tdb

Tcl-Debugger

softWorks Richard Schwaninger
personal engineering software

english/debugger.tex
Version 4.1
Document 1.3

20/08/1996

Contents

1. Introduction

Developing software with *Tcl/Tk* is a fast and efficient way to create good-looking programms in a small amount of time. This gives it a real advantage above compiler languages. Nevertheless one would wish better support for some of the default programming tasks.

While an interpreter misses the typical edit/compile/link phase of a compiler the development cyle is nonetheless marked by the fact that a programm is started, an error occurs, the error has to be found, the script edited and the program restarted again only to find the next error…

Now you may ask why a programmer-friendly language like Tcl does not have a good debugger like C. Currently there are only some minimal approaches to this problem (see the chapter on References below).

What are the requirements for a good Tcl debugger to allow comfortable, easy and fast programm development?

- Interactive tracing of the programms control flow (as do other debuggers)
- Examination of the programms state (variables, procedures, stack, etc)
- dynamic reloading of sources – that is you may repair the script while the programm is running
- Connection to an editor – ie. errors in the debugger position the editor to the appropriate point, changes to files in the editor are automatically propagated to the running programm
- easy customisation and extension with user commands
- easy use through a comfortable user interface

The debugger described in the following chapters closes this gap and gives the tcl programmer part of a fast and productive application development environment.

With `tdb` a programm may be executed in a controlled environment. The user can trace the programm step by step, may set breakpoints or inspect data structures and flow of control.

2. Range of Application

By default the debugger *tdb* is an integrated part of the development environment provided by *soft*Works. *tdb* may be also used to debug programms that link in the tcl libraries available from *soft*Works. It goes without saying that *tdb* can debug applications that use pure `Tcl` (and it's extensions).

tdb offers most of the functions of other debuggers (like *gdb*). The tcl language itself has some concepts that are very different from other (compiled) languages, so some debugger commands have no relation to any other debugger.

The standard functions are deliberately modelled after other well known debuggers, so a user can do productive work very fast (if you know *gdb* you will instantly feel at home). Some functions have no similarities as tcl offers functionality that other languages don't provide (eg dynamically changing code while the programm is running).

The interface to emacs is not integrated into the editor as it is in *gdb*. It depends on a co-operation where each part does what it can do best. As an added benefit other editors can be adapted easily (if they do support an appropriate communication interface).

The user interface of *tdb* supports both beginners and advanced programmers. It sports both menus and a command line interface and new commands can be added through an extension language (Tcl of course).

3. Functional Description

3.1 General Data

The main task of a tcl debugger is to show what happens *inside* of a programm. *tdb* does the following kind of things (plus some others in support of these) to help you catch bugs in the act:

- Start your tcl-scripts with arguments. The running programm gets the same environment as it would have when started from the shell.
- Stopping the program at user defined points (by stepwise execution, breakpoints or programm errors).
- Inspection of variables, procedures and the current state of the programm.
- Changing variables or parts of the tcl-script to correct the effects of one bug (and go on learning about another).

3.2 Scripts under control of *tdb*

Tcl-scripts will run without changes under `tdb`. As no compilation is necessary *tdb* may be used as the main development environment. The script is loaded into the debugger. When an error is encountered the bug is fixed in the editor (the appropriate file will be loaded automatically, the cursor positioned at the location of the error) and then the script (or just the procedure) is reloaded. You can directly continue to debug...

The typical development cycle (especially for big programms or graphical applications) will be drastically shortened as the programm does'nt have to be restarted.

3.3 Controlling a Script

tdb gives you full control about a running programm. You may trace step by step through a procedure, may jump over subroutines or return out of loops or procedures with a single command. You may mark critical points with breakpoints to allow returning into the debugger for a thorough inspection.

3.4 Displaying State

At every step the current state of the programm (variables, procedures, stack...) may be inspected. Scalar- and array variables can be changed both locally and at global scope. Small errors may be directly fixed this way. The currently executed code will be displayed both as source and with evaluated arguments. Moreover you may get an overview of loaded source files and defined procedures and commands.

3.5 Miscellaneous

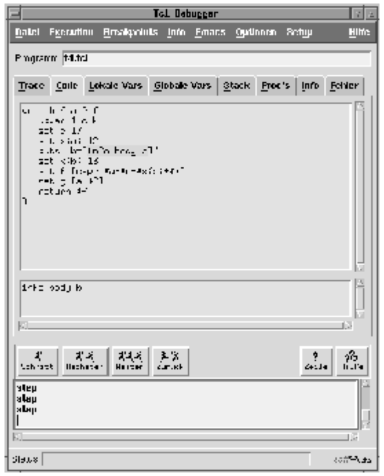
tdb can be used together with an editor (if the editor can be extended and allows communication through sockets). The current binding to *emacs* allows the automatic reloading of sources as soon as a file is saved in the editor. The debugger can display a procedure in the editor (as an example an error triggers the loading of the erronous procedure and the cursor will be positioned at the beginning of the corresponding procedure).

The debugger may be changed and extended with user defined commands. Such commands may shorten repetetive tasks or may define new functionality. Commands may be stored in files and may be (re)loaded from there. They are themselves written in Tcl and have complete access to the underlying object-oriented framework of the debugger.

The debugger allows direct execution of subroutines of the debugged script (eg. to print out a formatted listing of some data structures) and also allows debugging of single procedures.

A two-stage help system supports both beginners and advanced users. Each command (even if user defined) has a short usage message. If this is not enough the context sensitive help function (or the help index) branches to the hypertext help browser which gives exhaustive informations.

The graphical user-interface of *tdb* allows an easy and intuitive use of the debugger without sacrificing a mighty command language (Tcl).



3.6 Drawbacks

There is no easy way to implement debugging in tcl (and using tcl at the same time as an extension language). In fact it is nearly impossible to get at some needed information without modifying the sources.

This gives way to the biggest drawback of *tdb* – you cannot use your own (modified) tcl interpreters (like `tclsh` or `wish` and all C-code extensions must be compiled as dynamically loadable libraries (there should be no reason to use static libraries anyway).

4. Copyright

The programm *tdb* is available in two versions:

- public-domain version
- commercial version

4.1 Copyright for the public-domain version

This version is fully functional and contains essentially the same features as the commercial version. Missing is the graphical context display, the connection to an editor, dynamic reloading of sources and user configurability.

There is no guarany that the public-domain version will be available any longer, that it will be supported or that any bugs will be fixed.

This software may be used, copied, modified and distributed for free as long as the copyright stays intact and you don't make any money from it.

4.2 Copyright for the commercial version

This version is fully functional and contains all the features described in the introduction. With the purchase of this software you are entitled to use the services of *soft*Works (maintenance, support, upgrades). Ask for more details about the commercial version of *tdb* (or just start the license-manager and select `tdb`).

4.3 Generic Copyright Terms

This programm contains parts with their own copyrights like *Tcl* and its extensions *TclX*, *Tk*, *Blt* and *Tix*. These packages are all freely available from the internet.

IN NO EVENT SHALL THE AUTHOR OF THIS SOFTWARE BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES

ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE AUTHOR OF THIS SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHOR OF THIS SOFTWARE SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE AUTHOR OF THIS SOFTWARE HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Additional information may be found in the document *General Business Policy and Terms of Delivery*

5. References

5.1 Other Tcl-Debuggers

As far as I know there are the following other tcl debuggers:

- tcl-debugger from Karl Lehenbauer
  This is the first try for a Tcl-Debugger, no longer supported
- tcl-debugger from Don Libes
  (`code/tcl-debug.tar` in the tcl archive), an offsprint of Karl Lehenbauer debugger, combination with Expect
- Tk-interface to Libes debugger from Tom Tromey
  (`code/tk-debug-0.2.tar.gz` in the tcl archive)
- *TDebug*, a tcl debugger with a graphical interface
  from schmid@fb3-s7.math.tu-berlin.de,
  (`extensions/tdebug-1.0.tar.gz` in the tcl archive) This debugger may be compared with *tdb*, but is generally a lot slower.

6. Changes

6.1 Version 3.2

This is the first version made available to the public. The subversion `3.2.1` is the public-domain version, `3.2.2` is the commercial version.

6.1.1 Known Bugs

tdb uses multiple interpreters to accomplish it's task. The corresponding code in `tcl7.5` contains some bugs when interacting with the I/O-System of tcl which may force a core-dump of *tdb*. It is currently not possible to divert I/O of a debugged process into a tcl-window.

Debugging of event-loop based programms may show (interesting) effects. This problem is not limited to *tdb*, it is common to most debuggers. `tdb` does what you expect most of the time.

6.2 Version 4.1

This version fixes some bugs and introduces the concept of plugins to extend *tdb*'s functionality.

6.2.1 Bug Fixes

Tdb now uses `tcl7.5p1` and `tk4.1p1`. This fixes some bugs related to multiple interpreter I/O and allows input/output of the debugged programm to go into a debugger window.

The commercial version is a lot faster on startup as some basic libraries are now written in C. These libraries will be available to interested parties in source code.

The command line interface is cleaner now.

6.2.2 Known Bugs

Apparently there is a bug somewhere in tdb or it's accompanying libraries that causes a SIGSEGV sometimes after a tcl error happened. Currently there is no known workaround for this inconvenience.

6.2.3 News

Plugin modules may extend the debuggers functionality. Such modules may be written in a mix of C and tcl and can directly access all the internals of the debugger. This modularisation allows the mainstream to be as small as possible while the end user may extend the debugger into his specific direction (eg. to add specific data investigation functions).

Currently two plugins are available: The *script* plugin allows scripting (that is recording and replaying of debugger sessions) and addition of new debugger commands. It is included in the commercial version.

The other plugin adds popup menus to the debugger windows. You can directly set breakpoints, inspect variables or display a procedure by clicking into a debugger window.

A plugin available in the near future will add inspection and display commands for widget hierarchies. Users interested in a pre-alpha version may send us an email to get this plugin.