Editomat:
Adventures in Commercializing a Tcl Application

Clif Flynt
Noumena Corporation,
8888 Black Pine Ln,
Whitmore Lake, MI 48189,

`http://www.noucorp.com`
`clif at noucorp dot com`

October 23, 2016

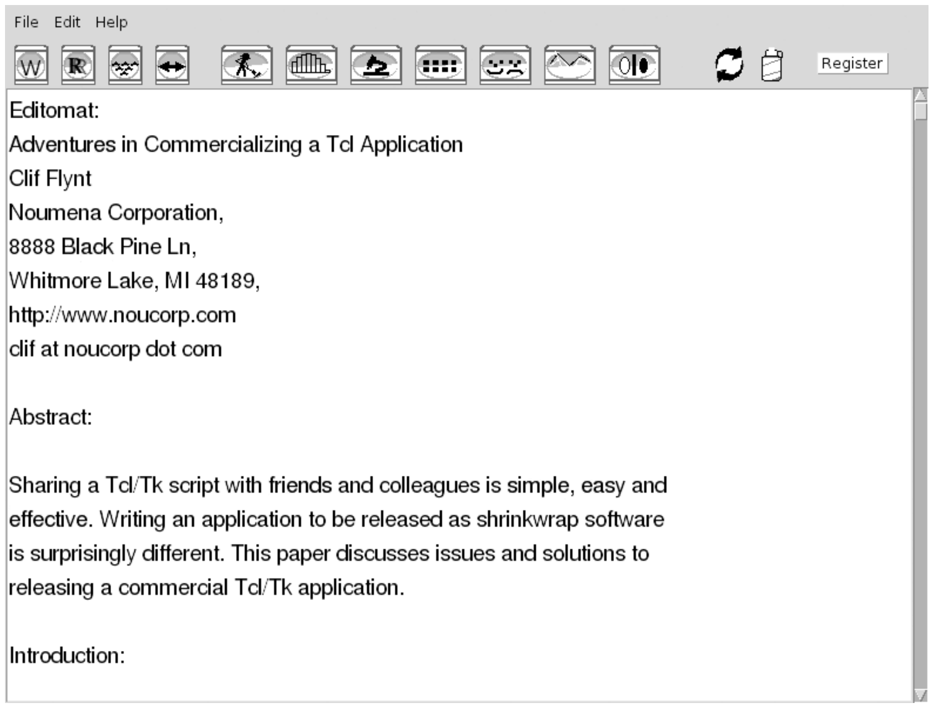
Abstract

Sharing a Tcl/Tk script with friends and colleagues is simple, easy and effective. Writing an application to be released as shrinkwrap software is surprisingly different. This paper discusses issues and solutions to releasing a commercial Tcl/Tk application.

1 Introduction:

I wrote my first FORTRAN program in 1970. Since then, I've written about a million lines of code for myself, my employers and clients. Until the last year, I never wrote something to be sold to the general public. It was a shock to discover how different releasing code to an in-house group or as freeware is from a mass-market release.

Editomat is a tool designed to help writers improve their prose. During the first two years of its life it grew from a trivial application using the text widget's search and tag features to highlight "weak" words to a comprehensive text analysis package. Along with finding meaningless words, it highlights repeated words, awkward sentences, poor constructions, potential grammar errors, as well as graphing sentence length, performing Content/Function sentence analysis, and generating statistical distributions of word and sentence types. It does complex analyses to report emotional content, the mood of a piece and provides comparisons with other works.



Editomat:
Adventures in Commercializing a Tcl Application
Clif Flynt
Noumena Corporation,
8888 Black Pine Ln,
Whitmore Lake, MI 48189,
http://www.noucorp.com
clif at noucorp dot com

Abstract:

Sharing a Tcl/Tk script with friends and colleagues is simple, easy and
effective. Writing an application to be released as shrinkwrap software
is surprisingly different. This paper discusses issues and solutions to
releasing a commercial Tcl/Tk application.

Introduction:

I used it on a daily basis and offered copies to friends who also used it and found it improved their prose. After describing the application at a writer's conference it appeared to be useful enough for a commercial release.

And then the fun started.

2 Requirements For Users:

An engineering or personal use program can be casual. It only needs to work "mostly" and awkward interfaces and workarounds are acceptable. The Motif or Win-95 look and feel is fine, and a hodgepodge of images and text on the buttons is good enough. Tooltip popups are optional and need not be consistent and even help is unnecessary if you're available when someone has a problem.

Commercial software needs to do more than merely work. It must work all the time. It needs an internally consistent, eye-pleasing GUI, a splash screen, acceptance of license, intellectual property protection, managing sales and registration, mating a registered copy to hardware, a website with a shopping cart, customer database and more.

2.1 GUI Considerations and Tk

My initial focus with Editomat was functionality. For personal use, it doesn't matter if something is pretty. it's important that it works. If I had a good-enough icon in my collection, it got used, otherwise, a button with a letter in it was adequate. Tooltips are utterly unnecessary in an application you write for yourself and use frequently. So far as help goes: "Use the Source, Luke".

Obviously, this is not a releasable product. Not even as pre-alpha.

The first project was to rework the GUI. A commercial GUI needs to be at least internally consistent. It's best if it conforms to the behavior the users expect, be it Windows 10, Google Docs or Warcraft.

Tk is frequently maligned for its archaic Motif and Win-95 appearance. The default Tk appearance is usable, but not salable. That doesn't mean that a good GUI can't be created with Tk. There are plenty of existence proofs of good Tk GUIs.

The `option` command and the style support in the ttk widgets provide tools to modify the look-and-feel of an application. Tk uses native widgets for many megawidgets like file selection, which provides users with the expected platform-specific behavior.

Editomat's GUI theme is a pun on laundromat–several "washers" to eclean up your prose. I opted for a *`cute`* GUI to capitalize on the joke. The buttons in Editomat resemble front-loading washing machines with curved corners on top control panel.

Each button uses the -image option to show a graphic. Tk's option command is used to set the borderwidth , padx and pady to 0. This displays only the image with no extra ornamentation, so the top appears to have curved corners, like Mac and Windows 10 buttons.

As further cuteness, when Editomat is busy, the washer doing the work spins and a completion arc marches around the front window. The spinning display might be handled with multiple images, but managing images for fifty states of the completion bar (0 to 100 in 2 percent steps) and the rotation positions becomes unreasonable.

Instead, I show execution by overlaying the body of the button with a canvas. The washer, spinning clothes and completion arc are drawn on the canvas and updated at intervals based on an "after" event. The canvas covers the button being processed courtesy of the "place" command.

To complete a minimal GUI, all buttons must have tooltip popup help and the Help menu should be populated.

The Tcler's Wiki, Tcllib and Tklib are prime sources for tools like tooltip popups. I use a variant of Stewart Allen's tooltip code from the wiki.

My help system is built around the venerable pure Tcl htmllib. For simple help pages, pure Tcl is fast enough. The HTML supported by this widget is sufficient for text, graphics and links. Being pure Tcl there's no compile required for the various ports of Editomat.

The only modification to the standard libhtml.tcl package is a custom link processor to display a new help page.

```
################################################################
# proc HMlink_callback {win href}
#    This proc is called by the html_library code to parse a
#       hypertext reference.
#
# Arguments
#   win         The text window used by the html_library
#               to display the text
#   href        A hypertext reference to use for the next page.
#
# Results
#   This example simply replaces the contents of the display
#   with hardcoded new text.

proc HMlink_callback {win href} {
  global newHTMLtxt

  # Clear the old contents from the window.

  HMreset_win $win

  # Display the new text.

   help::_insertHelp $href
}
```

3 Gentlemen, Start your engines:

An in-house application just starts. Nobody cares if they see the widgets being placed or if there's a blank screen. Commercial software needs to display a EULA acceptance page the first time it's run. On subsequent startups, it should have a splash screen to hide the ugly GUI building, a user tip to occupy the user for a few moments.

Tk's toplevel command provides the basis for splash, tips and EULA. The wm command unmaps the main window until it's fully populated.

3.1 Making a splash

I used the Tk toplevel to create a splash screen. To make it more interesting the splash display is built with a canvas and images. The Tk image's "-gamma" option changes an image's appearance and lets me fade in the image as the mainline code is initializing.

3.2 Providing a tip

Users will be upset if there's no help available, but that doesn't mean they'll read any documentation. The common technique for providing the user with a clue is the startup ″Usage Tip″. Editomat's tips tell the user how to find weak words, how to add new patterns to the list of patterns to highlight and such.

The toplevel, message and button commands provide the tools to create a startup tip. An after event removes the tip window after a few seconds.

3.3 Let me see your License and Registration

An inhouse application doessn't need a license or registration. A commercial product does. The license defines the relationship between you and the buyer. It specifies that they are only buying one copy for their own use, not one to resell a billion times on E-Bay. It protects you from being sued if the program crashes and destroys years of work.

In practice, a license is only worth the money you are willing to pay a lawyer to enforce it, but it gives you a leg to stand on if someone does steal your work or tries to hold you responsible for their problems.

I found a boilerplate license generator on the internet. I used
`https://www.binpress.com/license/generator`
and modified the licence it created to match my needs.

A license must be accepted, either with the popular ″by opening this package you accept the license″ or an explicit ″I accept the license″ button.

Since Editomat is distributed over the Net, not in a shrink-wrapped package, it uses the latter style. When Editomat is started for the first time the user must accept a EULA.

Editomat's Help/About menu option displays the license along with other information about Editomat. Since the help code already exists, I piggybacked the EULA acceptance onto that by creating the window, then removing the ″Dismis″ button and replacing it with ″Accept″ and ″Not Accept″ buttons. Thie acceptLicense procedure uses vwait to create a dialog modal, pausing the execution until the user has either accepted the EULA, or rejected it, in which case Editomat exits.

```
proc acceptLicense {} {
  set ::done 0

  # Display the "About" help screen.
  ::help::about

  # Reset title from default
  wm title .about "Accept License"
  # Remove buttons
  destroy {*}[winfo children .about.bb]

  # Add new buttons
  set ff .about.bb
  button $ff.acc -text "Accept" -command "set ::done 1"
  button $ff.rej -text "Not Accept" -command "set ::done -1"
  grid $ff.acc $ff.rej

  after idle {
  centerWindow .about
  }

  # Wait for user to click something
  vwait ::done

  # Update preferences data to reflect acceptance, or exit
  if {$::done > 0} {
    set ::State(accepted) 1
    savePrefs
    destroy .about
  } else {
    exit
  }
}
```

In the end, I spent almost as many hours cleaning up the appearance and adding help as I spent developing the algorithms.

4 Protecting Intellectual Property:

All ideas are simple once you understand them. The bulk of Editomat is straightforward text processing and algorithms that have been in the literature for decades.

Editomat's implementation and selection of tools is custom. It took me a couple years to find the best techniques and fine-tune the performance and behavior.

The philosophy of Tcl is to make code easy to share, but I don't intend to share the guts of Editomat, so Editomat uses a few tricks to protect its code.

First, Editomat is distributed as a wrapped executable, not as a script. This is partly to make it harder to read, but mostly to confirm that Editomat will run on an unsophisticated user's computer. Relying on a user to install the expected revision of Tcl/Tk is unlikely to lead to market penetration.

The next trick is that the bulk of Editomat's code is stored in an encrypted file within the wrapped executable. The encrypted data is decrypted with a "C" extension that's loaded at runtime from a small Tcl stub.

The last trick to keeping the code safe is to remove the "send" command to prevent tkcon from being attached to a copy of Editomat running on a Linux platform.

5 Regression Testing:

A tool for personal use, or even one used within a department doesn't need rigorous testing. If it fails, you'll be told about a bug quickly and once it's fixed everyone will be happy again.

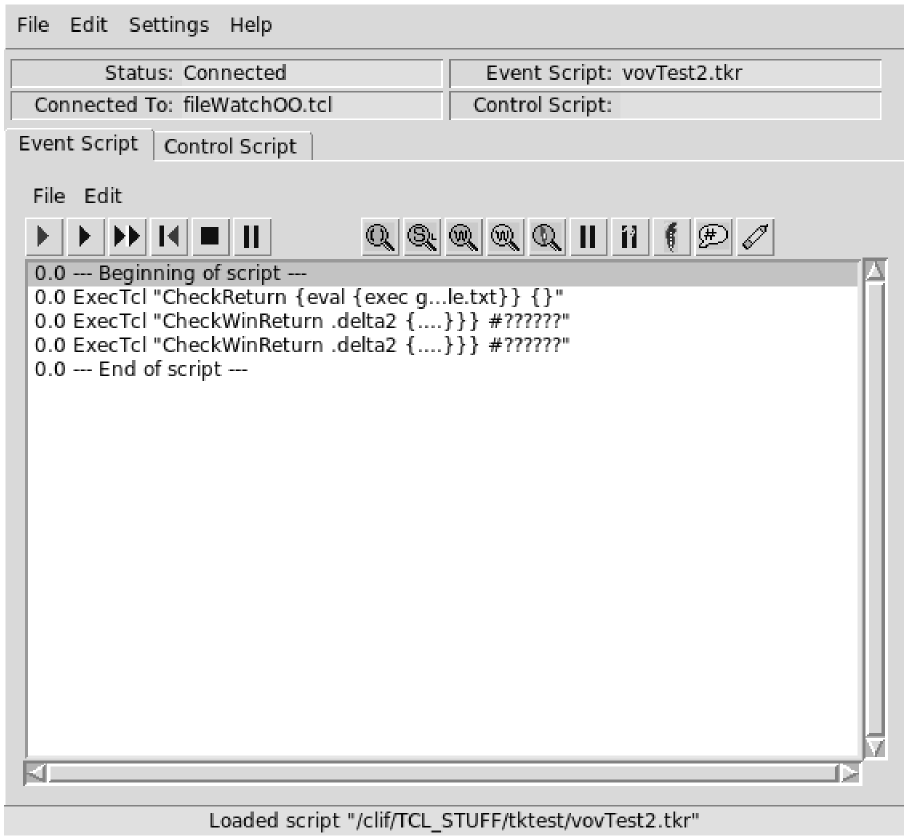
A commercial product needs to behave predictably and consistently. A new feature should not break or even change old behaviors.

5.1 TkTest for regression testing

TkTest is based on the TkReplay framework developed by Charles Crowley of University of New Mexico and described at the 4'th Tcl/Tk Workshop in 1995. TkTest was first presented at the Tcl/Tk Conference in 2004. At that point, TkTest was useful to exercise and test a GUI for consistent behavior on a test-by-test basis.

TkTest was later expanded to support automated regression testing on a suite of tests. This was described at the 2015 Tcl conference.

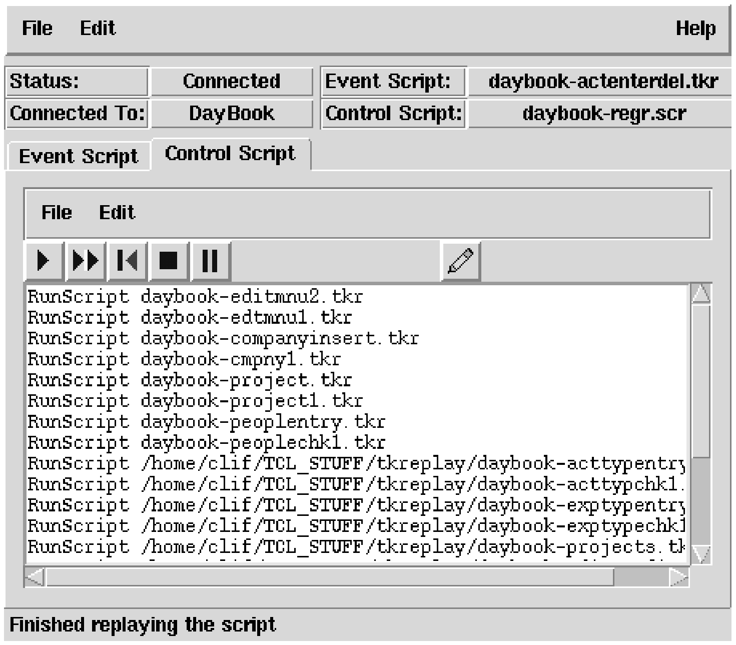
A TkTest script to check the the return value of a Tcl command and the contents of a couple windows resembles the following image.



```
0.0 --- Beginning of script ---
0.0 ExecTcl "CheckReturn {eval {exec g...le.txt}} {}"
0.0 ExecTcl "CheckWinReturn .delta2 {....}}} #??????"
0.0 ExecTcl "CheckWinReturn .delta2 {....}}} #??????"
0.0 --- End of script ---
```

```
Loaded script "/clif/TCL_STUFF/tktest/vovTest2.tkr"
```

These individual tests can be combined to create a larger test suite with the Control Script window.



```
RunScript daybook-editmnu2.tkr
RunScript daybook-edtmnu1.tkr
RunScript daybook-companyinsert.tkr
RunScript daybook-cmpny1.tkr
RunScript daybook-project.tkr
RunScript daybook-project1.tkr
RunScript daybook-peoplentry.tkr
RunScript daybook-peoplechk1.tkr
RunScript /home/clif/TCL_STUFF/tkreplay/daybook-acttypentry
RunScript /home/clif/TCL_STUFF/tkreplay/daybook-acttypchk1.
RunScript /home/clif/TCL_STUFF/tkreplay/daybook-exptypentry
RunScript /home/clif/TCL_STUFF/tkreplay/daybook-exptypechk1
RunScript /home/clif/TCL_STUFF/tkreplay/daybook-projects.tk
```

Finished replaying the script

TkTest's `runRegressionTest.tcl` script uses a file folder based schema to hold multiple sets of tests and expected results. The regression script steps through these folders, runs the tests and records the results.

In theory, this is simple and easy.

It turns out that Tk behaves slightly differently on different platforms. Even different versions of Linux can create a GUI with different fonts. Different size fonts affect what appears in the text widget and change the locations of highlights and tags.

My two approaches to this problem are

1. to add a facility to TkTest to ignore certain sets of data.
2. replace exact expected values with wildcard values.
3. to create customized tests for the different platforms.

Ignoring some data values is simple. Identifying the proper sets to ignore and manually updating the scripts is time-consuming and tedious.

Automating this process is on my list of things to do in TkTest.

TkTest already has a facility to rerun a test and record the new values as correct. This sped up recreating tests for the alternate platforms.

Once I have confirmed that a test is running correctly and the behavior is correct, I run manually those tests on the specific hardware it was developed for.

My current procedure is to run all the regression tests on my Linux development system for each minor release to confirm that no behavior has changed. I spot-test the Mac and Windows platforms to confirm the platform specific code has not been compromised.

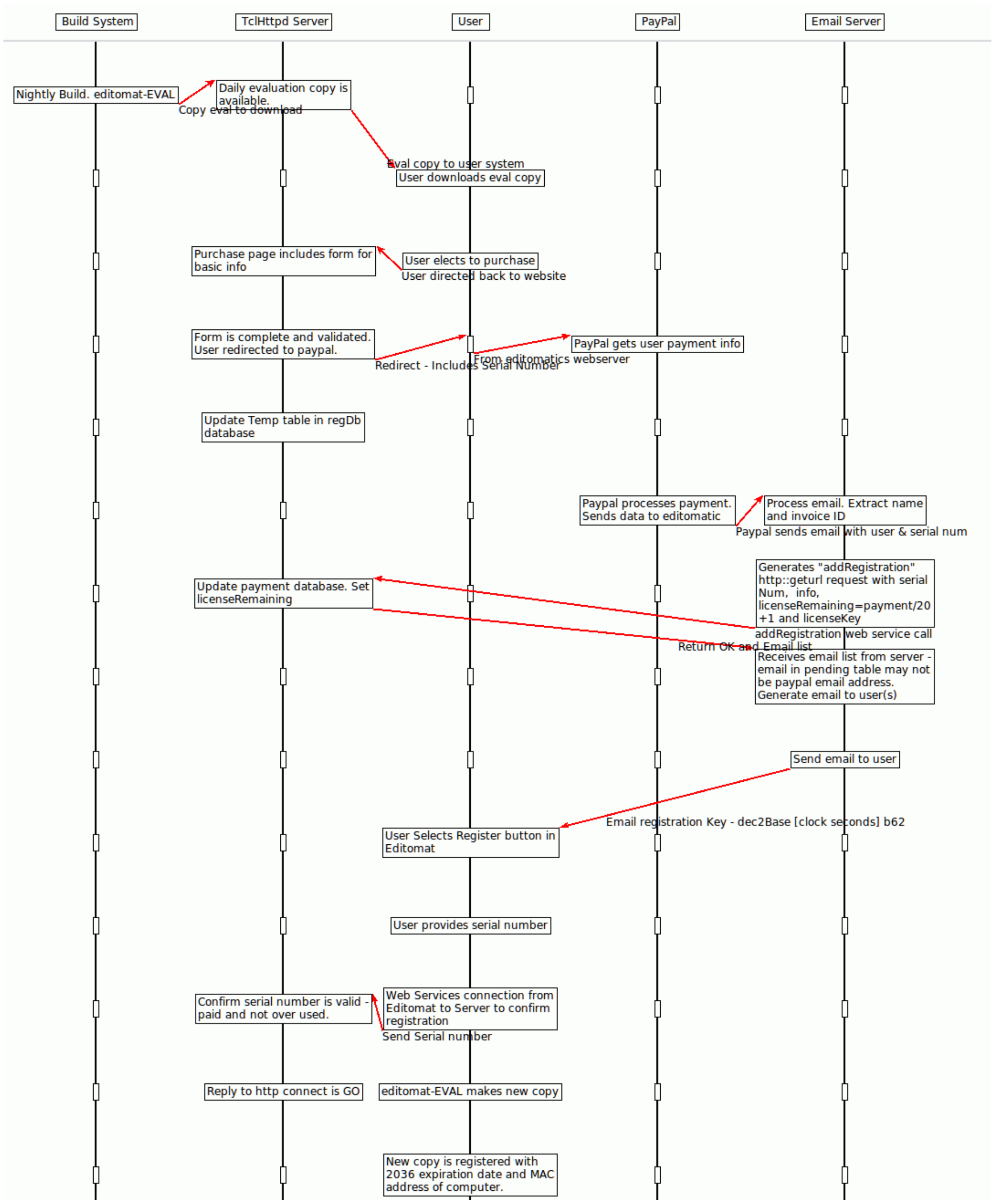
6 Sales and Distribution:

All the clever code in the world doesn't matter if nobody can buy or install the product.

The Editomat sales cycle starts with the user downloading a free evaluation copy from the Editomat web site. This copy is good for 30 days and will process documents up to 20,000 characters long. Once they've tried Editomat and are pleased, they can revisit the site to purchase a license.

Verifying that a user has paid for editomat and is only installing a single copy on a single machine is one of the trickiest parts of the Editomat system. It requires a website form, Paypal, web service calls, and an email processing robot.

The flow resembles this


Build System
TclHttpd Server
User
PayPal
Email Server
Nightly Build. editomat-EVAL
Daily evaluation copy is available.
Copy eval to download
Eval copy to user system
User downloads eval copy
Purchase page includes form for basic info
User elects to purchase
User directed back to website
Form is complete and validated. User redirected to paypal.
Redirect - Includes Serial Number
From editomatics webserver
PayPal gets user payment info
Update Temp table in regDb database
Paypal processes payment. Sends data to editomatic
Paypal sends email with user & serial num
Process email. Extract name and invoice ID
Generates "addRegistration" http::geturl request with serial Num, info, licenseRemaining=payment/20 +1 and licenseKey
addRegistration web service call
Update payment database. Set licenseRemaining
Return OK and Email list
Receives email list from server - email in pending table may not be paypal email address. Generate email to user(s)
Send email to user
Email registration Key - dec2Base [clock seconds] b62
User Selects Register button in Editomat
User provides serial number
Web Services connection from Editomat to Server to confirm registration
Send Serial number
Confirm serial number is valid - paid and not over used.
Reply to http connect is GO
editomat-EVAL makes new copy
New copy is registered with 2036 expiration date and MAC address of computer.

7 The flow of events for purchasing Editomat is:

1. The user downloads the evaluation copy of Editomat: editomat-EVAL.exe
2. The user drags the file to their desktop.
3. The user visits www.editomat.com/Purchase.
4. The user fills out a form to provide email address and other information.
5. The user is redirected to Paypal to complete the transaction.
6. Paypal sends confirmation email to editomat.com
7. An email robot on editomat.com processes the email and generates a webservice action to update the database with payment information
8. The httpd service replies with a registration code.
9. The email robot sends this registration code to the user.
10. The user clicks the "Register" button on editomat-EVAL.exe and types the 6 digit registration code in the entry widget.
11. Editomat sendss a web-service request to www.editomat.com with the registration code.
12. The www.editomat.com httpd daemon checks the database and sends a Go/No-Go reply
13. If the response is Go, Editomat copies editomat-EVAL.exe to editomat.exe and modifies an internal file to include the MAC address of the NIC.

Everything except the user actions is automated.

The editomat database records registration information for each user. This information includes the user's name, preferred email address, some demographic information, a registration code, and the number of registrations allowed for that code. The registration code is used to map payments to registrations. A user my purchase multiple registrations for a single code.

The final step for a user is registering their copy of editomat. When this step is complete, the editomat executable includes a copy of the MAC address of the primary NIC in an encrypted form. Editomat will only run on a machine with that MAC address.

Editomat uses tclhttpd and Tcl On Track as the base website engine, SQLIte for the database (via TDBC) and Paypal as the payment back end. Using Paypal required (finally) adding a payment back-end to Tcl on Track.

The Email robot that processes the Paypal payment reports is an extension of the robot described in Tcl 2010 "Handling Email with Tcl Assist". This robot uses expect to run the venerable mail program. It parses the headers and then invokes a rule engine to find patterns and determine the proper action.

When the phrase ”Notification of payment received” is seen in the subject, the processPayment procedure is invoked. This procedure uses the http::geturl command to send a request to the editomat web server. The reply is then parsed to provide an email address to receive the registration code.

One trick with this system is that the email address from Paypal is the one the user used to register with Paypal. This may not be the same as the email address the user provided when filling out the www.editomat.com/Purchase page.

If the two addresses do not match, the email robot sends a reply to both messages and apologises for spamming the user.

8 Designing a website is not a task for the faint of heart:

The tclhttpd server and Tcl On Track provide the basic engines for the website. The website uses a lot of CSS and a small amount of javascript to define look and feel.

The website works. It has helpful information, interacts with Paypal for purchasing the product and with an SQLite database for storing information.

Despite this, the editomat website is politely described as ”lackluster”.

It’s up for the third major facelift, and another round of user opinions.

9 Taking a sharp turn:

Editomat went into a limited release in July. The immediate feedback was that it needed to work with MS Word to be accepted. The initial users were willing to cut/paste from a word document into Editomat and then modify their prose in MS Word, but apparently those were outliers.

Fortunately, it’s relatively easy to merge a Tcl application with a Window application.

Twapi (`http://twapi.magicsplat.com/`) and Cawt (`www.cawt.com`) provide the basis for Editomat’s MS Word integration. TWAPI provides an interface from Tcl to the Windows API, and Cawt adds a set of high-level tools for Word interaction.

This is not to say it’s simple. Working with Word makes it obvious that the application was initially designed to run in 64K on an 8088. It has existed for decades with dozens of programmers extending it and adding new features. Judging from the slightly different behaviors of features like highlighting text and setting background shading, MS Word has been tweaked frequently without re-working the original design.

The Tcl text widget, for all its faults, is fast and versatile. These are not features of MS Word. Tk supports more colors than the human eye can distinguish. Word supports 16, of which only a few are useful.

9.1 Simple and too simple

The simple solution for word integration is to extract the text from MS Word, insert it into the text widget and export it back to word when the rework is complete.

This solution has several issues.

- MS Word users want MS Word controls, not Emacs.
- MS Word does not report font changes (like italics or bold), so this information is lost.

9.2 Issues with Word

Editomat marks problem words and sentences by highlighting them in the text window. DIfferent issues are highlighted in different colors. MS Word also supports highlighting words, and supports multiple color highlights. It seemed obvious to highlight the same words on the MS-Word screen.

In Tk, Editomat uses the text widget's search support to find problems. This search is fast. The Tcl regular expression engine finds a pattern in at most a few milliseconds.

The MS Word search feature takes over 200 milliseconds. A 200 millisecond time is adequate in an interactive session for finding the next occurance of a word, but it's abysmally slow when you are doing several thousand searches to find repeated words.

The obvious solution is to extract the text from Word and search it with the Tcl regular expression engine.

This isn't as trivial as it sounds either. The Tk text widget indexes the display with line and character location. Word addresses characters by their position in the text file.

However, this N'th character in the text does not map to N'th position on the screen. Word uses non-printing characters to define format changes and other internal information. The word API returns the printing characters, but not the non-printing characters. This makes the position of a word in the extracted text differ from positions on the screen.

When you ask Word to return a single character at a given location, it will return either a character, or an empty string if the character at that location is non-printing. This feature provides a kludgy way to build a mapping table from the character position in a string to the location in the MS Word display.

This technique requires reading characters one at a time to find the non-printing characters. Despite this kludgy trick for reading the text, the overall performance for finding repeated words was improved by about a factor of five over using Word's "Find" option.

9.3 Colors

In the Tk text widget, Editomat highlights problem words with a pastel color that's easy to distinguish from the black text.

MS Word supports 16 highlight colors, including black and white. Only 14 are available for marking text. Of these colors, half are so dark you can't see highlighted text behind them.

MS Word also supports adding background shading to letters. This feature is not implemented exactly the same as highlighting. Highlights can be removed by selecting the entire text and setting the highlight to 0. Shading can be overwritten, but the markers remain behind and new text added near the shaded text is displayed with shading and the previous shading can't be reshown.

The result is that highlighting is easy to use, but can't be mapped back to the expected colors defined in the Tk widget. Background shading can be mapped to Tk colors and is more versatile, but is harder to unset or rework.

10 Conclusion:

Making a functional Tcl/Tk application is fast and easy.

Making one with all the features for a commercial release is not quite so simple, but Tcl and Tk have features to make this easier than other languages.

Intellectual property can be protected by wrapping the application, and byte-code compiling or encrypting the body of the application.

Regression testing a Tk GUI can be done withTkTest. An advantage of Tk and TkTest is that the test harness can introspect into the running application. This feature lets you confirm that GUI actions affected the expected internal structures and allows internal functions to be run and confirm that the GUI was modified appropriately.

The tclhttpd engine, Apache/Rivet, or AOLServer are all good httpd servers to deliver web pages or Web Services and process forms. Making an attractive website is orthogonal to the engine.

Making any application work within the MS-Word framework is non-trivial. The Tcl tools TWAPI and Cawt make working with the Windows API in Tcl as easy as developing in Visual Basic and easier than developing a C# application.

The bulk of the effort in moving an application from "it works" to a commercial release is not a technical problem. It's a function of adding extra features and paying attention to design and desired functionality.