

Extending the Text Widget

Clif Flynt
Noumena Corporation
Whitmore Lake, MI
<http://www.noucorp.com>

Dae-young Kim, Gunes Koru, Urmita Banerjee, Yili Zhang
Health IT Lab at UMBC
Department of Information System
University of Maryland, Baltimore County, MD

Stephen Huntley
stephen.huntley@alum.mit.edu

September 27, 2018

Abstract

We enhanced the Tk Text widget into a full-featured word-processing tool. The enhanced Text widget supports two-keystroke emacs commands, WYSIWYG font and color control and saves text with the font information as a native dump format, XML or RTF document suitable for loading into *Office or MS-Word. We used a TclOO framework, to support expansion as new requirements creep in.

1 Introduction:

The Tk Text widget is a powerful tool that supports both displaying and editing text. The ability to tag sections of text allows text with multiple fonts, colors, underscore and overstrike to be displayed.

The Tk philosophy was to provide developers with basic tools to build complex GUIs, rather than providing all-in-one solutions that might more weighty than a project requires, or otherwise not applicable. Thus, the Text widget can be extended by wrapping it with another layer of Tcl code and intercepting the bindings.

Several extended Text widget packages have been created by wrapping Tcl code around a Text widget. Steve Huntley added bindings for more control options (<http://wiki.tcl.tk/44200>), and Vince Darley created a Tcl/Tk text editing package. The ctext widget in the tklib provides context-sensitive highlighting.

The primary limitations of these are the lack of a simple user interface for controlling the widget contents and the lack of output methods suitable for reloading the data into another Text widget, merging the document into standard text processing tools, or printing.

The unmodified Tk Text widget limits editing commands to arrow-buttons, delete, backspace and a few emacs commands. More advanced features like converting text to uppercase, deleting full words or search are not supported at the user level.

The Tk bind command provides a hook to extend the Text widget's editing features, and even process multi-keystroke operations.

Using the standard Tk button, entry, and combo-box widgets we created a GUI to manipulate tags. The bind command intercepts keystrokes to keep the tags mapped to characters correctly.

It was easy to add this support as a wrapper around a Text widget.

The Text widget's tag command allow it to display text using multiple fonts; varying family, slant, weight, size, foreground color, background color, underscore and overstrike by setting tags within the widget. These tags allow the display of complex formatted documents.

While the tags are easy to manipulate programmatically, the default widget has no way for a user to modify the contents of the display. Word processing applications like MS-Word, and LibreOffice provide the user with a simple GUI to select the font to be used.

Exporting the text contents with sufficient information to regenerate the display with all the fonts and colors is handled with another set of wrappers that use the Text widget's `dump` command to examine the Text widget's contents.

2 Architecture:

There are two orthogonal sets of classes used to implement the extended Text widget.

1. Mixin classes to extend editing commands
2. Super/Child classes that implement export functionality.

The rationale for this is that additional editing commands like extra Emacs or potential vi commands are orthogonal to the text formatting options like color, font, etc. There is no reason to link these together in a single class hierarchy.

One editing command class has been created. The `emacs00-1.0.tm` module adds support for `<Esc>` and `<Ctrl>-X` commands by binding to the `<Key-Release>` event.

The text formatting and export functions are implemented in a set of base and child classes. The base class provides a GUI to define font family, slant, weight, size and colors, along with the bindings to modify tags to implement the display selections.

The child classes add import and export functionality. These classes declare the base class as a super, allowing a child of type `formatTextWinRTF` to be created, but do not have constructors, allowing them to be mixed into the Text widget as needed.

This example shows how an object is created and extended with the emacs bindings and an export format.

```
set obj [formatTextBase .t -fontFamily courier \  
    -fontSize 14 -controlStyle row]  
oo::objdefine $obj mixin emBind  
  
# User selects SaveAs option:  
switch $exportFormat {  
    rtf {  
        oo::objdefine $obj mixin formatTextWinRTF  
    }  
    xml {  
        oo::objdefine $obj mixin formatTextWinXML  
    }  
}
```

Supported export formats include:

Text Widget Native	A serialization of the dump output. Both export and import implemented.
XML using odt tags	A subset of the .odt openDoc format xml without headers. Both export and import implemented.
RTF	RichText Format suitable for loading into most word processing packages. Only export implemented.

3 Implementation:

The components are implemented pure Tcl modules.

Lines	Name
218	emacsOO-1.0.tm Extended emacs commands.
680	formatTextWin-1.0.tm Includes selection GUI and instrumented dump output for export.
513	formatTextWinXML-1.0.tm Reworks the formatTextWin export command's output to XML.
105	formatTextWinRTFSub-1.0.tm Generates RTF output.
1543	ratsub-1.0.tm Modified RatFink RTF library from Joe English.

This modularity supports easy extension with more export formats, and more emacs commands or even vi or MS Word emulation, without including all features in each object.

3.1 Editing Extensions

The standard Text widget creates a binding on the "Text" class to catch <Key> events and invoke the tk::TextInsert command to insert the character into the Text widget.

The Emacs extended commands are defined with an associative array and a set of methods.

```
array set Actions {
    inX,s    {search -forwards}
    inX,r    {search -backwards}
    inEsc,u  {caseConvert toupper}
    inEsc,l  {caseConvert tolower}
    ...
}
method search {direction} {
    ...
}
```

The <Esc> and <Control-X> events are caught by adding two bindings to the specific Text widget, not the Text class. This allows both normal and extended Text widgets to be used together.

The new bindings redirect critical keys to key-specific methods.

```
bind $txt <KeyRelease-Escape> "[self] doEsc"
bind $txt <Control-KeyRelease-x> "[self] doX"
```

The doEsc and doX methods rework the bindings for the widget.

```
method doEsc {} {
    variable EmState
    # Save the current binding for keystrokes
    set EmState(preEsc) [bind $EmState(txt) <Key>]

    # Grab the next keystroke
    bind $EmState(txt) <Key> "[self] gotChar %s %K %T"
```

```

# Set current state
set EmState(EmState) inEsc
}

```

The `gotoChar` method implements the multi-character command and returns the widget to the normal behavior:

```

# Return to normal binding
switch $EmState(EmState) {
  inEsc {
    bind $EmState(txt) <Key> $EmState(preEsc)
  }
  inX {
    bind $EmState(txt) <Key> $EmState(preX)
  }
}

# Implement command associated with key
if {[info exists Actions($EmState(EmState), $K)]} {
  my {*}$Actions($EmState(EmState), $K)
} else {
  # User Input Error
}
set EmState(EmState) ""
return -code break

```

3.2 Formatting and Export

Fonts, sizes, weights, etc and the GUI to control them are implemented in the `formatTextWin-1.0.tcl` module. This class also provides a pre-processed dump output that can be exported and imported to refill a Text widget with the tag and font information intact. This pre-processed dump output is used by other child classes to generate industry standard formats.

3.2.1 GUI

The format control GUI has three configurations:

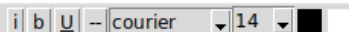
A linear set of buttons that can be mounted above or below the text widget or in a separate toplevel window.

```

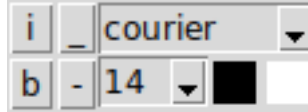
proc gotoStart {} {
  variable State
  $State(txt) see 1.0
  $State(txt) mark set insert 1.0
}

proc gotoEnd {} {

```



Two block formats that can be mounted in the main GUI or in a separate toplevel window.



3.2.2 Exporting Text Widget

Tcl's most basic philosophy is to never hide information from the programmer. Thus, the Text widget's `dump` command returns all the information about how the text and tags, and the `tag configure` command describes the fonts, colors, etc that each tag represents.

The `dump` return is formatted as a set of triplets:

- The type of element (text, tagon, tagoff, mark)
- The index of the element as line.column
- A detail, which might be the text at this location, or the name of a tag.

This example

This is bold
This is normal

returns this dump output (reformatted for easy readability).

```
tagon boldText 1.0
text {This is bold} 1.0
tagoff boldText 1.12
text {
} 1.12
tagon normal 2.0
text {This is normal} 2.0
text {
} 2.14
```

The tags define the format of the text between the tagon and tagoff elements. The Text widget allows tags to overlap. For example, a tag for red text could span many lines with internal tags to set bold or italic font for specific words. The `formatTextWin` classes force all formatting changes into separate tags, so there will be a tag for red normal text, red bold text, etc.

The `export` methods reformat dump output triplets into the commands required for the expected format.

3.2.3 Preprocessed Dump Output

The Text widget `dump` command returns locations of tags and text that describe a document. The `dump` command triplets are sorted into increasing order, but when items appear at the same location, they are not guaranteed to be in the order the new format needs. Also, the dump output does not describe what the tags mean.

The previous example returns this dump output (reformatted for humans to read).

```
tagon boldText 1.0
text {This is bold} 1.0
tagoff boldText 1.12
text {
} 1.12
tagon normal 2.0
text {This is normal} 2.0
text {
} 2.14
```

The XML equivalent to this is close to the `tagon/tagoff` style used by the Text widget.

```
<bold>This is bold</bold>
This is normal
```

The RTF format does not use a close tag, instead it treats the affected text as an argument to a command:

```
\b{This is bold}
```

The closing `tagoff` is not present, but the text within the tag must be marked in the RTF format.

One complicating issue with the dump output is that multiple items can exist at a given location. In the above example, the first tag and the text are both at position 1.0, and the order of the elements is easily parsed to put a command for "bold" into the output, followed by the text. However, if the text is defined before the tag, the parsing is slightly trickier.

```
text 1.0 "this is bold"
tagon 1.0 boldText
tagoff 1.12 boldText
```

This means that an export procedure needs to pre-process the dump return before it can serialize the output. Since this is universal, the base class `export` method generates a list that can be used by other export methods.

The dump output lists all tags, but does not return the configuration information for the tags. Obviously, an export/import pair will need to map the tag names to the configuration, so this is included in the base classes `export` method.

The base class `export` proc returns three lists:

- All tags and their configuration.
This list is used to initialize the tags when importing this data, or by the RTF and XML export methods to map Tcl tag names to the appropriate RTF commands and XML tags.
- All tag ranges as dump-style triplets
The tags are sorted so that there is no interleaving, to simplify the serialization.
- The text as a list of dump-style triplets
The `dump` command orders these the way RTF and XML want them.

The previous example generates an export text that resembles the following (reformatted for easy readability).

```

{boldText {-font {courier 14 bold} -foreground black}
 normalText {-font {courier 14} -foreground black}}

{tagon  boldText 1.0
 tagoff boldText 2.0
 tagon  normalText 2.0
 tagoff normalText 2.12}

{text {This is bold} 1.0 text {
} 1.12}

```

The tags may be reordered because dump output can invert tagon and tagoff elements when they appear at the same index:

```

{tagon  boldText 1.0
 tagon  normalText 2.0
 tagoff boldText 2.0
 tagoff normalText 2.12}

```

The tagon normalText and tagoff boldText triplets are in the opposite order as required for languages like HTML and XML. For example, the following XML is invalid.

```

<bold>
this is bold
<normal>
</bold>
this is normal
</normal>

```

The export code remembers tags that have been turned on and off and prevents mis-matched sets of tags.

This near-native format is sufficient to reproduce the contents of one text widget in another. In order to test the behavior, both `export` and `import` methods are implemented.

3.2.4 XML

The XML output conforms to the standard tags described in <http://officeopenxml.com/WPtextFormatting.php>.

The export facility uses the base `formatTextWin` export method to pre-process the contents of the Text widget. It uses the list of tags and tag formats to create an XML tag for each Text widget tag.

Finally, the method steps through the tags and text, replacing the tagon and tagoff sequences with the appropriate XML tags.

The XML output is simple enough that an `import` method was also written for this format.

The above example produces this XML output (again reformatted for humans):

```

{<w:sz w:val="14">
 <w:font w:name="courier">
 <w:b><w:color w:val="black">This is bold
 </w:color></w:b></w:font></w:sz>
 <w:sz w:val="14"><w:font w:name="courier"><w:color w:val="black">}
 {This is normal</w:color>}

```

This format is easy to parse and to facilitate testing, both `export` and `import` methods are implemented.

3.2.5 RTF

We used Joe English's RatFink package to generate the RTF output.

RTF stands for Rich Text Format. It's a Microsoft standard designed to make it possible to transfer documents between word processors with different native formats.

That Joe called his package RatFink speaks to his restraint. The RTF (sub) standard is ugly, inconsistent and obviously hacked together from a simple idea that met the real world like a drunk driver meets a bridge abutment.

However, for all its faults, RTF is one of the most common ways to transfer text between applications. It's understood by Scrivener, Editomat, MS Word, WordPerfect, LibreOffice and OpenOffice and many other applications.

Fortunately, applications that read RTF are relatively forgiving, and generating RTF requires only a subset of commands and possible ways to define the text.

As is common when a package is used for a project unlike the original, we modified the RatFink package slightly:

- Renamed from `ratfink` to `ratsub`.
- Renamed from Tcl extension package to a `.tm` module.
- Removed all the WinHelp support
- Added `proc init` to initialize the `rtf_state` variable
- Added procs `startColor` and `startColorByNum` to save non-black text.
- Added `proc rtf::exportTextWinToFile` that accepts two arguments,
 - name of text window to export information from
 - name of file to receive the output

The `rtf::exportTextWinToFile` proc breaks the convention of pre-processing the Text widget's contents in the base class and generating output from that.

The reason for this change is that RTF is strongly line/paragraph oriented, and the dump output is less so. Going directly to the Text widget supports this construct, which is (perhaps only slightly) easier to work with.

```
lassign [split [$textWin index end] .] lnCount ch
for {set ln 1} {$ln <= $lnCount} {incr ln} {
    foreach {typ nm index} [$textWin dump -all $ln.0 $ln.end] {
        ...
    }
}
```

The end result is that the RTF output (less headers, color tables, font tables, etc.) resembles this:

```
\pard\plain \s1\f0\fs20\li720 {\cs6\f2\fs28\b
\cf13This is bold}\par
\pard\plain \s1\f0\fs20\li720 {\cs6\f2\fs28\b
\cf13This is normal}\par
```

4 To Do

All software needs more testing. This package has seen light use in Noumena Corporation's Editomat and UMBC's DQT application.

The selection GUI supports the three basic font faces, serif, sans-serif and fixed. It should query the system to allow selection of all available font faces. That modification will also require updating the export functions to load the non-standard fonts in the export file's font definition tables.

The XML output uses a subset of OpenDoc XML tags. It does not provide the header, color tables, font definitions, etc that will allow it to export a .docx or .odt file.

The export format options should include PDF. Adding PDF support should be relatively easy using either the PDF library developed by Clif Flynt and expanded by Krzysztof Blicharski, pdf4tcl or one of the other PDF generating packages.

The primary consideration for PDF formatted documents will be determining when to break the output into pages. The Text widget treats text as a single long document, instead of discrete pages.

Joe English's RatFink extension was written long ago. It should be upgraded to modern conventions, and perhaps turned into a TclOO mixin class.

5 Availability

These packages are available from Noumena Corporation's website: www.noucorp.com, and will be submitted to tklib soon.