
Chapter 17

Geometry Managers: the Packer

Geometry managers determine the sizes and locations of widgets. Tk is similar to other X toolkits in that it does not allow widgets to determine their own geometries. A widget will not even appear on the screen unless it is managed by a geometry manager. This separation of geometry management from internal widget behavior allows multiple geometry managers to exist simultaneously and permits any widget to be used with any geometry manager. If widgets controlled their own geometry, then this flexibility would be lost: every existing widget would need to be modified to introduce a new style of layout.

This chapter describes the overall structure for geometry management and then presents the packer, which is the most commonly used geometry manager in Tk. The packer makes it easy to generate layouts such as rows or columns of widgets; it can also be used for more complex effects such as “put the menu bar across the top of the window, then the scrollbar across the right side, then fill the remaining space with a text widget.” The pack command is used to communicate with the packer; see Tables 17.1 and 17.2 for a summary of its features.

17.1 An overview of geometry management

A geometry manager’s job is to arrange one or more *slave* widgets relative to a *master* widget. For example, it might arrange three slaves in a row from left to right across the area of the master, or it might arrange two slaves so that they split the space of the master with one slave occupying the top half and the other occupying the bottom half. Different geometry managers embody different styles of layout. The master is usually the parent of

Ousterhout, "Tk and the Tk Toolkit"
Addison-Wesley 1994

<code>pack slave ?slave ...? option value ?option value ...?</code>	Same as the <code>pack configure</code> command described next.
<code>pack configure slave ?slave ...? option value ?option value ...?</code>	Arrange for the packer to manage the geometry of the widgets named by the <code>slave</code> arguments. The <code>option</code> and <code>value</code> arguments provide information that determines the dimensions and positions of the slaves. See Table 17.2 for details.
<code>pack forget slave</code>	Causes the packer to stop managing <code>slave</code> and unmap it from the screen. Has no effect if <code>slave</code> isn't currently managed by the packer.
<code>pack info slave</code>	Returns a list giving the current configuration of <code>slave</code> . The list consists of <code>option-value</code> pairs in exactly the same form as might be specified to the <code>pack configure</code> command. Returns an empty string if <code>slave</code> isn't currently managed by the packer. In Tk releases before 4.0 this action has the name <code>newinfo</code> instead of <code>info</code> .
<code>pack propagate master ?boolean?</code>	<code>boolean</code> specifies whether the packer should set the requested size for <code>master</code> based on the needs of its slaves. If <code>boolean</code> is omitted, returns 0 or 1 to indicate whether geometry propagation is currently enabled for <code>master</code> .
<code>pack slaves master</code>	Returns a list of the slaves on <code>master</code> 's packing list, in order.

Table 17.1. A summary of the `pack` command.

the slave but there are times when it's convenient to use other windows as masters (you will see examples of this later). Masters are usually frame widgets.

A geometry manager receives three pieces of information for use in computing a layout (see Figure 17.1). First, each slave widget requests a particular width and height. These are usually the minimum dimensions needed by the widget to display its information. For example, a button widget requests a size just large enough to display its text or bitmap. Although geometry managers aren't obliged to satisfy the requests made by their slave widgets, they usually do.

The second kind of input for a geometry manager comes from the application designer and is used to control the layout algorithm. The nature of this information varies from geometry manager to geometry manager. For example, with the packer an application designer can name three slaves and request that they be arranged in a row from left to right within the master; the packer checks the requested sizes of the slaves and positions them so that they abut in a row, with each slave given just as much space as it needs.

The third piece of information used by geometry managers is the geometry of the master window. For example, the geometry manager might position a slave at the lower left corner of its master, or it might divide the space of the master among one or more

lue ...?
ts named by
provide
the slaves. See

om the screen.
r.

e list consists of
specified to the
lave isn't
his action has

sted size for
itted, returns 0
nabled for

er.

masters (you

omputing a lay-
and height.
ay its informa-
play its text or
s made by their

plication
ormation varies
er an applica-
row from left to
s and positions
as it needs.
metry of the
e at the lower
one or more

<code>-after window</code>	Use <i>window</i> 's master as the master for the slave and insert the slave into the packing list just after <i>window</i> .
<code>-anchor position</code>	If the slave's parcel is larger than its requested size, this option determines where in the parcel the slave will be positioned. Defaults to <code>center</code> .
<code>-before window</code>	Use <i>window</i> 's master as the master for the slave and insert the slave into the packing list just before <i>window</i> .
<code>-expand boolean</code>	If <i>boolean</i> is a true value, the slave's parcel will grow to absorb any extra space left over in the master. Defaults to <code>false</code> .
<code>-fill style</code>	Specifies whether (and how) to grow the slave if its parcel is larger than the slave's requested size. <i>style</i> must be either <code>none</code> , <code>x</code> , <code>y</code> , or <code>both</code> . Defaults to <code>none</code> .
<code>-in master</code>	Use <i>master</i> as the master for the slave. <i>master</i> must be the slave's parent or a descendant of the slave's parent. If this option is not specified, the master defaults to the slave's parent.
<code>-ipadx distance</code>	<i>distance</i> specifies internal padding for the slave, which is extra horizontal space to allow inside the slave on each side, in addition to what the slave requests. Defaults to 0.
<code>-ipady distance</code>	<i>distance</i> specifies internal padding for the slave, which is extra vertical space to allow inside the slave on each side, in addition to what the slave requests. Defaults to 0.
<code>-padx distance</code>	<i>distance</i> specifies external padding for the slave (extra horizontal space outside the slave but inside its parcel on each side). Defaults to 0.
<code>-pady distance</code>	<i>distance</i> specifies external padding for the slave (extra vertical space outside the slave but inside its parcel on each side). Defaults to 0.
<code>-side side</code>	<i>side</i> specifies which side of the master the slave should be packed against. Must be <code>top</code> , <code>bottom</code> , <code>left</code> , or <code>right</code> . Defaults to <code>top</code> .

Table 17.2. A summary of the configuration options supported by the packer.

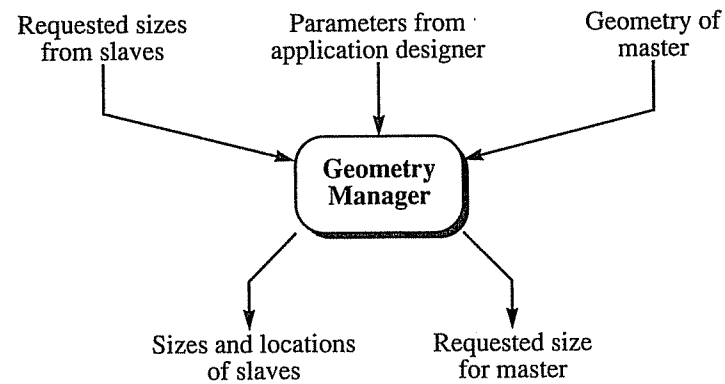


Figure 17.1. A geometry manager receives three kinds of inputs: a requested size for each slave (which usually reflects the information to be displayed in the slave), commands from the application designer (such as “arrange these three windows in a row”), and the actual geometry of the master window. The geometry manager then assigns a size and location to each slave. It may also set the requested size for the master window, which can be used by a higher level geometry manager to manage the master.

slaves, or it might refuse to display a slave altogether if it doesn't fit within the area of its master.

After it has received all of the preceding information, the geometry manager executes a layout algorithm to determine the width, height, and position of each of its slaves. If the geometry manager assigns a slave a size different from what it requested, the widget must make do in the best way it can. Geometry managers usually try to give widgets the space they requested, but they may produce more attractive layouts by giving widgets extra space in some situations. If there isn't enough space in a master for all of its slaves, some of the slaves may get less space than they requested. In extreme cases the geometry manager may choose not to display some slaves at all.

The controlling information for geometry management may change while an application runs. For example, a button might be reconfigured with a different font or bitmap, in which case it will change its requested dimensions. In addition, the geometry manager might be told to use a different approach (for example, arrange a collection of windows from top to bottom instead of left to right), some of the slave windows might be deleted, or the user might interactively resize the master window. When any of these things happens, the geometry manager recomputes the layout.

Some geometry managers will set the requested size for the master window. For example, the packer computes how much space is needed in the master to accommodate all of its slaves in the fashion requested by the application designer. It then sets the requested size for the master to these dimensions, overriding any request made by the

of

master widget itself. This approach allows for hierarchical geometry management, where each master is itself the slave of another higher level master. Size requests pass up through the hierarchy from each slave to its master, resulting ultimately in a size request for a top-level window, which is passed to the window manager. Then actual geometry information passes down through the hierarchy, with the geometry manager at each level using the geometry of a master to compute the geometry of one or more slaves. As a result, the entire hierarchy sizes itself to meet the needs of the lowest level slaves (the master widgets “shrink-wrap” around their slaves).

Each widget can be managed by at most one geometry manager at a time, although it is possible to switch geometry managers during the life of a slave. A widget can act as master to any number of slaves, and it is even possible for different geometry managers to control different groups of slaves within the same master. A single geometry manager can simultaneously manage different groups of slaves associated with different masters.

Only internal widgets may be slaves for geometry management. The techniques described here do not apply to top-level widgets since they are managed by the window manager for the display. See Chapter 22 for information on how to control the geometry of top-level widgets.

or each slave
the application
of the master
y also set the
manager to

17.2 Packer basics

The packer is the most commonly used geometry manager in Tk. It arranges the slaves for a master by positioning them one at a time in the master’s window, working from the edges toward the center. With the packer you can generate rows, columns, and many other arrangements.

The packer maintains a list of slaves for a given master window, called the *packing list*. The packer arranges the slaves by processing the packing list in order, packing one slave in each step. At the time a particular slave is processed, part of the area of the master window has already been allocated to earlier slaves on the list, leaving a rectangular unallocated area for all the remaining slaves, as shown in Figure 17.2(a). The current slave is positioned in three steps: allocate a parcel, stretch the slave, and position it in the parcel.

In the first step a rectangular region called a *parcel* is allocated from the available space. This is done by “slicing” off a piece along one side of the available space. For example, in Figure 17.2(b) a parcel is sliced from the right side of the available space. One dimension of the parcel is determined by the size of the available space and the other dimension is controllable. The controllable dimension is normally taken from the slave’s requested size in that dimension, but the packer allows you to request additional space if you wish. In Figure 17.2(b) the width is the controllable dimension; if the parcel had been sliced from the top or bottom, the parcel’s height would have been controllable instead of its width.

In the second step the packer chooses the dimensions of the slave. By default the slave will get the size it requested, but you can specify instead that it should be stretched

he area of its

ager executes
slaves. If the
s widget must
ets the space
gets extra
slaves, some
ometry man-

ile an applica-
or bitmap, in
y manager
of windows
be deleted, or
ings happens,

low. For
commodate
ets the
de by the

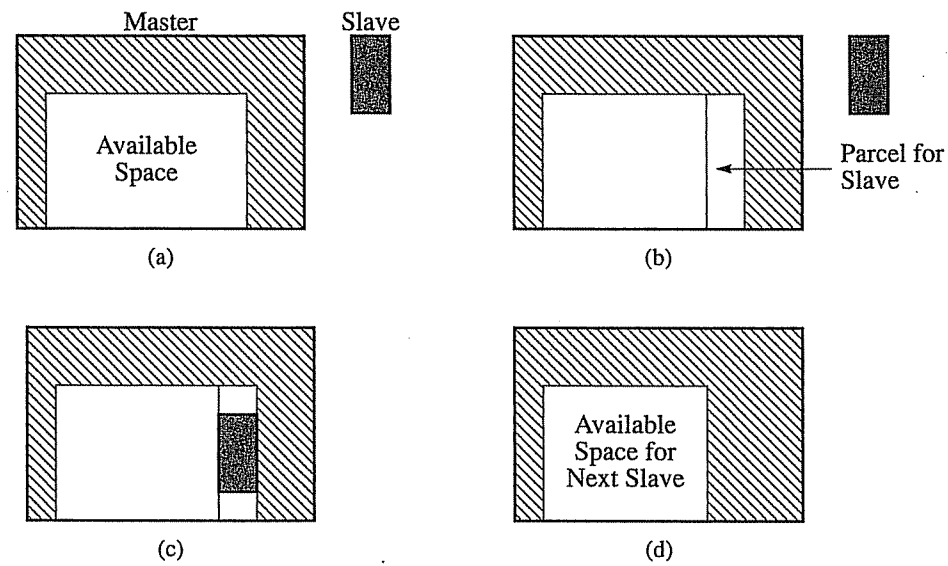


Figure 17.2. The steps taken to pack a single slave. Figure (a) shows the situation before packing a slave. Part of the master's area has already been allocated for previous slaves, and a rectangular region is left for the remaining slaves. The current slave is shown in its requested size. The packer allocates a parcel for the slave along one side of the available space, as shown in (b). The packer may stretch the slave to partially or completely fill the parcel, then it positions the slave over the parcel as in (c). This leaves a smaller rectangular region for the next slave to use, as shown in (d).

in one or both dimensions to fill the space of the parcel. If the slave's requested size is larger than the parcel, it is reduced to fit the size of the parcel.

The third step is to position the slave inside its parcel. If the slave is smaller than the parcel, you can specify an anchor position for the slave such as `n`, `s`, or `center`. In Figure 17.2(c) the slave has been positioned in the center of the parcel, which is the default.

Once the slave has been positioned, a smaller rectangular region is left for the next slave to use, as shown in Figure 17.2(d). If a slave doesn't use all of the space in its parcel, as in Figure 17.2, the leftover space will not be used for later slaves. Thus each step in the packing starts with a rectangular region of available space and ends up with a smaller rectangular region.

17.3 The pack command and -side options

The pack command is used to communicate with the packer. In its simplest form it takes one or more widget names as arguments, followed by one or more pairs of additional arguments that indicate how to manage the widgets. For example, consider the following script, which creates three buttons and arranges them in a row:

```
button .ok -text OK
button .cancel -text Cancel
button .help -text Help
pack .ok .cancel .help -side left
```

The pack command asks the packer to manage .ok, .cancel, and .help as slaves and to pack them in that order. The master for the slaves defaults to their parent, which is the main widget. The option -side left applies to all three of the widgets; it indicates that the parcel for each slave should be allocated on the left side of the available space. Because no other options are specified, the parcel for each slave is allocated just large enough for the slave. This causes the slaves to be arranged in a row from left to right across the master. Furthermore, the packer computes the minimum size needed by the master to accommodate all of its slaves and it sets the master's requested size to these dimensions. The window manager sets the main widget's size as requested, so the master ends up shrink-wrapped around the slaves as shown in Figure 17.3(a).

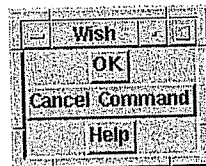
The packer recomputes the layout whenever any of the relevant information changes. For example, if you type the command



(a)



(b)



(c)

Figure 17.3. Simple packer examples. Figure (a) shows the layout produced by the command "pack .ok .cancel .help -side left". If the space needed by a slave changes, the layout changes to match, as in (b). If the information controlling the layout changes (e.g. the -side option is changed to top for each slave), the layout also changes to match, as in (c).

Parcel for Slave

before packing a rectangular size. The packer. The packer have over the shown in (d).

sted size is

aller than the ter. In Fig- s the default. for the next e in its parcel, ch step in the a smaller rect-

```
.cancel configure -text "Cancel Command"
```

to change the text in the middle button, then the button changes its requested size and the packer changes the layout to make more space for the button, as in Figure 17.3(b). If you then type the command

```
pack .ok .cancel .help -side top
```

the layout will change so that each slave is allocated at the top of the remaining space, producing the columnar arrangement shown in Figure 17.3(c). In this case the master's width is chosen to accommodate the largest of the slaves (.cancel). Each of the other slaves receives a parcel wider than it needs and the slave is centered in its parcel.

17.4 Padding

The packer provides four options for requesting extra space for a slave. The extra space is called *padding* and it comes in two forms: *external padding* and *internal padding*. External padding is requested with the `-padx` and `-pady` options; it causes the packer to allocate a parcel larger than requested by the slave and leave extra space around the outside of the slave. For example, the widgets in Figure 17.4(a) are packed with the options

```
-padx 2m -pady 1m
```

These options specify that there should be 2 millimeters of extra space on each side of each of the widgets and 1 millimeter of extra space above and below each widget.

Internal padding is requested with the `-ipadx` and `-ipady` options; it also causes a parcel to be larger than the slave requested, but in this case the slave window is enlarged to incorporate the extra space as in Figure 17.4(b). External and internal padding can be used together as in Figure 17.4(c), and different slaves can have different amounts of padding.

17.5 Filling

If a parcel ends up larger than its slave requested, you can use the `-fill` option to stretch the slave so that it fills the parcel in one or both directions. For example, consider the column of widgets in Figure 17.3(c): each widget is given its requested size, which results in a somewhat ragged look because the buttons are all different sizes. The `-fill` option can be used to make all the buttons the same size:

```
pack .ok .cancel .help -side top -fill x
```

Figure 17.5 shows the effect produced by this command: each of the slaves is stretched horizontally to fill its parcel, and since the parcels are as wide as the master window all of the slaves end up with the same width.

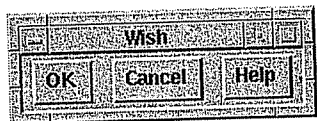
Figure 17.6 shows another simple example of filling. The three windows are configured differently so a separate `pack` command is used for each one. The order of the `pack` commands determines the order of the windows in the packing list. The `.label` widget



pack .ok .cancel .help -side left -padx 2m -pady 1m
(a)



pack .ok .cancel .help -side left -ipadx 2m -ipady 1m
(b)



pack .ok .cancel .help -side left -padx 2m -pady 2m \
-ipadx 2m -ipady 2m
(c)

Figure 17.4. Examples of padding. Each figure shows the results produced by a particular pack command, using the same widgets as in Figure 17.3(a). Figure (a) uses external padding, (b) uses internal padding, and (c) uses both external and internal padding.

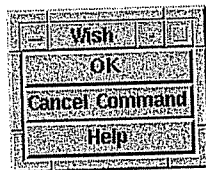


Figure 17.5. The effect of filling. This example is the same as the one in Figure 17.3(c) except that the windows are packed with the additional option -fill x.

is packed first, and it occupies the top part of the master window. The -fill x option specifies that the window should be stretched horizontally so that it fills its parcel. The

: the Packer

size and the .3(b). If you

ing space, pro- aster's width other slaves

extra space is iding. Exter- packer to allo- the outside of tions

ch side of idget. causes a par- enlarged to g can be used s of padding.

tion to stretch nsider the col- hich results in ll option can

is stretched window all of

vs are config- or of the pack abel widget

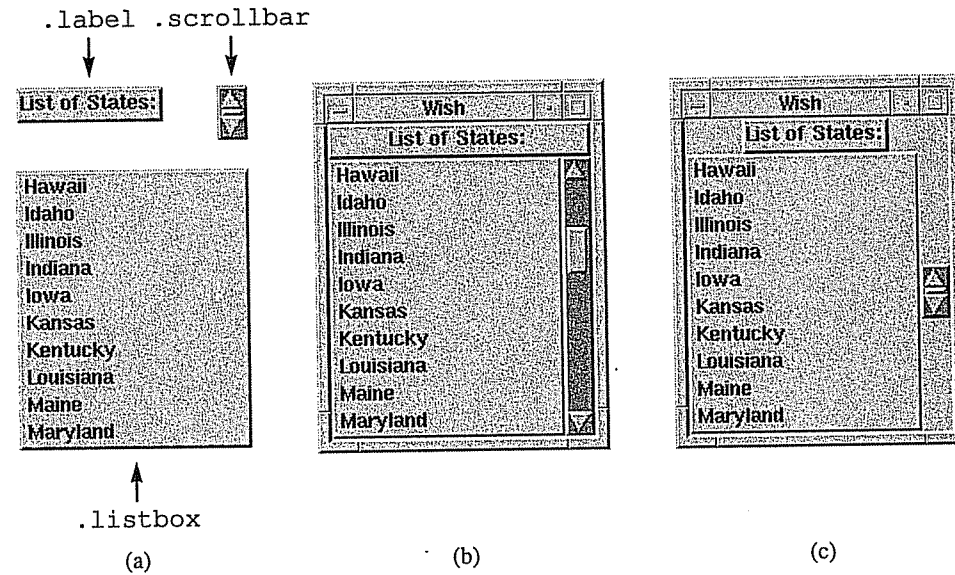


Figure 17.6. Another packer example. Figure (a) shows the requested sizes for three slaves. Figure (b) shows the result of packing the slaves with the script

```
pack .label -side top -fill x
pack .scrollbar -side right -fill y
pack .listbox
```

Figure (c) shows what would have happened without the `-fill` options.

scrollbar widget is packed next, in a similar fashion except that it is arranged against the right side of the window and stretched vertically. The widget `.listbox` is packed last. No options need to be specified for it: it will end up in the same place regardless of which side it is packed against.

17.6 Expansion

Sometimes a master window will have more space than needed by its slaves. This can occur, for example, if a user interactively stretches a window. When this happens the default behavior of the packer is to leave the extra space unused as in Figure 17.7(a). However, you can use the `-expand` option to tell the packer to give any extra space to a particular slave (for example, expand a text widget to fill all the extra space), or to divide the extra space uniformly among a collection of slaves (for example, distribute a collection of buttons uniformly across the available space). The `-expand` option makes it pos-

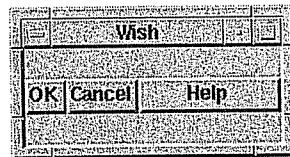
```
pack .ok
pack .he
    -fil
```

P



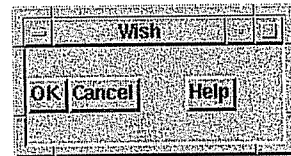
```
pack .ok .cancel .help -side left
```

(a)



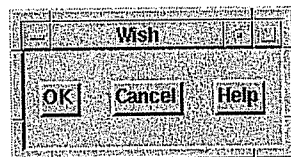
```
pack .ok .cancel -side left
pack .help -side left -expand 1 \
-fill x
```

(b)



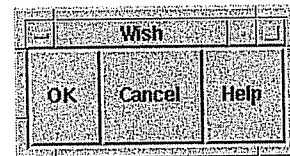
```
pack .ok .cancel -side left
pack .help -side left -expand 1
```

(c)



```
pack .ok .cancel .help \
-side left -expand 1
```

(d)



```
pack .ok .cancel .help -side left \
-expand 1 -fill both
```

(e)

Figure 17.7. The `-expand` option. In (a) the `-expand` option isn't used; in (b) and (c) it is used for a single window, and in (d) and (e) it is used for all of the windows so that extra space is shared. In each of these examples the size of the main widget was fixed with the command `"wm geometry . 160x60"` to simulate what would happen if the user had interactively resized it.

sible to produce layouts that are attractive regardless of how the user resizes the window, so that the user can choose the size he or she prefers.

Figure 17.7 shows how `-expand` works. If you specify `-expand` for a slave, as with `.help` in Figure 17.7(b), the slave's parcel expands to include any extra space in the master. In Figure 17.7(b) `"-fill x"` has also been specified for `.help` so the slave window is stretched to cover the entire width of the parcel. Figure 17.7(c) is similar to Figure 17.7(b) except that no filling is specified, so `.help` is simply centered in its parcel.

ates:

slaves. Figure

i against the
packed last.
less of which

. This can
opens the
: 17.7(a).
tra space to a
, or to divide
ute a collec-
makes it pos-

If you specify `-expand` for multiple slaves, their parcels share the extra space equally. Figure 17.7(d) shows what happens if all of the windows are packed with `-expand` but no filling, and Figure 17.7(e) shows what happens when the slaves are also filled in both dimensions.

Note: The options `-expand` and `-fill` are often confused because of the similarity of their names. The `-expand` option determines whether a parcel absorbs extra space in the master and `-fill` determines whether a slave's window absorbs extra space in its parcel. The options are often used together so that a slave's window absorbs all the extra space in its master.

17.7 Anchors

If a parcel has more space than its slave requested (for example, because you specified `-expand` for it) and if you choose not to stretch the slave with the `-fill` option, the packer will normally center the slave in its parcel. The `-anchor` option allows you to request a different position: its value specifies where the slave should be positioned in the parcel, using one of the forms listed in Section 16.11.2. For example, "`-anchor nw`" specifies that the slave should be positioned in the upper-left corner of its parcel. Figure 17.8(a) shows how `-anchor` can be used to left-justify a collection of buttons.

The `-fill` and `-anchor` options preserve external padding requested with `-padx` and `-pady`. For example, Figure 17.8(b) is the same as Figure 17.8(a) except that external padding has been requested. In this case the buttons aren't positioned at the left edges of their parcels, but instead they are inset by the `-padx` distance.

17.8 Packing order

The `-before` and `-after` options allow you to control a slave's position in the packing list for its master. If you don't specify one of these options, each new slave goes at the end of the packing list. If you specify `-before` or `-after`, its value must be the name of a window that is already packed and the new slave(s) will be positioned just before or after the given slave.

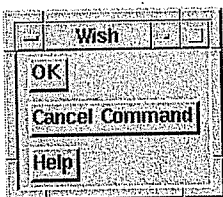
17.9 Hierarchical packing

The packer is often used in hierarchical arrangements where slave windows are also masters for other slaves. Figure 17.9 shows an example of hierarchical packing. The resulting layout contains a column of radiobuttons on the left and a column of checkbuttons on the right, with each group of buttons centered vertically in its column. To achieve this effect two extra frame widgets, `.left` and `.right`, are packed side by side in the main win-



```
pack .ok .cancel .help -side top -anchor w
```

(a)



```
pack .ok .cancel .help -side top -anchor w -padx 2m -pady 1m
```

(b)

Figure 17.8. Examples of the `-anchor` option. Figure (a) shows how `-anchor` can be used to left-justify a collection of buttons; compare to Figure 17.3(c), which is the same except for the absence of the `-anchor` option. Figure (b) shows how `-anchor` preserves external padding.

down, then the buttons are packed inside them. The packer sets the requested sizes for `.left` and `.right` to provide enough space for the buttons, then uses this information to set the requested size for the main widget. The main widget's geometry will be set to the requested size by the window manager, then the packer will arrange `.left` and `.right` inside it, and finally it will arrange the buttons inside `.left` and `.right`.

Figure 17.9 also illustrates a situation where a slave's master is different from its parent (the `-in` option is used to specify a master). It would have been possible to create the button windows as children of `.left` and `.right` (for example, `.left.pts8` instead of `.pts8`) but it is cleaner to create them as children of `.` and then pack them inside `.left` and `.right`. The windows `.left` and `.right` serve no purpose in the application except to help in geometry management; they have the same background color as the main widget, so they are not even visible on the screen. If the buttons were children of their geometry masters, changes to the geometry management (such as adding more levels in the packing hierarchy) might require the button windows to be renamed and would break any code that used the old names (such as entries in `.Xdefaults` files). It is better to give windows names that reflect their logical purpose in the application, build separate frame hierarchies where needed for geometry management, and then pack the functional windows into the frames.

s: the Packer

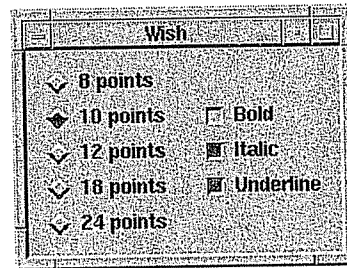
tra space
ed with
slaves are also

larity of their
pace in the
ice in its
s all the extra

u specified
option, the
lows you to
sitioned in the
chor nw"
ancel. Figure
ons.
d with `-padx`
pt that external
left edges of

in the packing
goes at the end
the name of a
before or after

s are also mas-
;. The resulting
buttons on the
eve this effect
the main win-



```

frame .left
frame .right
foreach size {8 10 12 18 24} {
    radiobutton .pts$size -text "$size points" -relief flat \
        -variable pts -value $size
}
checkboxbutton .bold -text Bold -relief flat -variable bold
checkboxbutton .italic -text Italic -relief flat -variable italic
checkboxbutton .underline -text Underline -relief flat \
    -variable underline
pack .left -side left -padx 3m -pady 3m
pack .right -side right -padx 3m -pady 3m
pack .pts8 .pts10 .pts12 .pts18 .pts24 -in .left -side \
    top -anchor w
pack .bold .italic .underline -in .right -side top -anchor w

```

Figure 17.9. An example of hierarchical packing. Two frame widgets, `.left` and `.right`, are used to achieve the column effect.

Note: In the example in Figure 17.9, the frames `.left` and `.right` must be created before the buttons so that the buttons are stacked on top of the frames. The most recently created widget is highest in the stacking order; if `.left` is created after `.pts8` then it will be on top of `.pts8`, so `.pts8` will not be visible. Alternatively, the `raise` and `lower` commands can be used to adjust the stacking order (see Section 26.5).

A slave's master must be either its parent or a descendant of its parent. The reason for this restriction has to do with X's clipping rules. Each window is clipped to the boundaries of its parent, so no portion of a child that lies outside of its parent will be displayed. Tk's restriction on master windows guarantees that a slave will be visible and unclipped if its master is visible and unclipped. Suppose that the restriction were not enforced, so that window `.x.y` could have `.a` as its master. Suppose also that `.a` and `.x` do not overlap at all. If you asked the packer to position `.x.y` in `.a`, the packer would set `.x.y`'s position as requested, but this would cause `.x.y` to be outside the area of `.x` so X would not display it even though `.a` is fully visible. This behavior would be confusing to application

designers so Tk restricts mastership to keep the behavior from occurring. The restriction applies to all of Tk's geometry managers.

17.10 Other options to the pack command

So far the pack command has been discussed in its most common form, where the first arguments are the names of slave windows and the last arguments specify configuration options. Table 17.1 shows several other forms for the pack command, where the first argument selects one of several operations. The "pack configure" command has the same effect as the short form that's been used up until now; the remaining arguments specify windows and configuration options. If "pack configure" (or the short form with no command option) is applied to a window that is already managed by the packer, the slave's configuration is modified; configuration options not specified in the pack command retain their old values.

The command "pack slaves" returns a list of all of the slaves for a given master window. The order of the slaves reflects their order in the packing list:

```
pack slaves .left
=> .pts8 .pts10 .pts12 .pts18 .pts24
```

The command "pack info" returns all of the configuration options for a given slave:

```
pack info .pts8
=> -in .left -anchor w -expand 0 -fill none -ipadx 0
    -ipady 0 -padx 0 -pady 0 -side top
```

The return value is a list consisting of names and values for configuration options in exactly the form you would specify them to pack configure. This command can be used to save the state of a slave so that it can be restored later.

The command "pack forget" causes the packer to stop managing one or more slaves and forget all of its configuration state for them. It also unmaps the windows so that they no longer appear on the screen. This command can be used to transfer control of a window from one geometry manager to another, or simply to remove a window from the screen for a while. If a forgotten window is itself a master for other slaves, the information about those slaves is retained but the slaves won't be displayed on the screen until the master window becomes managed again.

The command "pack propagate" allows you to control whether or not the packer sets the requested size for a master window. Normally the packer sets the requested size for each master window to just accommodate the needs of its slaves, and it updates the requested size as the needs of the slaves change. This feature is called *geometry propagation* and it overrides any size that the master might have requested for itself. You can disable propagation with the command

```
pack propagate master 0
```

at \

.italic

or w

.right, are

ated before
ently created
en it will be
id lower

he reason for
e boundaries
played. Tk's
clipped if its
ed, so that
o not overlap
.x.y's posi-
X would not
to application

where *master* is the name of the master window. This instructs Tk not to set the requested size for *master*, so that the size requested by the master widget itself will be used.

17.11 Other geometry managers in Tk

Although the packer is the most commonly used geometry manager in Tk, there are two other geometry managers that you may also find useful. The first of these is the *placer*. The *placer* is a simple fixed-placement geometry manager that allows you to position each slave at a fixed absolute or relative position in its master. For example, an absolute position might be “one centimeter down from the top edge of the master and two centimeters over from the left edge” and a relative position might be “halfway across the master and two-thirds of the way down”. You can also specify an absolute or relative size for the slave (“one centimeter across” or “half as wide as the master”), or leave the slave with the size it requested internally. See the reference documentation for complete details. The *placer* considers each slave independently so it’s hard to achieve arrangements like rows or columns with the *placer*, and layouts made with the *placer* may not be as flexible as those with the packer (for example, if you stretch a window the layout may not stretch to match). For these reasons the *placer* is used only for a few special purposes.

The third geometry manager in Tk is part of the canvas widget: it allows you to mix embedded widgets with other graphical element such as lines and text. Canvas widgets are described in more detail in Chapter 19.