Tk4.0 Overview and Porting Guide

John Ousterhout

Tk version 4.0 is a major new release with many improvements, new features, and bug fixes. This document provides an introduction to the new features and describes the most common problems you are likely to encounter when porting scripts from Tk 3.6, the previous release. This is *not* an introduction to Tk: I assume that you are already familiar with Tk 3.6 as described in the book *Tcl and the Tk Toolkit*.

The good news about Tk 4.0 is that it has many improvements over Tk 3.6. Here are a few of the most important new features:

- Tk 4.0 includes a general-purpose mechanism for manipulating color images (Tk 3.6 supports only monochrome images).
- The text widget in Tk 4.0 includes many new features such as tab stops, embedded windows, horizontal scrolling, and many new formatting options.
- The binding mechanism in Tk 4.0 is much more powerful in Tk 3.6.
- Motif compliance is much better. For example, there is now support for keyboard traversal and focus highlights.
- Many widgets have been improved. For example, buttons and labels can display multi-line justified text, and scales can handle real values.

The bad news about Tk 4.0 is that it contains several incompatibilities with Tk 3.6. Ever since the first release of Tk I have assumed that there would eventually be a major new release of Tk with substantial incompatibilities. I knew that I wouldn't be able to get all of the features of Tk right the first time; rather than live forever with all of my early mistakes, I wanted to have a chance to correct them. Tk 4.0 is that correction. I apologize for the incompatibilities, but I hope they improve Tk enough to justify the difficulties you encounter during porting. Tk 4.0 is a one-time correction: we will try very hard to avoid substantial incompatibilities (especially in Tk's Tcl-level interfaces) in future releases.

Sections 1-11 cover the major areas of change in Tk 4.0: bindings, focus, text widgets, Motif compliance, other widget changes, images, color management, event handling, support for multiple displays, the `send` command, and the selection. Section 12 summarizes several smaller changes. Section 13 lists all of the incompatibilities that affect Tcl scripts, along with suggestions for how to deal with them. The explanations here are not intended to be comprehensive, but rather to introduce you to the issues; for complete information on new or modified commands, refer to the reference documentation that comes with the distribution.

1 Bindings

The changes for Tk 4.0 that are most likely to affect existing Tcl scripts are those related to bindings. The new binding mechanism in Tk 4.0 is much more powerful than that of Tk 3.6, particularly in the way it allows behaviors to be combined, but several incompatible changes were required to implement the new features. These changes are likely to break most Tk 3.6 scripts. Fortunately, it is relatively easy to upgrade your bindings to work under Tk 4.0.

The basic mechanism for bindings is the same as in Tk 3.6. A binding associates a Tcl script with a particular event (or sequence of events) occurring in one or more windows; the script will be invoked automatically whenever the event sequence occurs in any of the specified windows. The Tk 4.0 binding mechanism has three major feature changes. First, there is a more general mechanism for specifying the relationship between windows and bindings, called *binding tags*. Second, the conflict resolution mechanism (which is invoked when more than one binding matches an event) has been changed to allow more than one binding script to execute for a single event. Third, the `Any` modifier is now implicit in all binding patterns. These changes are discussed separately in the subsections that follow.

Overall, the main effect of Tk 4.0's binding changes is that it allows more bindings to trigger than Tk 3.6 does. Feedback from the Tcl/Tk community about the Tk 3.6 binding mechanism indicated that it was too conservative about triggering bindings. This caused the system to lose behaviors relatively easily and made the binding structure fragile. It appears to be easier to deal with too many binding invocations than too few, so Tk 4.0 tries to err in this direction.

1.1 Binding tags

In Tk 3.6 you specify the window(s) for a binding in one of three ways:

- You give the name of a window, such as `.a.b.c`, in which case the binding applies only to that window.

- You give the name of a class, such as `Button`, in which case the binding applies to all the windows of that class.
- You specify `all`, in which case the binding applies to all windows.

In Tk4.0 you specify the window(s) using a more general mechanism called a *binding tag*. A binding tag may be an arbitrary string, but if it starts with a "." then it must be the name of a window. If you specify a class name or `all` as a binding tag, it will usually have the same effect as in Tk 3.6, but you may also specify other strings that were not permitted in Tk 3.6.

Each window in Tk 4.0 has a list of binding tags. When an event occurs in a window, Tk fetches the window's binding tags and matches the event against all of the bindings for any of the tags. By default, the binding tags for a window consist of the window name, its class name, the name of its nearest toplevel ancestor, and `all`. For example, a button window named `.b` will have the tags

```
.b Button . all
```

by default and all of the following bindings will apply to the window:

```
bind .b <Enter> {identify "press here to exit"}
bind Button <Button-Release-1> {%W invoke}
bind all <Help> {help %W}
```

So far, this mechanism produces the same behavior as in Tk 3.6 except that bindings created for a toplevel also apply to its descendants (see Section 1.5 for more on this issue).

You can use the `bindtags` command to change the binding tags for a window or their order. For example, the command

```
bindtags .b {.b MyButton all}
```

will change the binding tags for `.b` to the three values in the list. This provides a simple way to make radical changes the behavior of a window. After the above command is invoked none of the `Button` class bindings will apply to `.b`. Instead, bindings for `MyButton` will apply; this might give the button a totally different set of behaviors than a normal button. In addition, the `bindtags` command removes the "." tag, so bindings on "." will not apply to `.b`.

You can also place additional tags on a window with the `bindtags` command to combine a number of behaviors. For example,

```
bindtags .b {.b MyButton Button . all}
```

gives `.b` the behaviors of `MyButton` bindings as well as those specified by `Button` bindings.

Overall, binding tags are similar to the tag mechanisms already used internally by canvas and text widgets in Tk 3.6, except that binding tags apply to windows instead of graphical objects or textual characters.

1.2 Conflict resolution

It is possible for several bindings to match a particular event. In Tk 3.6 at most one event is actually allowed to trigger: a set of conflict resolution rules determines the winner. In general, a more specific binding takes precedence over a less specific binding. For example, any binding for a specific widget takes precedence over any class or `all` binding, and a binding on `<Control-a>` takes precedence over a binding on `<KeyPress>.`

The mechanism for conflict resolution is similar in Tk 4.0 except that one binding can trigger for *each* binding tag on the window where the event occurs. The bindings trigger in the order of the tags. Thus if button `.b` has the default binding tags, one binding for `.b` can trigger, followed by one for `Button`, followed by one for "`.`", followed by one for `all`. If there are no matching bindings for a given tag then none will trigger, and if there are several matching bindings for a given tag then a single one is chosen using the same rules as in Tk 3.6.

The philosophy behind binding tags in Tk 4.0 is that each binding tag corresponds to an independent behavior, so bindings with different tags should usually be additive. Suppose you defined the following binding:

```
bind .b <Enter> {puts "press here to exit"}
```

This binding will add to the behavior defined by the Button class binding for `<Enter>`. In Tk 3.6, the widget-specific binding will replace the class binding, which will break the behavior of the button so that it no longer has normal button behavior.

Sometimes there need to be interactions between binding tags. For example, you might wish to keep most of the default button behavior for `.b` but replace the default behavior for `<ButtonRelease>` with some other behavior. To allow bindings to be overridden, Tk 4.0 allows the `break` command to be invoked from inside a binding. This causes all remaining binding tags for that binding to be skipped. Consider the following binding:

```
bind .b <ButtonRelease-1> {myRelease .b; break}
```

This will cause the `myRelease` procedure to be invoked, then the `break` command will cause the class binding for the event to be skipped (assuming that the widget name appears before its class in the binding tags for `.b`), along with any bindings for other tags.

Note: *You cannot invoke `break` from within the `myRelease` procedure in the above example: this will generate a Tcl error. However, you can invoke the command "`return -code break`" in the procedure to achieve the same effect as the `break` in the binding script.*

1.3 Implicit Any

In Tk 3.6 extraneous modifiers prevent a binding from matching an event. For example, if a binding is defined for `<Button-1>` and the mouse button is pressed with the `Num-Lock` key down, then the binding will not match. If you want a binding to trigger even when extraneous modifiers are present, you must specify the `Any` modifier, as in `<Any-Button-1>`.

In Tk 4.0, all bindings have the `Any` modifier present implicitly. The `Any` modifier is still allowed for compatibility, but it has no meaning. Thus a binding for `<Button-1>` will match a button press event even if `NumLock`, `Shift`, `Control`, or any combination of them. If you wish for a binding not to trigger when a modifier is present, you can just define an empty binding for that modifier combination. For example,

```
bind .b <Control-ButtonPress-1> {# this script is a no-op}
```

creates a binding that will trigger on mouse button presses when the `Control` key is down. If there is also a `<ButtonPress-1>` binding for `.b`, it will no longer be invoked if the `Control` key is down, due to the conflict resolution rules. The script for the above binding is just a Tcl comment, so it has no effect when it is invoked. Alternatively, you could use `%s` in the binding script to extract the modifier state, then test to see that only desired modifiers are present.

1.4 Porting problems: widget bindings vs. class bindings

You are likely to encounter two problems with bindings when you port Tk 3.6 scripts to Tk 4.0: widget bindings vs. class bindings, and events on top-level windows. This section discusses the first problem and the following section discusses the second problem.

In Tk 3.6, if a widget-specific binding matches an event then no class binding will trigger for the event; in Tk 4.0 both bindings will trigger. Because of this change, you will need to modify most of your widget-specific bindings in one of two ways. If a widget-specific binding in Tk 3.6 was intended to supplement the class binding, this could only be done by duplicating the code of the class binding in the widget binding script. This duplicated code is no longer necessary in Tk 4.0 and will probably interfere with the new class bindings in Tk 4.0; you should remove the duplicated class code, leaving only the widget-specific code in the binding script. If a widget-specific binding in Tk 3.6 was intended to override the class binding, this will no longer occur by default in Tk 4.0; you should add a `break` command at the end of the binding script to prevent the class binding from triggering. If a widget binding in Tk 3.6 didn't conflict with a class binding, then you will not need to modify it for Tk 4.0. For example, a widget binding for `<Help>` in a text widget would not need to be modified, since it doesn't conflict with a class binding.

1.5 Porting problems: events on top-levels

The second binding problem you are likely to encounter in porting Tk 3.6 scripts to Tk 4.0 is that in Tk 4.0 a binding on a toplevel will match events on any of the internal windows within that top-level. For example, suppose you have a binding created as follows:

```
toplevel .t
button .t.b1 ...
button .t.b2 ...
bind .t <Enter> action
```

This binding will trigger not only when the mouse enters `.t`, but also when it enters either `.t.b1` or `.t.b2`. This is because the binding tags for a window include its nearest ancestor toplevel by default. The toplevel is present in the binding tags to make it easy to set up accelerator keys that apply in all the windows of a panel. For example,

```
bind .t <Control-a> {controlAProc %W}
```

will cause `controlAProc` to be invoked whenever `Control-a` is typed in any of the windows in `.t`. The procedure will receive the name of the focus window as its argument.

Unfortunately, if you have created bindings on toplevel windows in your Tk 3.6 scripts, they probably expect to trigger only for events in the toplevel, so the bindings will misbehave under Tk 4.0. Fortunately you can reproduce the behavior of Tk 3.6 by using the `%W` substitution in the binding script. For example, to ensure that `action` is invoked only for `Enter` events in a toplevel window itself, create the following binding in place of the one above:

```
bind .t <Enter> {
    if {"%W" == ".t"} {
        action
    }
}
```

When an `Enter` event occurs in a descendant of `.t` such as `.t.x`, a binding for `Enter` in `.t.x` will trigger first, if there is one. Then the above binding will trigger. Since `%W` will be substituted with `.t.x`, the `if` condition will not be satisfied and the binding will not do anything.

An alternative solution is to remove the toplevel window from the binding tags of all its internal windows. However, this means that you won't be able to take advantage of the tag to create key bindings that apply everywhere within the toplevel.

1.6 Internal bindings in canvases and texts

The same changes in conflict resolution described in Section 1.2 also apply to bindings created internally for the items of a canvas or the tags of a text widget. If a canvas item or character of text has multiple tags, then one binding can trigger for each tag on each event. The bindings trigger in the priority order of the tags. Similar porting problems are likely to occur as described in Section 1.4; if a binding for one tag needs to override that of another tag, you'll need to add a `break` command under Tk 4.0; if a binding for one tag duplicated the code from another tag's binding, so that they will compose in Tk 3.6, you'll have to remove the duplicated code in Tk 4.0.

2 Focus management

The input focus is another area where Tk 4.0 contains major changes. Fortunately, the focus changes should not require as many modifications to your Tk 3.6 scripts as the binding changes.

2.1 One focus window per toplevel

Tk 3.6 only keeps track of a single focus window for each application, and this results in two problems. First, it doesn't allow an application to use multiple displays since this could result in multiple simultaneous focus windows, one on each display. Second, the Tk 3.6 model doesn't work very well for applications that have multiple toplevels: when the mouse moves from one toplevel to another, the focus window should switch to whatever window had the focus the last time the mouse was in the new toplevel, but Tk 3.6 does not remember this information.

Tk 4.0 corrects both of these problems. It remembers one focus window for each toplevel, which can be queried with the `focus -lastfor` command. When the window manager gives the focus to a toplevel window (because the mouse entered the window or because you clicked on the window, depending on the focus model being used by the window manager), Tk passes the focus on to the remembered window. Several windows in an application can have the focus at the same time, one on each display the application is using. When asking for the current focus window in the `focus` command, you can use the `-displayof` switch to specify a particular display.

When you set the focus to a window with the `focus` command, Tk remembers that window as the most recent focus window for its toplevel. In addition, if the application currently has the focus for the window's display, Tk moves the focus to the specified window; this can be used, for example to move the focus to a dialog when the dialog is posted, or to perform keyboard traversal among the toplevels of an application. If the application doesn't currently have the focus for the display, then Tk will not normally take the focus from its current owner. However, you can specify the `-force` argument to `focus` to insist that Tk grab the focus for this application (in general this is probably not a good idea, since it may clash with the window manager's focus policy).

2.2 Keyboard traversal

Tk 4.0 has a much more complete implementation of keyboard traversal than Tk 3.6. In Tk 3.6 there is built-in support only for keyboard traversal of menus. In Tk 4.0 keyboard traversal is implemented for all widgets. You can type `Tab` to move the focus among the windows within a toplevel and `Shift+Tab` to move in the reverse direction. The order of traversal is defined by the stacking order of widgets, with the lowest widget first in the traversal order. All Tk widgets now provide a `-takefocus` option, which determines whether the window should accept the focus during traversal or be skipped. This option has several features; see the `options.n` manual entry for details.

All of the Tk widgets provide a traversal highlight ring as required by Motif. The highlight ring turns dark when the widget has the input focus. Its size and colors are controlled by the `-highlightthickness`, `-highlightbackground`, and `-highlightcolor` options. You may notice that widgets appear to have extra space around them in Tk 4.0; this is due to the traversal highlight ring, which is normally the same color as the background for widgets.

2.3 Support for focus-follows-mouse

Both Tk 3.6 and Tk 4.0 use an *explicit focus model* within a toplevel. This means that moving the mouse among the windows of a toplevel does not normally move the focus; you have to click or perform some other action (such as pressing `Tab`) to move the focus. Tk 3.6 has no support for an *implicit focus model* where the window under the mouse always has the focus. In Tk 4.0 you can invoke the library procedure `tk_focusFollowsMouse` to switch to an implicit focus model; in this mode whenever the mouse enters a new window the focus will switch to that window.

2.4 No default focus window, no “none” focus.

Tk 3.6 has the notion of a default focus window, which receives the focus if the focus window is deleted. It is also possible for an application to abandon the input focus by setting the focus to `none`. In Tk 4.0 both of these features have been eliminated. There is no default focus window, and the focus can never be explicitly abandoned. If the focus window is destroyed, Tk resets the input focus to the toplevel containing the old focus window. If the toplevel is destroyed, the window manager will reclaim the focus and move it elsewhere.

If you really want to abandon the focus in Tk 4.0 so that keyboard events are ignored, you can create a dummy window with no key bindings (set its binding tags to an empty string to be sure), make sure that is never mapped, and give it the input focus.

2.5 Better focus events

Tk 3.6 has a quirky event model for `FocusIn` and `FocusOut` events: when the window manager gives the focus to a toplevel, Tk generates a `FocusIn` event for the toplevel and another `FocusIn` event for the focus window, but no events for any other windows. When the window manager moves the focus somewhere else, `FocusOut` events are generated for these same two windows. In Tk 4.0, `FocusIn` and `FocusOut` events are generated in the same way as `Enter` and `Leave` events: when the focus arrives, a `FocusIn` event is generated for each window from the toplevel down to the focus window, with different detail fields for different windows (see Xlib documentation for information on these values). The reverse happens when the focus leaves a window.

2.6 Porting issues

If you didn't have any special focus-related code in Tk 3.6, then you shouldn't need to make any changes for 4.0; things will just work better. If you wrote code in Tk 3.6 to get around the weaknesses with its focus mechanism, then you should remove most or all of that code. For example, if you implemented keyboard traversal yourself, or if you built your own mechanism to remember a separate focus window for each toplevel and give it the input focus whenever the toplevel gets the focus, you can simply remove this code, since Tk 4.0 performs these functions for you. If you wrote code that depends on the weird event model in Tk 3.6, that code will need to be rewritten for Tk 4.0. The Tk 4.0 model is general enough to duplicate any effects that were possible in Tk 3.6.

3 Text widgets

Text widgets have undergone a major overhaul for Tk 4.0 and they have improved in many ways. The changes to text widgets are almost entirely upward-compatible from Tk 3.6.

3.1 Embedded windows.

Tk 3.6 supported two kinds of annotations in texts: marks and tags. In Tk 4.0 a third kind of annotation is available: an embedded window. This allows you to embed other widgets inside a text widget, mixed in with the text. The text widget acts as a geometry manager for these windows, laying them out and wrapping them just as if each embedded window were a single character in the text. You can even have texts with nothing in them but embedded windows. The `window` widget command for text widgets provides several options to manage embedded windows.

3.2 More options for tags.

In Tk 4.0 tags support many new options providing additional control over how information is displayed. Here is a summary of the new options:

- You can now specify tab stops with the `-tabs` option. Each tab stop can use left, center, right, or numeric justification. Tab stops can also be specified for the widget as a whole.
- You can specify justification (left, center or right) with the `-justify` option.

- You can now specify line spacing with three options, `-spacing1`, `-spacing2`, and `-spacing3`, which control the spacing above a line, between wrapped lines, and below a line.
- You can now specify margins with the `-lmargin1`, `-lmargin2`, and `-rmargin` options.
- You can now adjust the vertical position of text (e.g. for superscripts or subscripts) with the `-offset` option.
- You can now specify the wrapping style (word wrapping, character wrapping, or none) with the `-wrap` option.
- You can now request overstriking with the `-overstrike` option.

3.3 Bindings

The default bindings for text widgets have been completely rewritten in Tk 4.0. They now support almost all of the Motif behavior (everything except add mode and secondary selections). They also include a substantial subset of the Emacs bindings for cursor motion and basic editing. The `tk_strictMotif` variable disables the Emacs bindings.

3.4 Miscellaneous new features

In addition to the major changes described above, text widgets also include the following new features:

Horizontal scrolling. Text widgets can now be scrolled horizontally as well as vertically, using the `-xscrollcommand` option and the `xview` widget command.

Searching. Text widgets have a new `search` widget command, which provides efficient searching of text widgets using either exact matching, glob-style matching, or regular expressions. You can search forwards or backwards.

Mark gravity. In Tk 3.6 marks always had “right gravity”, which means they stick to the character on the right side of the mark; if you insert at the position of a mark, the new character goes before the mark. In Tk 4.0 you can specify whether marks have left or right gravity.

Screen information. In Tk 4.0 there are two new widget commands for text widgets that return information about the screen layout. The `dlineinfo` widget command returns the bounding box of a display line (all the information displayed on one line of the window, which may be either a whole line of text or a partial line if wrapping has occurred). The `bbox` widget command returns the screen area occupied by a single character.

Extended insert command. The `insert` widget command now supports an additional argument giving a list of tags to apply to the new characters. You can also include several text and tag arguments in a single `insert` command.

See command. There is a new `see` widget command, which adjusts the view in the widget if needed to ensure that a particular character is visible in the window.

3.5 Porting issues: tag stickiness, change in end

There are two changes in text widgets that may require modifications to Tk 3.6 scripts. The first change has to do with tag stickiness. In Tk 3.6, tags are sticky to the right: if you insert new text just after a tagged range, the new text acquires the tags of the preceding character. If you insert text before a tagged range in Tk 3.6, the new characters do not acquire the tags of the range. In Tk 4.0, tags are not sticky on either side: new text acquires a tag from surrounding characters only if the tag is present on both sides of the insertion position. The sticky behavior in Tk 3.6 was rarely useful and special code was often needed to work around it. You should be able to eliminate this code in Tk 4.0.

The second incompatible change in text widgets is that the index `end` now refers to the position just after the final newline in the text, whereas in Tk 3.6 it referred to the position just before the final newline. This makes it possible to apply tags to the final newline, which was not possible in Tk 3.6, but you may need to modify your scripts if you depend on the old position of `end`.

4 Better Motif compliance

All of the widgets have been modified in Tk 4.0 to improve their Motif compliance. This was done by adding features that were missing and reworking the bindings to comply with Motif conventions. I believe that the widgets are now completely Motif compliant except for the following missing features:

- There is no support for secondary selections.
- There is no support for “add mode” in widgets such as texts and listboxes.
- There is no support for drag and drop.

Please let me know if you find any other discrepancies between the Tk widgets and Motif widgets. We plan to eliminate the remaining incompatibilities over the next year or two.

5 Widget changes

All of the Tk 4.0 widgets have been improved over their 3.6 counterparts, mostly in small and backwards compatible ways. Here is a summary of the widget improvements; see Section 13 for information about incompatible changes.

- All widgets now have a `cget` command, which provides an easier way to retrieve the value of a configuration option. In other situations where configuration options are used, such as for menu entries or text tags, a `cget` command is also available.

- All widgets now have `-highlightthickness`, `-highlightbackground`, and `-highlightcolor` options for displaying a highlight ring when the widget (or one of its descendants) has the input focus.
- Entry widgets now support justification and provide a `-show` option for (not) displaying passwords. They will autosize to fit their text if `-width 0` is specified.
- The label/button family of widgets now supports multiline text and justification, including new options `-wraplength` and `-justify`. These features make the message widget obsolete. There is also a new `-underline` option for highlighting a character for keyboard traversal.
- Listboxes now support all of the Motif selection modes, including single selection, multiple selection, and multiple disjoint selections, via the `-selectmode` option. They will autosize to fit their contents if `-width 0` or `-height 0` is specified. There are new `see`, `bbox`, and `activate` widget commands.
- Canvas polygons now support `-outline` and `-width` options for drawing outlines.
- Scale widgets now support real values as well as integers (see the `-resolution` and `-digits` options), and they have a `-variable` option to link to a Tcl variable. They have two new widget commands, `coords` and `identify`, and their bindings are now defined in Tcl rather than being hardwired in C code as in Tk 3.6.
- Scrollbar widgets now have a new interface to the controlling widget, which provides more flexibility than the old style (but the old style is still supported for compatibility). There is a new option `-jump` to prevent continuous updates while dragging the slider, and a new option `-elementborderwidth` to control the border width of the arrows and slider separately from the widget's outer border. There are four new widget commands, `activate`, `delta`, `fraction`, and `identify`, and the default bindings are now defined in Tcl rather than being hardwired in C code as in Tk 3.6.
- Menu entries now have several new configuration options such as `-foreground` and `-indicatoron`, and tear-off menus have been reimplemented to be more Motif-like. New menu entries can be created in the middle of a menu using the `insert` widget command, and there is a `type` widget command that returns the type of a menu entry.
- Menubuttons now have a `-indicatoron` option for displaying an option menu indicator. There is now support for option menus via the `tk_optionMenu` procedure, and popups are simplified with the `tk_popup` procedure.
- The variable `tk_strictMotif` is used in more places to enforce even stricter Motif compliance.

6 Images

Tk 4.0 contains a general-purpose image mechanism for displaying color pictures and other complex objects. There is a new command, `image`, which may be used to create image objects. For example, the command

```
image create photo myFace -file picture.ppm
```

creates a new image named `myFace`. The image is of type `photo` (a full-color representation that dithers on monochrome or color-mapped displays) and the source data for the image is in the file named `picture.ppm`. Once an image has been created, it can be used in many different places by specifying a `-image` option. For example, the command

```
label .l -image myFace
```

will create a label widget that displays the image, and if `.c` is a canvas widget the command

```
.c create image 400 200 -image myFace
```

will create an image item in the canvas that displays `myFace`.

The image mechanism provides a great deal of flexibility:

- Once an image has been defined, it can be used in many different places, even on different displays.
- Images provide image commands, analogous to widget commands, that can be used to manipulate the image; any changes in an image are automatically reflected in all of its instances.
- There can be many different types of images. Tk 4.0 has two built-in types, `photo` and `bitmap`. Other image types can be defined in C as extensions (see the documentation for the `Tk_CreateImageType` library procedure). The photo image type was implemented by Paul Mackerras, based on his earlier photo widget.
- Within the photo image type, there can be many different file formats. In Tk 4.0, only PPM, PGM, and GIF formats are built-in, but other formats can be added as extensions (see the documentation for the `Tk_CreatePhotoImageFormat` library procedure). Readers for XPM, TIFF, and others are available from the Tcl community.

7 Color management

Tk 3.6 suffers from a relatively weak mechanism for managing colors. It uses only the default colormap for a screen, and if all the entries in that colormap fill up then Tk switches to monochrome mode and “rounds” all future colors to black or white. This approach is becoming increasingly unpleasant because of applications such as Frame and Web browsers that use up all the entries in the default colormap.

Tk 4.0 has a much more powerful color management mechanism. If a colormap fills up, Tk allocates future colors by picking the closest match from the available colors, so that it need not revert to monochrome mode. Tk also manages colors better by delaying color allocation until colors are actually needed; in many cases, such as 3D borders, colors are never needed. When colors are scarce Tk changes the way it displays beveled borders so that it uses stippling instead of additional colors for the light and dark shadows. You can find out whether a colormap has filled up using the new command `winfo colormap-full`.

Tk 4.0 also allows you to allocate new colormaps for toplevel and frame widgets with the `-colormap` option, and you change the visual type in these widgets (with the `-visual` option) to take advantage of visuals other than the default visual for a screen. New commands `winfo visualsavailable` and `wm colormapwindows` have been added to help manage colormaps and visuals.

The default color scheme in Tk 4.0 has changed from a tan palette ("bisque") to a gray palette, which seems to becoming standard for Motif. There is a new Tcl procedure `tk_setPalette` that changes the palette of an application on the fly, and there is also a procedure `tk_bisque` to restore the palette to the old bisque colors.

The Tk 3.6 color model mechanism is no longer necessary so it has been removed in Tk 4.0. If you want to find out whether a screen is monochrome or color, you cannot use the `tk colormodel` command anymore; use `winfo depth` instead.

8 Event handling: fileevent and after

Tk 4.0 contains several improvements in the area of event handling besides those already mentioned for bindings:

- There is a new command `fileevent` for performing event-driven I/O to and from files. The `fileevent` command is modelled very closely after Mark Diekhans' `add-input` extension, which has been used widely with Tk 3.6.
- The `after` command has two new options, `idle` and `cancel`. `After idle` can be used to schedule a script as an "idle handler", which means it runs the next time that Tk enters the event loop and finds no work to do. `After cancel` may be used to delete a previously-scheduled `after` script, so that it will no longer be invoked.

9 Multiple displays

Although Tk has always allowed a single application to open windows on several displays, the support for multiple displays is weak in Tk 3.6. For example, many of the bindings break if users work simultaneously in windows on different displays, and mechanisms like the selection and the input focus have insufficient support for multiple displays.

Tk 4.0 contains numerous modifications to improve the handling of multiple displays. Several commands, such as `selection`, `send`, and `focus`, have a new `-displayof` argument so that you can select a particular display. In addition, the bindings have been reworked to handle interactions occurring simultaneously on different displays. With Tk 4.0 it should be possible to create applications that really use multiple displays gracefully.

10 The send command

The `send` command has been completely overhauled for Tk 4.0 to eliminate several problems in Tk 3.6 and add a number of new features:

- Tk 3.6 aborts a `send` command if no response is received within 5 seconds; this made it very difficult to invoke long-running commands. Tk 4.0 eliminates the timeout and uses a different mechanism to tell if the target application has crashed.
- The `winfo interps` command no longer returns the names of applications that have exited or crashed.
- Asynchronous sends are possible using the `-async` switch.
- Commands can be sent to displays other than that of the root window, using the `-displayof` switch.
- Window server security is now checked on each `send`, so Tk 4.0 deals better with changes in the security of the server.
- More complete error information (including the `errorCode` and `errorInfo` variables) is propagated back to the sender after errors.
- You can query and change the name of an application with the `tk appname` command.

Unfortunately the improvements to the Tk 4.0 `send` mechanism required substantial changes to the transport protocol for sends; this makes it impossible for Tk 4.0 applications to communicate with Tk 3.6 applications via `send`. The new transport protocol is more flexible than the old protocol, so it should be possible to make protocol improvements in an upward-compatible way.

11 The selection and clipboard

In Tk 3.6 the selection mechanism can deal only with the display of the root window and with the primary selection; there is no support for multiple displays, secondary selections, or the clipboard. Tk 4.0 eliminates all of these shortcomings. The `-displayof` option can be used to specify a particular display in the selection command, and there is now full access to all of the X selection types. Tk 4.0 also includes a new `clipboard` command for manipulating the clipboard.

12 Miscellaneous changes

Here is a quick summary of the remaining changes in Tk 4.0:

- The `wish` application has been modified so that the `-file` switch is no longer needed or recommended. This makes `wish` just like `tclsh`, where you specify the script file as the first argument to the program, e.g. `wish foo.tcl`. The `-file` switch is still permitted for backward compatibility, but its use is deprecated.
- `Wish` now sets the application's class from the application name (what appears in the title bar of the window by default), rather than always using `Tk` as the class as in Tk 3.6. This makes application-specific options easier to use.
- Toplevel windows are now resizable by default, whereas in Tk 3.6 they were not. You can use the `wm resizable` command to make windows non-reiszable.
- Tk 4.0 patches around an Xlib bug whereby long-running applications tended to reach the end of the space of X resource ids, wrap around to 0 again, and then crash. Tk now reuses resource identifiers so that wrap-around should never occur.
- There is a new `winfo manager` command that tells which geometry manager is controlling a particular widget.
- There is a new `bell` command that does what its name suggests.
- There are new `winfo pointerx`, `winfo pointery`, and `winfo pointerxy` commands that can be used to query the position of the mouse pointer.

13 Summary of Incompatibilites

This section lists all of the incompatible changes in Tk 4.0 that may require changes in Tcl scripts written for Tcl 3.6. Each incompatibility is described in terms of the problem it produces when you run your Tk 3.6 script under Tk 4.0 and a possible work-around. Only Tcl-level incompatibilities are covered here. For incompatible changes at the C level, see the `README` and `changes` files in the distribution. The problems and solutions are roughly in order of importance, with the most important problems first.

Problem #1: When you change the background color of a widget, a small ring in the default background color remains around the edge of the widget.

Solution: This is the focus traversal highlight, whose color is specified separately from `-background`; use the `-highlightbackground` option to change the color of the highlight. Or, you can set `-highlightthickness` to 0 to eliminate the traversal highlight altogether.

Problem #2: Bindings defined for a widget no longer replace the corresponding class bindings, so unwanted class bindings get invoked in addition to the widget bindings.

Solution: Add a `break` command at the end of the widget binding, or rework the widget binding so that it's OK for the class binding to execute.

Problem #3: Bindings on toplevel windows are invoked when events occur for internal windows inside the toplevels.

Solution: Use the `%W` substitution to extract the name of the window where the event actually occurred, and only execute the rest of the binding script if this matches the name of the toplevel.

Problem #4: The `-command` option for a cascade menu entry is no longer invoked when the submenu is posted.

Solution: Use the `-postcommand` option for the submenu instead.

Problem #5: The `-geometry` option is no longer supported by listboxes, frames, and toplevels.

Solution: Use the `-width` and `-height` options instead.

Problem #6: The procedure `tk_listboxSingleSelect` no longer exists.

Solution: Use the `-selectmode` option on the listbox instead.

Problem #7: Canvases no longer have a `-scrollincrement` option.

Solution: Use the new `-xscrollincrement` and `-yscrollincrement` options instead.

Problem #8: The `tk colormodel` command no longer exists.

Solution: To find out whether a window is monochrome or color, use `winfo depth` to extract the window's depth; a depth of 1 means monochrome.

Problem #9: The class of Tk applications is no longer `Tk`, so options specified for the `Tk` class in your `.Xdefaults` file are no longer used.

Solution: Modify your `.Xdefaults` file (and any Tcl code that sets options) to specify the name of the application (with the first letter capitalized) as the class instead of `Tk`.

Problem #10: When text is added to a text widget just after a tagged area, the new text no longer receives the tag.

Solution: Explicitly tag the new text with the desired tags. If you want the tags on the new text to be the same as those at some other point in the text, you can use the `tag names` widget command to query existing tags.

Problem #11: Widgets appear larger than they did in Tk 3.6.

Solution: There are two issues here. The first is that all widgets now have a focus traversal highlight ring that turns dark when the widget has the focus; this is required for Motif compliance but you can eliminate it by specifying a 0 value for the `-highlightthickness` option. The second issue is that the default padding for buttons and menubuttons has been increased to match the sizes of Motif widgets. If you don't mind being different from Motif, you can set the `-padx` and `-pady` options back to their Tk 3.6 values (use the `configure` widget command in Tk 3.6 to see what the old values were).

Problem #12: Listboxes now return the selection as a string with newlines separating the values, rather than a Tcl, list.

Solution: Modify your code to handle the new format. You can convert the selection back into the old list format with a script like the following:

```
split [selection get] \n
```

Problem #13: Tk 4.0 applications cannot `send` to or be sent from Tk 3.6 applications.

Solution: The only solution is to upgrade all your applications to Tk 4.0.

Problem #14: In texts, `end` now refers to a position just after the final newline, instead of the final newline.

Solution: If you wish to refer to the final newline, use the index `end-1char` instead of `end`.

Problem #15: In entry widgets, `sel.last` now refers to the character just after the last selected one, rather than the last selected one. The second index for the `delete` widget command has changed in the same way.

Solution: Add one to the values used in your scripts.

Problem #16: Because `Any` is implicit in all bindings, bindings trigger when extra modifiers are present, whereas they didn't trigger in Tk 3.6.

Solution: In most cases it's probably fine to ignore the extra modifiers. If you really don't want any actions to be taken when extra modifiers are present, create additional bindings for the cases with extra modifiers, and specify a single blank character (or any script that does nothing) as the script for those bindings. Alternatively, you can use the `%s` substitution to extract the mouse and modifier state in the event binding, then you can test this value for modifiers you do or don't want.

Problem #17: In scrollbars there is no longer a `-foreground` or `-activeforeground` option, and `-background` has a different meaning.

Solution: Use `-troughcolor` everywhere that you used `-background` in Tk 3.6, `-background` everywhere you used to use `-foreground`, and `-activebackground` everywhere you used to use `-activeforeground`.

Problem #18: Options for colors seem to have changed in scale widgets.

Solution: Use `-background` where you used to use `-sliderforeground`, `-troughcolor` where you used to use `-background`, and `-activebackground` everywhere you used to use `-activeforeground`.

Problem #19: Scale widgets no longer accept hexadecimal or octal numbers in the `set` command or the `-from` and `-to` options.

Solution: Use `format` or `expr` to convert the values to decimal.

Problem #20: In checkbuttons, radiobuttons, and menu entries, the `-selector` option no longer exists.

Solution: Use `-selectcolor` instead of `-select`. To specify that no indicator should be drawn at all, use the `-indicatoron` option instead of setting `-select` to an empty string.

Problem #21: The indices of menu entries have changed, and operations on menu entry 0 no longer work.

Solution: This is because menus now have a tearoff entry at the top by default, and this occupies entry 0, so your first entry is now entry 1. You can either set the `-tearoff` option to 0 to eliminate the tearoff entry or add 1 to all the indices you use in your scripts.

Problem #22: The `enable` and `disable` widget commands are no longer supported by menus.

Solution: Use the `-state` configuration option instead.

Problem #23: The `activate` and `deactivate` widget commands are no longer supported by buttons, checkbuttons, radiobuttons, and menus.

Solution: Use the `-state` configuration option instead.

Problem #24: Canvas arc items no longer use the `-fill` and `-stipple` options for drawing when the `-style` option is `arc`.

Solution: Use the `-outline` and `-outlinestipple` options instead.

Problem #25: The variable `tkVersion` no longer exists (it has been obsolete for several releases).

Solution: Use `tk_version` instead.

Problem #26: The syntax of the `scan` widget commands for texts has changed.

Solution: Modify your code to use the new syntax.

Problem #27: `wish` no longer recognizes the `-help` option.

Solution: Implement this option yourself in your `wish` scripts.

Problem #28: Tk 4.0 always prints real numbers such as canvas coordinates with a decimal point. This can cause syntax errors if you later use them in situations where integers are expected.

Solution: Change your code so that real numbers work OK, or use the `expr` command (with the `round` function) to convert the numbers to integers.

Problem #29: The `pack info` command returns different information, and `pack newinfo` no longer exists.

Solution: Use `pack info` where you used to use `pack newinfo`. `Pack info` was obsolete, so it has been eliminated.

Problem #30: The `view` widget command for entries no longer exists, nor does the `-scrollcommand` option.

Solution: Use `xview` where you used to use `view`; use `-xscrollcommand` where you used to use `-scrollcommand`.

Problem #31: The `-padx` and `-pady` options are ignored for the button family of widgets if a bitmap or image is being displayed: the padding is always 0.

Solution: Pack the button inside a frame, with extra padding in the frame. Or, redo the image or bitmap to incorporate padding into it.

Problem #32: In radiobuttons, the `-value` option no longer defaults to the name of the widget; it defaults to an empty string.

Solution: Specify the widget's name explicitly as the value of the option.

Problem #33: The `-menu` option for menubuttons and cascade menu entries may refer only to a child of the menubutton or menu.

Solution: Rename menus to meet this requirement.

Problem #34: The interpretation of `@y` in menus has changed: it never returns `none`, even if the y-coordinate is outside the menu (it returns the index of the closest entry).

Solution: If you care about this distinction, check the y-coordinate explicitly to see if it is less than 0 or greater than or equal to the window's height (use `winfo height` to get the height).

Problem #35: The `invoke` and `activate` widget commands for menus no longer post cascaded submenus.

Solution: Use the `postcascade` widget command to post submenus.

Problem #36: The selection targets `APPLICATION` and `WINDOW_NAME` are no longer supported.

Solution: Use targets `TK_APPLICATION` and `TK_WINDOW` instead.

Problem #37: There is no longer a default focus.

Solution: None: modify your code not to depend on this feature.

Problem #38: The `focus` command now returns an empty string to indicate that the application doesn't have the input focus, instead of `none`.

Solution: Modify your code to check for an empty string instead of `none`.

Problem #39: `FocusIn` and `FocusOut` events are delivered to more windows than they used to be.

Solution: Modify your code to use the new set of events. The old event set was somewhat bizarre, and the new set matches more closely what happens elsewhere, such as with `Enter` and `Leave` events.

Problem #40: `wm maxsize` and `wm minsize` no longer accept empty arguments. This means that you cannot use these commands to make windows non-resizable.

Solution: Use the `wm resizable` command to make windows resizable.

Problem #41: In the placer, if you specify both `-x` and `-relx` then they add, instead of the most recent specification replacing the earlier one. Ditto for `-y` and `-rely`, `-width` and `-relwidth`, and `-height` and `-relheight`.

Solution: If you no longer want one of these options to be used, set it to 0 explicitly.

Problem #42: The command "`focus none`" doesn't work in Tk 4.0.

Solution: Create a dummy widget that is never mapped and set the focus to that widget.

Problem #43: `%D` substitutions are no longer supported in bindings, nor are the event types `CirculateRequest`, `ConfigureRequest`, `MapRequest`, and `ResizeRequest`.

Solution: Use the name of the display instead of %D to identify a display; you can get the display name with the `winfo screen` command. The desupported event types never really worked anyway, so there should be no code that depends on them.

Problem #44: `%` binding substitutions that return window identifiers, such as `%a` and `%S`, now produce hexadecimal results instead of decimal.

Solution: Use the `format` command to turn them back to decimal.

Problem #45: `Enter`, `Leave`, `FocusIn`, and `FocusOut` events with detail `NotifyInferior` are now ignored by the binding mechanism, so they're not visible to Tcl scripts.

Solution: In most cases, Tcl scripts work better if these bindings are ignored. You can still use C code to access these events if you really need them. Or, create bindings on the inferior windows and use `NotifyAncestor` bindings on the children instead of `NotifyInferior` bindings on the parent.