a single-file WISH interpreter

Renaming the freeWrap program to some other file name causes freeWrap to behave as an a stand-alone, single-file WISH that can be used to run any TCL/TK script or freeWrap program package. This can be done in the following manner.

Copy freewrap.exe to a new file name

Example: *copy freewrap.exe wishrun.exe*

Use the new file as you would normally use WISH

Example: *wishrun script_name.tcl*

freeWrap response to untrapped errors

FreeWrap will respond to untrapped script errors in the following manner:

Application Type	***Response to Untrapped Script Error***
Wrapped console applications (created with freewrapTCLSH)	Wrapped application prints resulting error message to standard error stream.
Wrapped graphical applications (created with freewrap)	Wrapped application displays message window which must be acknowledged before the user can interact with the application again.
Running a script from the command line as a single-file TCLSH interpreter. Example: freewishTCLSH myErrorScript.tcl	The associated error message will print to the standard error stream.
Running a script from the command line as a single-file WISH interpreter. Example: freewish myErrorScript.tcl	Errors that occur before TK enters its event loop will cause the TK console to open and display the error message. An example of this case can be seen in the following line of code which is not contained in a procedure and executes immediately when starting the script. *puts Hello World* Errors resulting from delayed code execution display a message window which must be acknowledged before the user can interact with the application again. Examples of this case can be seen in the following lines of code. *after 3000 {puts Hello World}* *proc sayHello {} { puts Hello World }*
Sourcing a script when interactively running as a single-file TCLSH interpreter.	The associated error message will print to the standard error stream. Example session: *% source dtest1.tcl*

Application Type	Response to Untrapped Script Error
	can not find channel named "Hello" *%*
Sourcing a script when interactively running as a single-file WISH interpreter.	Errors that occur before TK re-enters its event loop will print the error message to the TK console. An example of this case can be seen in the following line of code which is not contained in a procedure and executes immediately when sourcing the script. *puts Hello World* Errors resulting from delayed code execution display a message window which must be acknowledged before the user can interact with the application again. Examples of this case can be seen in the following lines of code. *after 3000 {puts Hello World}* *proc sayHello {} { puts Hello World }*

freeWrap program packages

FreeWrap normally produces an executable file when wrapping an application. However, it is also possible the create a file that only contains the wrapped files for the application. This allows you to distribute smaller packages that can later be run using freeWrap as a single-file TCLSH or WISH interpreter. A freeWrap program package can contain all the files for your application in a single compressed file.

Use the -p option when wrapping your application in order to create a freeWrap program package instead of an executable file.

Example wrapping: freewrap -f listOfiles.txt myapp.tcl -p

Example execution: freewish myapp.fwp

FreeWrap program packages are not encrypted and can be run using a copy of freeWrap 6.3 or later..

freeWrap's console window

Under freeWrap, the ***console*** command is available for both Windows and UNIX. The console window is the location that will receive any STDOUT or STDERR output. The console can also be used to interactively enter TCL/TK commands. Use ***console show*** to display the window and ***console hide*** to remove it.

Using the DDE and Registry packages (Windows only)

The DDE and Registry packages have been compiled into freeWrap. There is no need to load them with a ***package require*** command. Simply use the ***dde*** and ***registry*** commands without any preceding ***package require*** command.

Using wrapped files.

Wrapping

When running a wrapped application, the first file specified on the command line at the time of wrapping will be executed as a TCL/TK script. All other files specified on the command line or in a file load list are available to this executing script.

You CAN do the following with the wrapped files.

1. ***source*** them
2. ***open*** them
3. ***read*** them
4. ***close*** them
5. ***glob*** them
6. **load** them
7. Use any ***file*** commands that do not write to the files
8. Use them with the ***image create*** command
9. Specify them for ***-bitmap*** widget options.

You CANNOT do the following with the wrapped files.

1. ***File delete*** them (since they exist in the application, not on disk)

Naming and referring to wrapped files

Details

All files included in a wrapped application must be referred to by their full path within the application. However, any relative or full path specification can be used on the freeWrap command line.

Windows users will notice that freeWrap strips all drive letter information from a file's path prior to storing it inside the wrapped application.

When referenced inside the wrapped program, the paths to the wrapped files are also prefixed by */zvfs* and will have no drive letter. To the wrapped application, all of its internal files will appear to be under the */zvfs* directory.

For example, if an application is wrapped to include the file C:\projects\myproject\libmodule1.tcl with the following command:

freewrap myapp.tcl libmodule1.tcl

You would need to use a source command within the application such as:

source /zvfs/projects/myproject/libmodule1.tcl

DO NOT expect the relative path of wrapped files to change when you move the executable program.

FreeWrap takes a "snapshot" of the file path for all wrapped files. You must use the same, full path (minus any drive letter) that existed at the time of wrapping to refer to the wrapped file. It is also important that the file paths you use in your program exactly match the letter case that exists at the time of wrapping.

These rules also apply to the ***file*** or ***open*** commands. Also, make sure you add a path to auto_path which corresponds to the wrapped tclIndex or pkgIndex.tcl files you include in your application. For example, if your wrapping command is:

freewrap myapp.tcl c:\devel\myapp1\tclIndex c:\devel\myapp1\libmodule1.tcl

you should add /zvfs/devel/myapp1 to auto_path.

The easy way

At run time, your application can use the ::zvfs::list command to determine exactly where the files are in the virtual directory structure. For example:

```
set filePath [::zvfs::list */u1.tcl]
```

You may also wish to add some code to your application that enables it to find the necessary file either when developing or when wrapped. For example:

```
set filePath {}
if {[namespace exists ::freewrap]} {
    if {$::freewrap::runMode eq {wrappedExec}} {
        set filePath [::zvfs::list */u1.tcl]
       } {
           set filePath $::env(SrcPath)/utils/u1.tcl
         }
} {
    set filePath $::env(SrcPath)/utils/u1.tcl
  }

if {$filePath eq {}} {
    puts {Could not find file u1.tcl}
   } {
       source $filePath
       # Place additional code here.
```

```
}
```

In summary

You should use the paths to the files as they exist at the time of wrapping and add /zvfs as the root directory to these paths. Wrapping takes a "snapshot" of the file path for all wrapped files. The **::zvfs::list** command can be used to make it easier to find the correct path within the wrapped application.

Do not use relative paths to refer to wrapped files within the application since relative paths will not be found.

Wrapping and using TCL/TK extensions (packages)

TCL/TK extensions can be wrapped into your application and then loaded dynamically at run time. Alternatively, if you are willing to recompile freeWrap, TCL/TK extensions may also be statically compiled into freeWrap. See http://sourceforge.net/projects/freewrap for versions of freeWrap that already include some statically compiled TCL/TK extensions.
Wrapped applications can load TCL/TK shared binary extension that have been compiled with the new TEA (i.e., stubs) interface. Stubs-enabled shared libraries can be included in the wrapped application or exist as separate files.

TCL's package search mechanism uses the ***glob*** command to recursively search directories specified in the auto_path variable to find packages. Unfortunately the ***glob*** command does not do the same for Virtual File System (VFS) files or their directories. This means TCL's package require command will not descend subdirectories when searching for packages. However, the fix for this is simple. Add the desired package's file path to the auto_path variable before using the package require command. This can be done with two lines of code similar to:

```
lappend auto_path /zvfs/tcl/lib/mypkg1.0 ;# Ensure our app can find the files
package require mypackage
```

Script only extensions

Packages consisting only of TCL/TK scripts are generally easy to wrap.

As an example, let us consider the BWidget 1.8 extension. Under Windows the following short batch file could be used to wrap a sample program using BWidgets named BWidget_demo.tcl.

```
REM wrapBWidget.bat file
dir /S /B .\BWidget1_8 >Bwidget_files.txt
START /WAIT freewrap.exe BWidget_demo.tcl -f Bwidget_files.txt
```

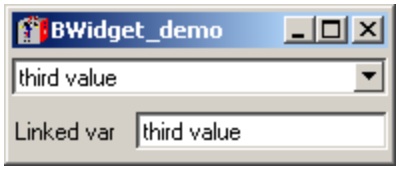
These batch commands create a text file containing the list of files that make up the extension (one file name per line). These commands also assume that the BWidget package has been installed in the BWidget1_8 directory immediately below the current directory.

For our example, the BWidget_demo.tcl file if found in the current directory and contains the following TCL/TK commands.

```
lappend ::auto_path [file dirname [zvfs::list */Bwidget1_8/pkgIndex.tcl]]
package require BWidget

set combo [ComboBox .combo \
   -textvariable comboVal \
   -values {"first value" "second value" "third value" "fourth value" "fifth value"} \
   -helptext "This is the ComboBox"]

set ent [LabelEntry .ent -label "Linked var" -labelwidth 10 -labelanchor w \
   -textvariable comboVal -editable 0 \
   -helptext "This is an Entry reflecting\nthe linked var of ComboBox"]

pack $combo $ent -pady 4 -fill x
```

The first line of this sample script allows the [package require BWidget] command to find the BWidget extension.

Running this wrapped application will produce the following pop-up window.



Extensions containing a single binary file

TCL extensions with a single binary file (other script files may be present) are also easy to wrap if:

1. The binary library does not call other binary libraries located inside the wrapped application.
2. The binary library does not call scripts located inside the wrapped application.

Your application code simply needs to add the path to the binary extension's pkgIndex.tcl file to the auto_path variable so that the usual ***package require*** or ***load*** commands will find the extension.

The **zvfs::list** command can be used to make setting auto_path easier. For example:

```
lappend ::auto_path [file dirname [zvfs::list */myExtension/pkgIndex.tcl]]
```

More complex extensions with both scripts and binary libraries

If the TCL extension contains more than one binary library or one of the libraries is dependent on other libraries wrapped into your application you will need to use something like the procedure found in the following example. This situation is rare and you may never need to do this. However, the Snack sound package is one such extension and we will use it for our example.

The Snack extension has a supporting script whose loading is normally automated by the [package require] command. We need to use the [package require] command but the associated pkgIndex.tcl file

doesn't really know how to find the supporting script or DLL.

This is easy to correct by making a slight modification to Snack's pkgIndex.tcl file so that it can find and load the DLL.

Here is a procedure for wrapping an application containing and using Snack version 2.2.10.

1. Modify the Snack pkgIndex.tcl file as follows:

   Replace the following single line in the pkgIndex.tcl file

   ```
   package ifneeded snack 2.2 "[list load [file join $dir libsnack.dll]];
   [list source [file join $dir snack.tcl]]"
   ```

   with (the following is a single line. It may suffer from wrap around)

   ```
   package ifneeded snack 2.2 "[list load [zvfs::list */libsnack.dll]];[list
   source [zvfs::list *$dir/snack.tcl]]"
   ```

   This modification will find and load the libsnack dynamic library as well the the snack.tcl file.

2. Add some code to the application to adjust the auto_path variable. We must ensure that all wrapped directories containing pkgIndex.tcl files are added to auto_path. For this project, the following code is added at the beginning of the application. It should be executed before the [package require] command.

   ```
   foreach fpath [zvfs::list */win/*/pkgIndex.tcl] {
               lappend auto_path [file dirname $fpath]
         }
   ```

   As you can see, with this code, the location of the files is determined at run time. We don't have to keep track of them to properly update the code. The [zvfs::list] command is very useful for locating your wrapped files.

3. Use a [package require snack] command in the application to load the Snack package.

4. FreeWrap the application making sure all the supporting DLL and script file are included.

Special variables and commands provided by freeWrap

The ::freewrap namespace

FreeWrap has a namespace which contains all of the freeWrap specific variables, commands and procedures. These variables, commands and procedures may be referenced using the ::freewrap:: prefix or imported into any other namespace.

Variables

The following variables are defined in the ::freewrap namespace of each wrapped application.

Name	***Description***
errormsg	This variable is set by the ::freewrap::unpack procedure when a file cannot be

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>written to the requested destination.</td></tr>
<tr><td>patchLevel</td><td>Revision level of the freeWrap program used to wrap the application.</td></tr>
<tr><td>progname</td><td>The proper name for the freeWrap program for the current operating system. This is normally freewrap.exe under Windows and freewrap under UNIX.</td></tr>
<tr><td>runMode</td><td>This variable indicates whether freeWrap is running as:
<table>
<tr><th>Value of variable</th><th>Meaning</th></tr>
<tr><td>interactiveShell</td><td>an interactive shell</td></tr>
<tr><td>programPackage</td><td>a wrapped application without the freeWrap executable core.</td></tr>
<tr><td>standAloneShell</td><td>a stand-alone shell running a script</td></tr>
<tr><td>wrappedExec</td><td>a wrapped executable program</td></tr>
</table>
</td></tr>
</table>

Procedures

The following procedures are defined in the ::freewrap namespace of each wrapped application. The commands names starting with shell_ are only available under the Windows operating system.

Syntax: isSameRev file_name

Description: Checks whether the specified file contains a copy of the same freeWrap revision as the currently executing program.

Returns: 0, if file does not contain a copy and 1, if file contains a copy.

Syntax: iswrapped file_name

Description: Determines whether the file named file_name is a freeWrapped application.

If file_name is a freeWrapped application this procedure returns a value of 1.

If file_name is NOT a freeWrapped application this procedure returns a value of 0.

Syntax: shell_assoc_exist extension

Description: Check whether a key exists for an extension

Example: shell_assoc_exist .txt => 1
Example: shell_assoc_exist .NEVER => 0

Syntax: shell_fileType_exist fileType

Description: Determine whether a file type exists

Example: shell_fileType_exist txtfile => 1
Example: shell_fileType_exist NEVER => 0

Syntax: shell_fileExtension_setup extension, fileType

Description: Creates a file extension and associates it with fileType.
Example: shell_fileExtension_setup .txt txtfile

Remove connection between extension and fileType
Example: shell_fileExtension_setup .txt ""

Syntax: shell_fileType_setup fileType, title

Description: Creates a file type.
Example: shell_fileType_setup txtfile "Text Document"

Syntax: shell_fileType_open fileType, openCommand

Description: Creates an open command. Sets action for double click.
Example: shell_fileType_open txtfile "C:\\WINDOWS\\NOTEPAD.EXE %1"

Syntax: shell_fileType_print fileType, printCommand

Description: Creates a print command for right mouse button menu.
Example: shell_fileType_print txtfile "C:\\WINDOWS\\NOTEPAD.EXE /p %1"}

Syntax: shell_fileType_icon fileType, icon

Description: Sets an icon for a fileType.
Example: shell_fileType_icon txtfile "C:\\WINDOWS\\SYSTEM\
\shell32.dll,-152"
Example: shell_fileType_icon txtfile "C:\\mydir\\myicon.ico"

We can give a name.ico file or a dll or exe file here. If a dll or exe file is used the index for the resource inside the file must be specified

Syntax: shell_fileType_quickView fileType, quickViewCmd

Description: Sets the command to execute to perform a quick view for a fileType.
Example: shell_fileType_quickView txtfile "write.exe %1"

Syntax: shell_fileType_addAny_cmd fileType, cmdName, cmd

Adds any command you want to a fileType.
Example: shell_fileType_addAny_cmd scrfile config "%1"

Syntax: shell_fileType_setMenuName fileType, cmdName, str

Description: Change description in right mouse menu for a command associated with a fileType.
Example: shell_fileType_setMenuName txtfile print "Print file"

Syntax: shell_fileType_showExt fileType, yesOrNo

Description: Always show the extension on the fileType.
Example: shell_fileType_showExt txtfile 1

Turn off "Always show" of extension on the fileType
Example: shell_fileType_showExt txtfile 0

Syntax: shell_fileType_setCmdOrder fileType, cmds

Description: Over-ride the default ordering of commands on right mouse menu.
Example: shell_fileType_setCmdOrder txtfile {print open}

Syntax: shell_fileType_neverShowExt fileType, yesOrNo

Description: Never show extension on fileType.
Example: shell_fileType_neverShowExt txtfile 1

Turn off "Never show" of extension on the fileType.
Example: shell_fileType_neverShowExt txtfile 0

Syntax: shell_getCmds file

Description: Retrieves all the commands associated with an extension.
Example: shell_getCmds file.txt => open print

Syntax: shell_getCmd_imp file, cmd

Description: Retrieves the implementation of a command given a file extension and command.
Example: shell_getCmd_imp test.txt open => C:\\WINDOWS\\NOTEPAD.EXE %1

Syntax: unpack file_name, [destdir]

Description: This function unpacks file_name from a freeWrapped application's ZVFS archive
into a file system directory. The destination directory for the file may be specified with the optional destdir argument. If this optional argument is not specified, the function will select a temporary directory appropriate for the operating system. Unpack, on success, returns the full path to the newly created file and on failure, an empty string. This function is useful for creating local copies of wrapped shared libraries (e.g., DLLs) that can then be loaded into your wrapped TCL/TK application.

Commands

The following TCL commands are defined in the ::freewrap namespace of each wrapped application.

Syntax: getSpecialDir dirType

Description: Find "Start Menu", "Desktop" and similar directory locations under Windows. DirType must be one of the following strings:

ALTSTARTUP	FONTS
APPDATA	HISTORY
BITBUCKET	INTERNET
COMMON_ALTSTARTUP	INTERNET_CACH
COMMON_DESKTOPDIRECTORY	NETHOOD
COMMON_FAVORITES	NETWORK
COMMON_PROGRAMS	PERSONAL
COMMON_STARTMENU	PRINTERS
COMMON_STARTUP	PRINTHOOD
CONTROLS	PROGRAMS
COOKIES	RECENT
DESKTOP	SENDTO
DESKTOPDIRECTORY	STARTMENU
DRIVES	STARTUP
FAVORITES	TEMPLATES

Syntax: makeZIP [-options] [-b path] [-t mmddyyyy] [-n suffixes] ZIPfile FileList [-xi list]

Description: This command duplicates most of the functionality of the Info-Zip application using the following syntax which is almost identical to the ZIP command line program.

The default action is to add or replace files in ZIPfile with the files specified by FileList. FileList is a space-separated list of file names.

The following optional arguments may be used.

-0	store only
-1	compress faster
-9	compress better
-A	adjust self-extracting exe
-b path	use this directory path for the temporary file
-d	delete entries in zipfile
-D	do not add directory entries
-f	freshen: only changed files
-F	fix zipfile (-FF try harder)
-i list	include only the following file names

-j	junk (don't record) directory names
-J	junk zipfile prefix(unzipsfx)
-l	convert LF to CR LF (-ll CR LF to LF)
-m	move into zipfile (delete files)
-n suffixes	don't compress these suffixes
-o	make zipfile as old as latest entry
-r	recurse into directories
-t mmddyyyy	exclude files earlier than the specified date
-u	update: only changed or new files
-x list	exclude the following names
-X	exclude extra file attributes
-y	store symbolic links as the link instead of the referenced file

Syntax: Nagle chanID [ON | OFF]

Description: Use this command to report or set the status of the Nagle algorithm for a TCP socket opened by TCL.

Argument	Explanation
chanID	The first command argument must be the name of a currently open socket returned by TCL.
state	This is the second, optional argument for the command. If this argument is not provided then the command returns the current status of the TCP socket specified by chanID. A return value of ON indicates that the Nagle algorithm is currently active for the socket. A return value of OFF indicates that the Nagle algorithm is not currently active for the socket. If the second supplied argument is ON or a TCL logical true value then the Nagle algorithm for the socket will be activated. If the second supplied argument is OFF or a TCL logical false value then the Nagle algorithm for the socket will be turned off.

If the second argument is present, the resulting status of the Nagle algorithm will be returned as the result.

Syntax: shortcut linkPath[-objectPath objectPath] [-description description] [-workingDirectory dir] [-icon path index] [-arguments args]

Description: Creates a Windows shortcut. The only required parameter is the linkPath. This means you can create a shortcut with no target, which probably isn't useful. The icon of the shortcut will default to the icon of the target item if not specified.

Argument	Explanation
linkPath	The path to the shortcut file (including the extension .lnk)
objectPath	The target of the link
description	Shortcut description
workingDirectory	Working (startup) directory for the target of the shortcut
path index (icon)	Specifies the path to a file and the index of the icon in that file to use for the shortcut
args	Arguments passed to the target of the shortcut when started.

Character encodings

All encoding files available with the TCL distribution have been compiled into the freeWrap application.

Under the Windows version of freeWrap, the ***encoding names*** command is not able to list your wrapped encoding files. However, the other ***encoding*** commands will work correctly. As an alternative, you could use the ***zvfs::list*** command to find the wrapped encoding files. For example:

```
set encFiles [zvfs::list /tcl/encoding/*]
```

The TCL encoding files included with freeWrap can be accessed by specifying their full ZVFS path when using one of the encoding commands. For example:

```
encoding system /tcl/encoding/cp850
```

The Tcl **source** command always reads files using the system encoding. A difficulty arises when distributing scripts internationally, as you don't necessarily know what the system encoding will be. Fortunately, most common character encodings include the standard 7-bit ASCII characters as a subset. Therefore, you are usually safe if your script contains only 7-bit ASCII characters.

If you need to use a specific character set for the scripts that you distribute, you can provide a small "bootstrap" script written in 7-bit ASCII. The bootstrap script should first set the system encoding to the desired value then **source** the desired script.

For example, the contents of the bootstrap script (named myprogram.tcl here) could be:

```
# Set the desired encoding first.
encoding system /tcl/encoding/cp850
# Now, let's run the real program.
source myrealprogram.tcl
```

You would then wrap your application using the following command:

```
freewrap myprogram.tcl myrealprogram.tcl
```

How freeWrap encryption works

FreeWrap now includes ZIP 2.0 style file encryption. FreeWrap encrypts all files stored into its internal ZIP Virtual File System. It also encrypts all files wrapped into a single-file executable application. The password required for handling the encrypted files is embedded into the wrapped program.

The password key is included as a function. Embedding the password as a compiled function makes it extremely difficult to deduce the password by looking at the executable file.

When a freeWrapped application is run the freeWrap core of the program will check to ensure that the wrapped files are still encrypted with the original key. If one of the

wrapped files has been replaced the freeWrapped application will detect this tampering and refuse to run.

FreeWrap can easily be built to use a different embedded password that no one else has access to. This is important for generating wrapped applications where you wish to prevent others from viewing the source code. Remember, due to freeWrap's ability to mount ZIP files as a subdirectory, a person having the same version of freeWrap (with the same password) can easily read the encrypted files within your application. Therefore, those people interesting in securing the files that make up their application should use a copy of freeWrap (built with its own unique password) that no one else has access to.

FreeWrap program packages, however, are not encrypted since doing so would require distribution of the copy of freeWrap used to create the package in order to run them. Such a copy of freeWrap could then be easily used to defeat the encryption of the wrapped package.

Building freeWrap

FreeWrap does not include any alterations to the TCL/TK core. It is plain-vanilla TCL/TK with a few extensions such as Tktable thrown in.

Dependencies

Compilation of the freeWrap executable programs requires the following additional libraries not provided with the freeWrap distribution:

1. TCL (Tool Command Language) static library
2. TK (Tool Kit for TCL) static library
3. Zlib compression static library
4. Info-ZIP compiled object files
5. Tktable extension
6. tcllib and tklib
7. TWAPI (TCL Windows API extension) - for Windows builds only

The following helper applications are needed as part of the freeWrap build process and will be called by the Make files. These programs should be accessible from the directories in which freeWrap is compiled.

1. tclsh (used to run the setinfo.tcl and libraryCopy.tcl scripts)
2. info-ZIP (used to attach zip archive to end of freeWrap program)

Build environment

Current distribution

The freeWrap 6.76 executable program distributions were built under the following environments:

64-bit Linux

Built on OpenSUSE 42.3 64-bit.

Should run on any recent 64-bit Linux distribution.

64-bit Windows

Built on Windows 11 using the MSYS2 platform with the mingw-w64 tool chain targeting 64-bit Windows. The following MSYS2 installer was retrieved from www.msys2.org:

msys2-x86_64-20241116.exe

MSYS2 requires 64-bit Windows 7 or newer. After running the MSYS2 installer, an MSYS2 UCRT64 terminal was opened and the following commands were executed in order to install MAKE and a few other basic development utilities.

```
pacman -S base-devel
```

The following command was next run to install mingw-w64 support to generate 64-bit programs.

```
pacman -S mingw-w64-x86_64-toolchain
```

FreeWrap 6.76 Build Process

All necessary source code, other than for freeWrap, should be copied to a location with a common top directory. For example:

/Dev/fwbuild/win64/dyncall-1.4

/Dev/fwbuild/win64/tcl8.6

/Dev/fwbuild/win64/tk8.6

/Dev/fwbuild/win64/tcllib-2.0

/Dev/fwbuild/win64/tklib-0.8

/Dev/fwbuild/win64/Tktable2.11.1

/Dev/fwbuild/win64/twapi-5.0.2

/Dev/fwbuild/win64/zip31c

/Dev/fwbuild/win64/zlib-1.25

\Under Linux freeWrap should be compiled using the normal GNU make and gcc programs.

Under Windows freeWrap is built using MSYS2 and minGW. The MSYS2 and minGW environment allows freeWrap to be built with almost the same procedures under both Windows and Linux. For Windows, perform all package builds from a MINGW64 console.

The freeWrap distribution includes a separate Makefile for each target operating system.

Use the following steps to build under either Linux or Windows.

The following procedural steps have been written with the assumption that the resulting compilation products will be installed under the /usr/local subdirectory.

1. **Compile static library for TCL**

   A static TCL library should be built by using the configure script found in the operating system specific directory of the TCL distribution (i.e., ***unix*** directory for Linux, ***win*** directory for Windows). Change to this directory and issue the following command line.

   ```
   ./configure --prefix=/usr/local --exec_prefix=/usr/local --disable-shared
   ```

   Next, use the normal

   ```
   make
   ```

   command to perform the compilation.

   Unfortunately, under Linux, the normal **make** process compiles TCL's sqlite source code for use with the TCL stubs interface. We need to correct this since we want to statically link sqlite into freeWrap. After running **make** once, perform the following to make this correction:

   a. From the normal TCL build directory (unix or win), edit the Makefile found under pkgs/sqlite3.47.2

   b. Modify the Makefile by removing the following options from the definition of DEFS:

      -DUSE_TCL_STUBS=1
      -DUSE_TCLOO_STUBS=1

   c. Delete the pkgs/sqlite3.47.2/tclsqlite3.o file.

   d. Run **make** again from the operating system specific directory of the TCL distribution (i.e., ***unix*** directory for Linux, ***win*** directory for Windows).

   Use the normal

   ```
   make install
   ```

   command to complete the installation of TCL

2. **Compile static library for TK**

A static TK library should be built by using the configure script found in the operating system specific directory of the TK distribution (i.e., ***unix*** directory for Linux, ***win*** directory for Windows). Change to this directory and issue the following command line.

```
./configure --prefix=/usr/local --exec_prefix=/usr/local --disable-shared
```

Next, use the normal

```
make
make install
```

commands to perform the compilation and installation.

3. **Build zlib compression library**

Obtain the source code package for version 1.2.5 of the **zlib** general purpose data compression library. Do not use any other version of the **zlib** library (e.g., 1.2.5.1, 1.2.8). This is the last version of **zlib** known to work properly with the **ZIP** program you will be compiling in the next step.

Source code for this library is available from the **zlib** archive at http://zlib.net/fossils

Under Linux

Run the following commands from the top of the source tree:

```
./configure –static
make
make install
```

Under Windows

Build the **zlib** libraries using the Makefile.gcc file under the win32 directory of the **zlib** source code tree.

You will need to add the following lines at the top of the win32/Makefile.gcc file after the definition of IMPLIB:

```
SHAREDLIB =
INCLUDE_PATH=/usr/local/include
LIBRARY_PATH=/usr/local/lib
```

```
BINARY_PATH=/usr/local/bin
```

Run the following commands from the top of the source tree. Ignore any error about libzdll.a

```
make -f win32/Makefile.gcc
make -f win32/Makefile.gcc install
```

4. Build the Info-Zip program

Obtain the ZIP-3.1c program source code from Info-Zip's SourceForge site:
https://sourceforge.net/projects/infozip/files/unreleased%20Betas/Zip%20betas/

This source code package is one of the unreleased Betas for the ZIP program. Do not use any other version of the ZIP program (e.g., 2.32 or 3.0). FreeWrap includes a customized version of ZIP's zip.c module that will only work properly with version ZIP 3.1c. This customized file incorporates all the info-zip functionality within freeWrap.

Compile the ZIP program. The freeWrap Makefile uses the object modules produced as a result of building the ZIP program. These modules provide the ZIP program features within freeWrap. Ignore any error messages from the Makefile that are not directly associated with the ZIP program itself.

The freeWrap Makefile also uses the ZIP program to help assemble the freeWrap executable. Therefore, after building the ZIP program place the program in the *linux* or *win* subdirectory (as appropriate) of the freeWrap build tree.

Under Linux

Create and modify the flags file

Run the following two commands from the top of the source tree.

make -f unix/Makefile clean
make -f unix/Makefile flags

Modify the *flags* file at the top of the source tree as follows:

1. Add the following definition statement in front of -DUNIX.
   -DUSE_ZLIB
2. Ensure the LFLAGS2 option is set as follows:

   LFLAGS2 = ”/usr/local/lib/libz.a”

   where /usr/local/lib/libz.a is the path where the static zlib library was installed.
3. Ensure OBJA=””
4. Ensure the -DBZIP2_SUPPORT definition is not present.

Build the zip program

Use the following command from the top of the source tree.

make -f unix/Makefile generic

Ignore any error message associated with building the zipcloak target.

Under Windows

Modify two files

In file win32/makefile.gcc, you need to make the following modifications:

1. Define USEZLIB at the top of the file using a line such as:

   USEZLIB=1

2. Modify the following line:

   CCFLAGS=$(CFLAGS) -DUSE_ZLIB $(LOC)

   to look like:

   CCFLAGS=$(CFLAGS) -DUSE_ZLIB $(LOC) -I/usr/local/include

   in order to point to the directory where the zlib.h file was installed.

3. Modify the following line:

   LIBS=-L. -lz -luser32 -ladvapi32

   to look like:

   LIBS=-L. -L/usr/local/lib -lz -luser32 -ladvapi32

   in order to point to the directory where the libz.a library file was installed.

In the win32/osdep.h file change line 459 to read:

if (!defined(__EMX__) && defined(__MINGW32__) && !defined(__CYGWIN__))

This will force the compilation to use the stdlib.h and mbstring.h headers and avoid an undefined symbol at link time.

In the zip.h file at the top of the source code tree, change the definition of the CR macro to CRTN:

```
#define CR        13
```

to look like:

```
#define CRTN    13
```

In the zipup.c file, change all uses of the CR macro to CRTN.

In the zip.c file, modify the line 5283 to look like:

```
crc_32_tab = (const ulg *)get_crc_table();
```

Build the ZIP program

Use the following command from the top of the source tree.

```
make -f win32/makefile.gcc zip.exe
```

5. Build the Tktable TK extension

Obtain **Tktable 2.11.1** from https://github.com/chpock/tktable.

Use the link to tclconfig found at this location to download a ZIPped copy of tclconfig project.

Extract the contents of the Tktable file.

Next, copy the contents of the tclconfig ZIP archive to the tclconfig directory found under the Tktable directory.

From the top of the Tktable source tree, run the configure script with the --disable-shared option.

./configure --disable-shared

This will create a Tktable Makefile that will produce a static library which the freeWrap Makefile will use.

However, before using the Tktable Makefile you must look for and remove the following text from the definition of DEFS in Makefile.

```
-DUSE_TCL_STUBS=1 -DUSE_TCLOO_STUBS=1 -DUSE_TK_STUBS=1
```

After removing this text you can then do the normal:

make
make install

6. Build the TWAPI TCL extension

If building freeWrap for Windows and you want to include the TWAPI extension then:

(a) Retrieve the TWAPI source code from https://github.com/apnadkarni/twapi/tags.

Extract the TWAPI source code package to the top TWAPI source code directory.

(b) Create a directory named **build** immediately under the top of the TWAPI source code tree.

(c) Open an MSYS2 console and change to the new **build** directory.

(d) On 64-bit Windows enter the following commands into the MSYS2 console.

../configure --enable-threads --disable-shared --enable-64bit
make clean
make
make install

(e) Retrieve the dyncall source code from http://www.dyncall.org.

(f) Open an MSYS2 console and change to the top of the *dyncall* source code directory tree.

(g) Enter the following commands into the MSYS2 console.

make -f Makefile.embedded CC=gcc
make -f Makefile.embedded install DESTDIR=/usr/local

where /usr/local is the path to where the generated *dyncall* library, include, and man page files should be copied.

7. Install the tcllib package

After copying the tcllib package to the common source code directory structure, install the tcllib-2.0 library by running the following two commands from the top of the tcllib package directory. Ignore any errors about critcl.

./configure --exec=/usr/local --exec-prefix=/usr/local
make install

Ignore any error messages pertaining to critcl.

8. Install the tklib package

After copying the tklib package to the common source code directory structure, install the tklib-0.8 library by running the following two commands from the top of the tklib package directory.

./configure --exec=/usr/local --exec-prefix=/usr/local
make install

9. Compile freeWrap

Use the Makefile that comes with the freeWrap source code distribution to build freeWrap.

(a) Background Information

The freeWrap Makefile will use the *main.c* and other source code files provided with the

freeWrap distribution.

The freeWrap Makefile will use the *src/main.c* file provided with the freeWrap source code distribution to build any version of freeWrap. This file has been written to use the ZIP Virtual File System and perform initialization normally done by the standard TCL/TK distribution.alias

The *main.c* source code is also the location in which any statically linked TCL/TK extensions should be initialized. The file currently contains source code for initializing initializing the Tktable and TWAPI extensions.

The src/Windows/TWAPI_pkgIndex.tcl file is a customized copy of the pkgIndex.tcl file included with the TWAPI binary distributions. This is included into freeWrap during the building of freeWrap with the Makefile. TWAPI_pkgIndex.tcl allows for proper loading and initialization of the individual TWAPI modules.

The src\Windows*ico.tcl* file is a copy from the **ico** package of tklib corrected to support 256x256 pixel windows icons. This is included into freeWrap during the building of freeWrap with the Makefile.

The *src/freelib.c* and *src/freewrapCmds.tcl* files implement some TCL commands added by freeWrap.

The *src/zipmain.c* file provides the code to implement the ::freewrap::makeZIP command.

The *src/zvfs.c* file implements the ZIP Virtual File System used by freeWrap.

The file *src/fwcrypt.c* will be automatically generated by the freeWrap Makefile the first time freeWrap is built. This file provides a function that returns the password freeWrap will use for encryption. A new randomly selected password is generated whenever this file is recreated.

FreeWrap encrypts the TCL scripts and data files that you wrap into your application. If you are going to use the cross-wrapping capability of freewrap to wrap applications across different operating systems (using the -w option) you need to ensure that the versions of freeWrap on each operating system use the same *src/fwcrypt.c* file. (This is already the case for the binary distributions of freeWrap.) Otherwise, the wrapped application will issue the following error message when run:

“This application has an unauthorized modification. Exiting immediately.”

The *src/setinfo.tcl* script is used by the freeWrap Makefile in the creation of the freeWrap program.

The *generic/freewrap.tcl* script is the portion of freeWrap that controls the wrapping process.

Files found in the *src/themes* directory provide a number of ttk themes included in freeWrap.

Some necessary files for building under the Windows operating system are included in the *src/Windows* directory. These files are automatically used by the freeWrap Makefile.

(b) Modify the Makefile to reflect the file paths for your computer system.

i. The path defined by INSTALL_BASE within the Makefile must point to the installation directory for the version of TCL/TK from which you are building freeWrap. The Makefile will copy the TCL/TK scripts and libraries it needs from this location.

The LIB_TCL variable in the freeWrap Makefile must point to the static libraries created by the TCL compilation. These files are usually stored someplace like /usr/local/lib/libtcl86.a when "make install" is run as part of the TCL build process.

ii. To build different versions of freeWrap (e.g., freeWrapTCLSH), modify the Makefile to have the proper value of FW_EXT. See the comments in the Makefile for details.

(c) **After modifying the freeWrap Makefile, use the following commands to build freeWrap.**

Under Linux

The Linux Makefile is written to use the gcc compiler.

Make command: `make`

Under Windows

The Windows Makefile is written to support the minGW and MSYS2 environment.

Make command: `make`

10. Encryption password generated during compilation

With the addition of ZIP 2.0 style file encryption, the freeWrap program now contains an embedded password key compiled into freeWrap. The password key is included into freeWrap as a function. The source code for this function is automatically generated by the Makefile the first time freeWrap is built. This source code is placed in a file named *fwcrypt.c*. Embedding the password into freeWrap as a function makes it difficult to detect the password by looking at the executable file.

A plain text copy of the password is stored in the *currentPWD.txt* file found in the freeWrap source code directory.

Each generation of this file results in a completely different password. This allows you to build your own version of freeWrap with its own password that no one else knows. This is important for generating wrapped applications where you want to prevent others from viewing the source code. Due to freeWrap's ability to mount ZIP files as a subdirectory, a person having the same version of freeWrap (with the same password) can easily read the encrypted files within your

application. Therefore, those people interesting in securing the files that make up their application should compile their own copy of freeWrap.

ZVFS: The ZIP Virtual File System TCL Extension

Introduction

The freeWrap program is a TCL/TK script that has been attached to a single-file version of the WISH shell program. The single-file WISH was created with the help of the ZIP Virtual File System (ZVFS) source code provided by D. Richard Hipp. The ZVFS code has been adapted for use with TCL's virtual file system interface.

ZVFS is an extension to TCL that causes TCL to view the contents of a ZIP archive as real, uncompressed, individually-accessible files. Using ZVFS, you "mount" a ZIP archive on a directory of your file system. Thereafter, all of the contents of the ZIP archive appear to be files contained within the directory on which the ZIP file is mounted. The ZVFS extension is written in the C language.

For example, suppose you have a ZIP archive named **example1.zip** and suppose this archive contains three files named **abc.tcl**, **pqrs.gif**, and **xyz.tcl**. You can mount this ZIP archive as follows:

```
zvfs::mount  example1.zip  /zip1
```

After executing the above command, the contents of the ZIP archive appear to be files in the **/zip1** directory. So, for instance, you can now execute commands like these:

```
source /zip1/abc.tcl
image create photo img1 -data /zip1/pqrs.gif
puts "The size of file xyz.tcl is [file size /zip1/xyz.tcl]"
```

The files **/zip1/abc.tcl**, **/zip1/pqrs.gif**, and **/zip1.xyz.tcl** never really exist as separate files on your disk drive. They are always contained within the ZIP archive and are not unpacked. The ZVFS extension intercepts Tcl's attempt to open and read these files and substitutes data from the ZIP archive that is extracted and decompressed on the fly.

Using ZVFS

The ZVFS has been compiled into freeWrap using TCL's Virtual File System (VFS) interface. This extension provides the following new TCL commands:

- **zvfs::mount** ZIP-archive-name mount-point
- **zvfs::unmount** ZIP-archive-name
- **zvfs::exists** filename
- **zvfs::info** filename
- **zvfs::list** ?(**-glob**|**-regexp**)? ?pattern?

As discussed above, the **zvfs::mount** command mounts a new ZIP archive file so that the contents of the archive appear to TCL to be regular files. The first argument is the name of the ZIP archive file. The

second argument is the name of the directory that will appear to hold the contents of the ZIP archive. The ZIP archive may be unmounted using the **zvfs::unmount** command.

The **zvfs::exists** command checks to see if the file named as its first argument exists in a mounted ZIP archive. You can do almost the same thing with the built-in **file exists** command of TCL. The **file exists** command will return true if the named file is contained in a mounted ZIP archive. But **file exists** will also return true if its argument is a real file on the disk, whereas **zvfs::exists** will only return true if the argument is contain in a mounted ZIP archive.

The **zvfs::info** command takes a single argument which is the name of a file contained in a mounted ZIP archive. If the argument is something other than such a file, this routine returns an empty string. IF the argument is a file in a ZIP archive, then this routine returns the following information about that file:

- The name of the ZIP archive that contains the file
- The uncompressed size of the file in bytes
- The compressed size of the file in bytes
- The offset of the beginning of the file in the ZIP archive

The **zvfs::list** command returns a list of all files contained within all mounted ZIP archives. If a single argument is given, that argument is interpreted as a glob pattern and only files that match that glob pattern will match. If the **-regexp** switch appears then the argument is interpreted as a regular expression and only files that match the regular expression are listed.

Limitations

The files in a ZIP archive are read-only. You cannot open a ZVFS mounted file for writing.

The renaming or deletion of files in the ZVFS is not supported.

Overlays

ZVFS allows you to mount a ZIP archive on top of an existing file system. TCL first looks for the file in the ZIP archive and if it is not found there it then looks in the underlying file system. You can also mount multiple ZIP archives on top of one another. The ZIP archives are searched from the most recently mounted back to the least recently mounted.

This overlay behavior is useful for distributing patches or updates to a large program. Suppose you have a large application that contains many TCL scripts which you distribute as a single ZIP archive file. You can start up your application using code like the following:

```
foreach file [lsort -dictionary [glob appcode*.zip]] {
  zvfs::mount $file /appcode
}
```

This loop finds all ZIP archive (in a certain directory) that begin with the prefix **appcode**. It then mounts each ZIP archive on the same /**appcode** directory.

You can use this scheme to ship the TCL scripts of your application in a file named **appcode000.zip**. If

there is later a change or update to your program that effects a small subset of the TCL scripts, you can create a patch file named **appcode001.zip** that contains only the scripts that changed. By placing **appcode001.zip** in the same directory as **appcode000.zip** and restarting the application, all the files in **appcode001.zip** will override files with the same name in **appcode000.zip**. Subsequent updates can be named **appcode002.zip**, **appcode003.zip**, and so forth.

This kind of update scheme makes it very easy to back out a change. Suppose after trying out a particular update, the user decides they do not like it and want to go back to the prior version. All they have to do is remove (or rename) the appropriate **appcode*.zip** file and restart the application and the code automatically reverts to its previous configuration. Updates are completely and trivially reversible.

Using The Executable As The ZIP Archive

The directory information for most executable formats is at the beginning of the file and the directory information for the ZIP archive format is at the end of the file. This means that you can append extra data to an executable and the operating system will not care and you can add information to the start of a ZIP archive and the ZVFS extension will not care. So then, there is nothing to prevent you from appending the ZIP archive to the executable that contains a TCL interpreter and thereby put your entire application into a single standalone file. The freeWrap application does this.

FreeWrap is a compiled C program that creates a TCL interpreter, adds the ZVFS extension, reads the TCL initialization scripts from the attached ZIP archive then executes a TCL script from the same archive. The capabilities of the ZIP archive program Info-ZIP has been compiled into freeWrap. These capabilities are used by freeWrap to perform all file additions and deletions to the archive portion of freeWrapped applications.

When you execute freeWrap or freeWrapped applications, the operating system loads and runs the first part of the file as the executable. Then the freeWrap code calls the ZVFS extension to read the TCL scripts from the end of the file.

ZVFS source code

The source to the ZVFS extension is contained in a single C file named **zvfs.c** and is included with the freeWrap source code distribution.