

Realizing Windows Look & Feel with Tk

Ron Wold

Model Technology
10450 SW Nimbus Avenue, Bldg RB
Portland Oregon, 97223
503-641-1340
rwold@model.com

Brian Griffin

Model Technology
10450 SW Nimbus Avenue, Bldg RB
Portland Oregon, 97223
503-641-1340
bgriffin@model.com

Abstract

ModelSim is a software tool in the Electronic Design Automation (EDA) industry used by digital hardware design engineers. The graphical interface for this tool was written using Tcl/Tk, which, by default, follows the Motif look and feel. An effort was undertaken to convert the interface from Motif to a Microsoft Windows look and feel. This paper will discuss the issues that were addressed, the Tcl/Tk changes that were made, and the technologies that were developed to achieve the Microsoft Windows style user interface.

Keywords

Tcl/Tk, look and feel, Microsoft Windows, GUI widgets, multiple document interface, MDI, printing.

1. Introduction

A software product intended to support multiple hardware platforms faces unique development issues. One such issue is how to address the different graphical user interface (GUI) technologies that are native to each platform. Tcl/Tk is a proven solution to this problem – it defines a single technology base that runs on many different platforms. But, a user interface is comprised of much more than the technology it uses; a user interface is also defined by its look and feel and a set of standards for user interaction.

2. Background

In the early 80's, as hardware design became more complex, designers started turning to more abstract representations to describe digital hardware. This led to the development of Hardware Description Languages (HDLs) similar to software programming languages such as C and ADA. These languages provided executable specifications that could be exercised and tested before any silicon was created. Two languages have emerged as the standard used today for digital design, VHDL [1] and Verilog [2].

These languages are high-level programming languages. Thus, *ModelSim* functions as a software IDE providing run control, source viewing and editing, breakpoints, and the examination and

ModelSim is a registered trademark of Model Technology Incorporated. Model Technology is a trademark of Mentor Graphics Corporation. PostScript is a registered trademark of Adobe Systems Incorporated. UNIX and OpenLook is a registered trademark of AT&T in the USA and other countries. Windows, Microsoft, and MS-DOS are registered trademarks of Microsoft Corporation. Motif is a registered trademark of The Open Software Foundation.

setting of run-time data. In addition to traditional debugging features, the tool has to deal with the concurrent aspects of the design languages. This leads to many views, many windows, and lots of user interaction.

Originally released under the MS-DOS operating system, *ModelSim* was ported to run under Windows 3.1, and then later to several Unix platforms. At one point, three different versions of the GUI were maintained, supporting OpenLook (Sun), Motif (HP) and Microsoft Windows technology. The overhead of three separate versions spurred the search for a single user interface technology. The search ended with Tcl/Tk.

Over time, customer interaction led the *ModelSim* development team to revisit the Motif look and feel. Terms such as “old”, “outdated” and “clunky” were used to describe the *ModelSim* interface. Although the software had the necessary quality and functionality, it became clear that a user's perception of a tool is also based on an abstract notion of its “look and feel”.

3. The Microsoft Windows Paradigm

Many factors combine to make a good user interface. One important factor is consistency. “Software that is full of inconsistencies, even minor ones, forces users to keep thinking about it” [3]. Consistency refers not only to the interactions within a single tool, but also to the interactions across all tools in the customer's suite. Another important factor is familiarity. Tools that are easy to use often are so because the user has experience with software that works similarly.

In an effort to improve the consistency and familiarity of the *ModelSim* interface, a decision to adopt the Microsoft Windows look and feel was made. This is not to say that Microsoft defines a better methodology than OpenLook or Motif, but rather; that the Microsoft look and feel will be more universally familiar to all ilks of *ModelSim* users.

4. The Window

The first step in obtaining the Microsoft look and feel was to define the basic window frame. We determined the window should look as if it were built using the Microsoft Foundation Classes (MFC).

Figure 1 is an example of the original *ModelSim* Main window, Figure 2 is an example of the revised Main window. Both were written with Tcl/Tk, but there were a number of minor changes made to achieve the look of an MFC-based window. Note the difference in the menu bar and toolbar. Besides color and font, the height and spacing are different. The frame in Figure 2 contains an indented marquee, a textured grab corner, and a flat edge. The padding around various objects is less than the Tk defaults.

Toolbar icons were required, and they were used in place of buttons labeled with words. The toolbar icons have balloon help and the same icon transparency and size as a standard Windows icon. Wherever possible, a native Windows icon is used, rather than creating a new icon.



Figure 1. Original Main window (Motif)

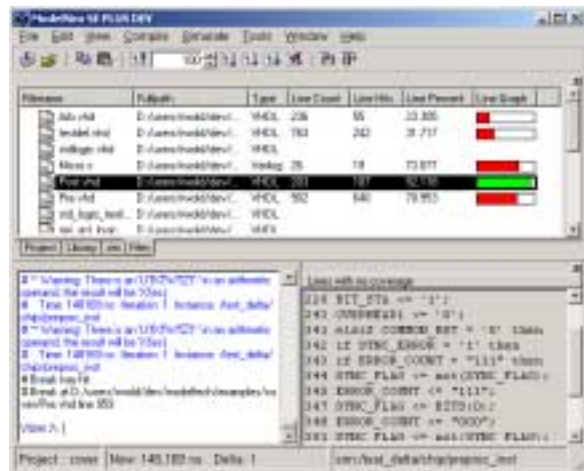


Figure 2. Revised Main window (Windows)

5. Color and Fonts

Matching the color and font of the user's desktop theme was an important step to creating a "native" look and feel. Currently, Tcl/Tk applications do not respond to appearance changes made to the Windows desktop. Following the desktop settings, or rather, not following them, highlights the degree to which an application is integrated with the rest of the environment. This issue was resolved by incorporating a patch from Ian Lance Taylor[8]. Responding to Windows' messages on color and font changes should be intrinsic functionality in Tk.

6. Menus

In addition to matching the look of the Microsoft menu bar, attention was paid to the menu labels and their placement. Windows-compliant tools follow a specific naming convention. Common operations use specific names and are found in specific locations. When defining the menu pick labels, we looked for analogous functionality used in existing Windows-compliant tools. For example, the command *Create* was replaced with *New*, and the command *Reload* was changed to *Refresh*.

Windows users have also become accustomed to the popup menu. Clicking the right mouse button creates a popup menu that is

sensitive to the location of the mouse and to the current selection. Common operations and inquiries on selected items are placed in the popup. For example, users have come to expect a *Properties* command at the bottom of a popup menu.

7. Command Line

As in many other Tcl/Tk tools, ModelSim's original cockpit was a command line. Most commands were issued via the keyboard. Windows applications, however, rarely support a command line; users issue commands from menus, toolbars and dialog boxes. Achieving a Windows look and feel required a de-emphasis of the command line. All commands were made available through menu picks, dialog boxes or toolbar icons.

Windows tools provide users with a starting place. With the de-emphasis of the command line, it became necessary to create a new window pane, a place from which the user would start. We chose to call this pane a workspace.

The workspace is actually a tabbed pane: as the user's design progresses, additional frames are added or removed in the workspace. The key to a workspace is that it provides a persistent view. It is a place where the user can see the current state of things, much like looking at a browser showing the current files in a directory.

8. Dialog Boxes

Dialog boxes play a large role in any user interface. If a Tk programmer creates a dialog, adds some widgets and packs them without regard to final form, the resulting dialog will not look modern or intuitive. There are a number of factors that must be considered in order to achieve an effective look and feel with Tcl/Tk. These factors are both visual and behavioral.

8.1 Layout

Layout has little effect on the functionality of a dialog, but it does affect the user's ability to find and understand the functionality. The layout of a dialog cannot be left up to the grid manager and the default geometry of widgets. Tk's algorithmic layout will guarantee that everything fits, but the results will not be aesthetically pleasing. Figure 3 and Figure 4 are dialogs that provide the same functionality.

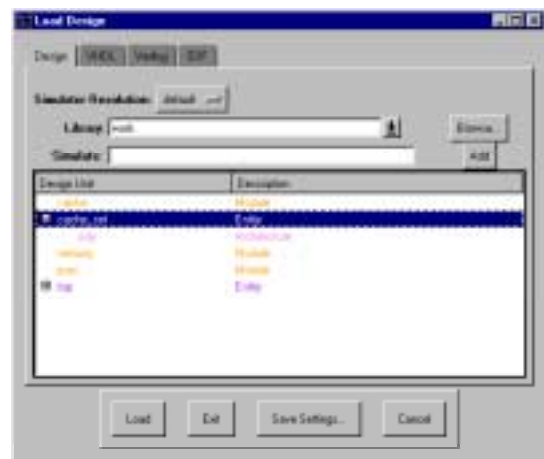


Figure 3. Dialog box layout: Before



Figure 4. Dialog box layout: After

We found the following goals to be particularly helpful with respect to dialog box layout:

Effective Use of Space- Attempt to reduce the amount of empty space in every dialog box. It is easy to create large “holes” in a dialog box using Tk. Reducing the amount of wasted space results in a tighter fit and smaller dialog boxes. In Figure 4, the combo box in the lower right corner is larger than is necessary. Rather than leave a “hole” in the dialog, the box was expanded to fill in the void area.

Alignment- Widgets should line up wherever possible. In Figure 3, the layout of the first three rows was left to the grid manager and nothing seems to line up properly. Alignment should be considered in all four directions, not just left and right edges. In Figure 4, note the last row with the Simulate and Simulator Resolution labeled frames. It was necessary to add padding to the Simulate frame, such that the height matched the Simulator Resolution frame. Further, the Simulator Resolution frame was widened so that the right edge would align with the list box above it.

Grouping- In Figure 3, it is not obvious that the Add button is associated with the Simulate text entry box. In Figure 4, the association was clarified by grouping the objects in a labeled frame.

Padding- In Figure 3, notice that the Browse and Add buttons are touching. Padding should be consistent across all widgets in the dialog, as well as across all dialogs.

Size and Location – Unless specified, the size of a widget is impacted by the other widgets on the dialog and how it is packed. In Figure 3, the Library combo box will likely never hold a name larger than 10 characters, yet the default Tk layout provides space for 50 characters. Widgets should be sized and placed in a location that correlates with the widget's significance.

8.2 Modality

A non-modal dialog does not require a cancel button. All operations are made persistent the moment they are issued and users can perform other operations while the dialog is still up. By default, dialogs defined in Tk are non-modal. Yet non-modal dialog boxes can be confusing for the majority of dialogs that are essentially properties of GUI objects and are rarely found in the Windows look and feel.

Converting a non-modal dialog to modal is more difficult than it appears. Commands issued from a non-modal dialog are persistent.

The user will see the changes in the tool's primary window the moment they are issued. Modal dialog commands are not persistent, so the primary window cannot be used to provide feedback. Modal dialogs require a second view of the data. This view is contained within the dialog and as the user issues commands, the secondary view reflects the changes. A modal dialog must retain the commands that are issued by the user. If the user presses the OK button, the commands are forwarded to the primary window, where they are made persistent.

ModelSim had several non-modal dialog boxes. For some of these dialogs, it did not make sense to change the modality. Dialogs requiring continuous visibility were converted to persistent windowpanes.

8.3 Widget Choice

Many of the widgets available to the Tk programmer look and behave differently from widgets found in Windows. For example, Figure 5 and Figure 6 are two types of non-editable drop down combo boxes. Figure 5 is a Tix widget and Figure 6 is an Incr Widget. Both widgets perform the same functionality, but the Incr Widget is a much better choice for achieving a Windows like look and feel. It displays and behaves almost identically to the standard Windows combo box.

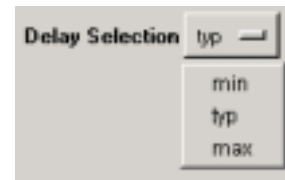


Figure 5. Tix option menu widget

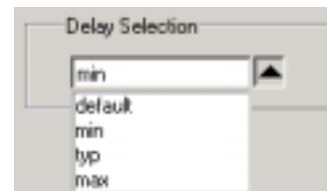


Figure 6. Incr combo box

8.4 Tri-State Widgets

Consider the following scenario: within Windows Explorer you select two files, one that is read-only and one that is not. Next you bring up the "Properties" dialog and there is a check box control based on the read-only property. The Windows look and feel defines that the control be displayed as tri-state. The control is neither true nor false, but is in a third state meaning undefined or ambiguous.

Tri-state controls allow users to change properties on multiple objects at once, even though the objects' properties may differ. Without tri-state controls, a dialog cannot represent the “mixed” state. This Tk deficiency is usually handled by enforcing a single select model. With a single select model, the user can select only one object at a time, and the widget's value is never ambiguous. Allowing only one object to be selected removes the requirement

for tri-state behavior, but it also makes a common manipulation of many objects incredibly tedious.

The Windows look and feel clearly defines how a widget is displayed in each of the following states: disabled, enabled, and tri-state. Figure 7 is an example of how a check box is displayed in the various states.

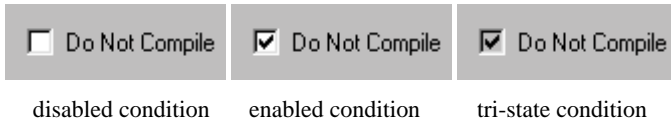


Figure 7. Tri-state checkbox

With Tcl/Tk it was necessary to implement this display behavior.

8.5 Typing vs. Clicking

The Windows interface avoids typing wherever possible; pointing and clicking with the mouse is always the preferred method for providing input. Consider Figure 8, which shows a dialog for entering a pathname to an existing file. The dialog appears normal to an OpenLook or Motif user, but a Windows user would find this dialog odd given that the dialog is requesting a single pathname. Instead, one would expect to see the Open common dialog (Figure 9), which can be navigated entirely with the mouse. When defining dialogs with a Windows look and feel, text entry boxes should be used only where absolutely necessary.

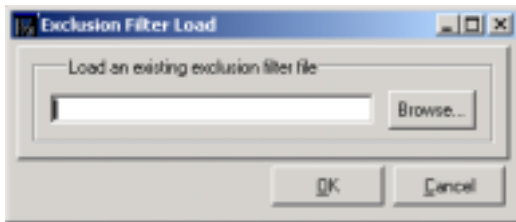


Figure 8. Entry box used for file pathnames.

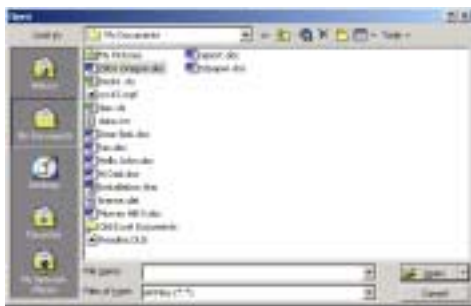


Figure 9. Windows common dialog for file pathnames.

9. Drag & Drop

Dragging and dropping objects (DND) between windows is a common capability of the Windows interface. DND is not native to Tcl/Tk, but there are extensions available that provide support for it. If an object can be selected, then it may be appropriate to

add dragging support. If a pane allows items to be added to it, then it is a likely candidate for supporting dropping.

DND support not only involves interaction between the various windows of a single tool, but also with other tools as well. Initially ModelSim allowed objects to be dragged between its various windows. Later, TkDND [5] was added, which provided support for dragging and dropping with external tools as well.

10. Customization and Dockable Panes

Tools that provide many different views of information confront issues with display real estate. The Windows look and feel employs various techniques for handling screen real-estate issues. One method is called view customization and is used to hide unnecessary data. For example, hiding unnecessary columns in a multi-column list reduces the overall width of a window, freeing space for more important data.

A second method for handling screen real estate is through dockable panes. Dockable panes, including tool and menu bars, allow the user to adjust height, width, and position (Figure 10).

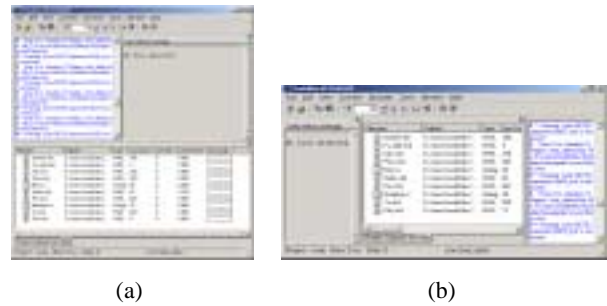


Figure 10. Dockable panes

Tcl/Tk does not define dockable panes so it was necessary to implement this functionality. This is achieved by defining an arbitrary grid object in the main windowpane. The grid allows the width and height to be adjusted, and the “pack -in” facility allows the panes to be repositioned. The Windows dockable pane model allows panes to be dragged *outside* the parent frame, and when dropped they become a stand-alone window. Due to the common ancestry constraint with pack -in, this functionality can be achieved only by constructing a new frame.

11. Printing

For most Windows users, printing is a basic function that is expected. In ModelSim there was one window in particular that users expected to have a printing function. The wave window provides a graphical display of user data, and it's contents could not be represented with a textual report.

To support printing, Windows drawing routines require a device context (DC), a structure that describes the output device. Output is sent to the printer by changing the DC from the screen to a printer. Unfortunately, Tcl/Tk does not provide a method for changing the DC. After reviewing the GDI and Printer extensions [6] for Tk, it was concluded that access to the DC could be gained via custom widgets. Fortunately, the wave window had been constructed from custom widgets due to performance considerations.

Figure 11 is a diagram of the call sequence to illustrate how printing is supported. The non-shaded boxes represent the call chain prior to the addition of printing. The custom widget makes Xlib calls to render itself. On platforms that use an Xserver, the calls are handled directly by the operating system. For non-Xserver platforms, the Xlib calls are translated into equivalent native calls.

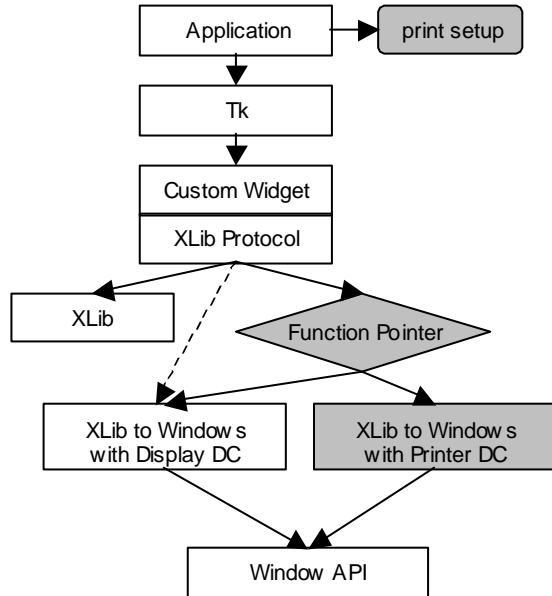


Figure 11. How printing is supported

Supporting printing was not as simple as swapping the DC on the Windows API call. The translation code was defined within Tk, and there was not an obvious method for changing the DC. The solution involved duplicating the Tk translation routines and placing them within the custom widget. These duplicate routines were then modified to use a different DC. Next, the custom widget's Xlib calls were converted from a direct call to a function pointer. The function pointer could be toggled to call either Tk's display routines or the custom widget's printer routines.

The next concern involved pagination, or size of the output and how it is scaled to a printed page. The printer setup action involved bringing up a print dialog where the user selected page size, orientation, margins, etc. Also, the user specified which printer to send the output to, allowing the tool to create the DC for that printer. Pagination is then handled at the application level by combining page size information from the user and printer with total view size information from the widget. The widget is then directed to print each page.

The solution described here is specific to this custom window and involves participation by both the widget and the application. It is less than ideal, but may have implications for a general solution.

12. Multiple Top Level Windows and MDI

At one point, ModelSim supported a multiple document interface (MDI). When the tool was ported to Tcl/Tk, the MDI functionality was not available, so each separate window pane became a stand-alone, top-level window. There were nine different top-level windows, and a user could easily become overwhelmed with sizing

adjustments and visibility issues. Despite adding automatic window placement options such as tiling and pre-defined layout schemes, multiple top-level windows still did not reflect a Windows look and feel.

To solve this issue, several of the top-level windows became several panes within the Main window. In the first phase, we moved several commonly used windows into the Main window. These windows were "re-parented", and have been called *static* window panes. For our next phase, we started working with the Mysund MDI extension [7]. The first stage of the MDI implementation was to implement the widget using IncrTk, as it provides a cleaner interface to the MDI widget.

At the time of this writing the MDI project is not complete. Once the functionality stabilizes, the remaining top-level windows will be incorporated into the Main window, resulting in a tool comprised of a single, top-level frame.

As an aside, Microsoft, who first introduced the MDI concept, appears to be abandoning MDI in favor of Web based interfaces:

"Note MDI is an application-oriented model. Many new and intermediate users find it difficult to learn to use MDI applications. Therefore, many applications are switching to a document-oriented model. Therefore, you may want to consider other models for your user interface. However, you can use MDI for applications which do not easily fit into an existing model until a more suitable model is introduced." [4]

However, because of the complex, multi-dimensional nature of this application, a simple Web or document interface would never do.

13. Wizards

Windows applications frequently include wizards, a series of dialogs that walk users through a complex operation. The dialogs are presented in a predefined order, and the user cannot move to the next dialog until the current dialog has been filled out. Users can always move back to an earlier dialog at any time.

Tcl/Tk does not provide support for wizards; However, all of the functionality necessary to define a wizard exists. To insure consistency, we chose to define a wizard infrastructure or basic wizard. The basic wizard contains the next, back and cancel buttons, a frame for the current page's contents, and the necessary validate functions that allow testing of the current page's data.

14. Multi-Column Hierarchical List

The multi-column hierarchical list can be found in just about every Microsoft Windows tool (e.g. Microsoft Explorer.) Although several Tcl/Tk widgets support much of the needed functionality, no single widget contains all of the features required by ModelSim. It was necessary to develop a new widget that supported navigation through hierarchical expansion of list items, multiple columns, resizing of columns, and sorting by clicking on the column headers. In addition, the list must be capable of displaying icons and other graphical indicators that are commonly found in Windows lists. The list also must support adding new columns or hiding existing columns.

Name	Status	Type	Order	Modified
counter.vhd		VHDL	-	01/31/02 11:18:50 AM
Mico.v	?	Verilog	2	05/30/02 10:04:36 AM
Pic.vhd	✓	VHDL	0	06/03/02 04:01:04 PM
test coverage design		Simulation		
testdel.vhd	✓	VHDL	1	05/30/02 09:57:36 AM
top level design		Folder		
documentation		Folder		
tolpaper.doc		.doc file		04/16/02 01:27:04 PM

Figure 12. Multi-column hierarchical list

Several attempts were made to use existing hierarchical widgets, but each attempt resulted in limited success. It was difficult to support column resizing and sorting using the Tix widget, and the IncrWidget has performance problems with large trees.

With the development of a multi-column hierarchical list that matched the Windows look and feel (Figure 12), we began the process of replacing all of the other Tcl/Tk list widgets within the tool. Consolidating on a single technology has two major advantages. First, all lists throughout the tool have the same look and feel. Second, maintaining ownership of the list widget allows us to modify the widget and obtain the exact behavior we need.

15. Conclusion

The authors confronted many issues trying to achieve a Windows look and feel in a Tcl/Tk-based application. Although the key points vary widely, the issues fall into three categories. The first category refers to issues that required development of a custom solution. In an ideal world, these items would be available within Tcl/Tk for use by all.

- A standard multi-column hierarchical list
- Intrinsic tri-state widgets
- Standard dialog box wizard support
- Dockable/Undockable window panes
- A standard toolbar that supports customization as well as docking
- Printing support, or at least, printing hooks
- MDI support
- Drag and Drop

The second category includes issues that did not require technology development, but that must be considered at all times to achieve a Windows look and feel.

- Dialog box modality

- Usage of *Microsoft Windows like* widgets only
- Dialog box layout, spacing and alignment
- Limiting the number of top-level widgets

Lastly, a number of issues were discovered that do not deal directly with Tcl/Tk, but that are related to the overall goal of achieving a Windows look and feel.

- Usage of Microsoft Verbs and Nouns
- Multiple selection support wherever possible
- Provide point and click operations over typing
- Always provide popup menus and property choices

16. Acknowledgements

Our thanks to the Tcl/Tk community for providing a powerful, flexible, and portable GUI platform.

Many thanks go to the colleagues and wives for the hours spent critiquing this paper.

17. References

- [1] Doulos, A Brief History of VHDL, http://www.doulos.com/fi/desguidevhdl/vb2_history.htm.
- [2] Doulos, A Brief History of Verilog, http://www.doulos.com/fi/desguidevlg/vb2_history.htm
- [3] Johnson, Jeff, GUI bloopers: Don'ts and Do's for Software Developers and Web Designers, Morgan Kaufman Publishers, San Francisco, (2000), pp 42.
- [4] MSDN web site: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/multipledocumentinterface.asp>
- [5] Petasis, George, TkDND, <http://www.iit.demokritos.gr/~petasis/Tcl/tkDND/tkDND.html>.
- [6] Schwartz, Michael I., GDI and Print extensions for Tcl, <http://www.du.edu/~mschwartz/Gdi.txt>, <http://www.du.edu/~mschwartz/Printer.txt>.
- [7] Svensson, Jesper, Mysund MDI <http://www.geocities.com/SiliconValley/Lab/6236/tcltk.html>
- [8] Taylor, Ian Lance, <ian@cygnus.com>, Subject: Adapt to user requested color changes in Tk on Windows, Newsgroups: comp.lang.tcl, Date: 1998/04/16