*Python\_Tkinter\_Programming\_c04*

***PART*** *2 Displays*

*In this section of the book we are going to examine the components that are used to build an appli- cation. We will begin with Tkinter widgets in chapter 4 and an explanation of their key features and their relationship to the underlying Tk widgets they are driving. Remember that Tkinter provides an object-oriented approach to GUIs, so that even though the behavior of the widgets is the same as those widgets created within a Tcl/Tk program, the methods used to create and manipulate them are quite different from within a Tkinter program.*

*En esta sección del libro vamos a examinar los componentes que se utilizan para crear una aplicación. Comenzaremos con los widgets de Tkinter en el capítulo 4 y una explicación de sus características clave y su relación con los widgets de Tk subyacentes que impulsan. Recuerde que Tkinter proporciona un enfoque orientado a objetos para las GUI, de modo que aunque el comportamiento de los widgets es el mismo que el de los widgets creados dentro de un programa Tcl/Tk, los métodos utilizados para crearlos y manipularlos son bastante diferentes desde dentro de Tkinter. programa.*

*Once we have looked at the widgets and examined Pmw (Python MegaWidgets), which provides a valuable library of application-ready widgets, we will discuss laying out the screen using the various geometry managers that are defined in chapter 5.*

*Una vez que hayamos visto los widgets y examinado Pmw (Python MegaWidgets), que proporciona una valiosa biblioteca de widgets listos para aplicaciones, discutiremos el diseño de la pantalla utilizando los diversos administradores de geometría que se definen en el capítulo 5.*

*Chapter 6 explains how to make your application react to external events. This is an important chapter, since it covers a variety of methods for handling user input.*

*Chapter 7 shows the application of classes and inheritance as they apply to Tkinter. This is impor- tant for programmers new to object-oriented programming and it may be useful for those who are used to OOP as it applies to C++ and Java, since there are some notable differences. Then, in chapter 8, I will introduce more advanced techniques to drive a variety of dialogs and other interaction models.*

*Chapter 9 introduces panels and machines; this may be a new idea to some readers. It shows how to construct innovative user interfaces which resemble (in most cases) the devices that they control or monitor.*

*Chapter 10 gives information on building interfaces that permit the user to draw objects on a screen. It then explains methods to change their properties. You will also find some example code which illustrates how Tcl/Tk programs from the demonstration programs distributed with the software can be converted to Tkinter quite easily. Chapter 11 explains how to draw graphs using fairly conventional two-dimensional plots along with some alternative three-dimensional graphics.*

***CHAPTER 4*** *Tkinter widgets*

*4.1 Tkinter widget tour 31  
4.2 Fonts and colors 47  
4.3 Pmw Megawidget tour 49*

*4.4 Creating new megawidgets 73*

*In this chapter I’ll present the widgets and facilities available to Tkinter. Pmw Python Mega- Widgets, will also be discussed, since they provide valuable extensions to Tkinter. Each Tkinter and Pmw widget will be shown along with the source code fragment that produces the display. The examples are short and simple, although some of them illustrate how easy it is to produce powerful graphics with minimal code.*

*En este capítulo presentaré los widgets y las funciones disponibles para Tkinter. También se analizarán los megawidgets de Pmw Python, ya que proporcionan valiosas extensiones para Tkinter. Cada widget de Tkinter y Pmw se mostrará junto con el fragmento de código fuente que produce la visualización. Los ejemplos son breves y sencillos, aunque algunos ilustran lo fácil que es producir gráficos potentes con un código mínimo.*

*This chapter will not attempt to document all of the options available to a Tkinter pro- grammer; complete documentation for the options and methods available for each widget is presented in appendix B. Similarly, Pmw options and methods are documented in Appendix C. Uses these appendices to determine the full range of options for each widget.*

*Este capítulo no intentará documentar todas las opciones disponibles para un programador de Tkinter; La documentación completa de las opciones y métodos disponibles para cada widget se presenta en el apéndice B. De manera similar, las opciones y métodos de PMW se documentan en el Apéndice C. Utiliza estos apéndices para determinar la gama completa de opciones para cada widget.*

***4.1 Tkinter widget tour***

*The following widget displays show typical Tkinter widget appearance and usage. The code is kept quite short, and it illustrates just a few of the options available for the widgets. Some- times one or more of a widget’s methods will be used, but this only scratches the surface. If you need to look up a particular method or option, refer to appendix B. Each widget also has references to the corresponding section in the appendix.*

*Las siguientes pantallas de widgets muestran la apariencia y el uso típicos de los widgets de Tkinter. El código es bastante breve e ilustra solo algunas de las opciones disponibles para los widgets. A veces se utilizarán uno o más de los métodos de un widget, pero esto sólo araña la superficie. Si necesita buscar un método u opción en particular, consulte el apéndice B. Cada widget también tiene referencias a la sección correspondiente en el apéndice.*

*With the exception of the first example, the code examples have been stripped of the boil- erplate code necessary to import and initialize Tkinter. The constant code is shown bolded in the first example. Note that most of the examples have been coded as functions, rather than classes. This helps to keep the volume of code low. The full source code for all of the displays is available online.*

*Con la excepción del primer ejemplo, a los ejemplos de código se les ha quitado el código repetitivo necesario para importar e inicializar Tkinter. El código constante se muestra en negrita en el primer ejemplo. Tenga en cuenta que la mayoría de los ejemplos se han codificado como funciones, en lugar de clases. Esto ayuda a mantener bajo el volumen de código. El código fuente completo de todas las pantallas está disponible en línea.*

***4.1.1 Toplevel***

*The Toplevel widget provides a separate container for other widgets, such as a Frame. For simple, single-window applications, the root Toplevel created when you initialize Tk may be the only shell that you need. There are four types of toplevels shown in figure 4.1:*

*El widget Toplevel proporciona un contenedor independiente para otros widgets, como un marco. Para aplicaciones simples de ventana única, la raíz Toplevel creada cuando inicializa Tk puede ser el único shell que necesita. Hay cuatro tipos de niveles superiores que se muestran en la figura 4.1:*

* ***1*** *The main toplevel, which is normally referred to as the root.*
* *El nivel superior principal, que normalmente se denomina raíz.*
* ***2*** *A child toplevel, which acts independently to the root, unless the root is destroyed, in which case the child is also destroyed.*
* *Un nivel superior secundario, que actúa independientemente de la raíz, a menos que se destruya la raíz, en cuyo caso el hijo también se destruye.*
* ***3*** *A transient toplevel, which is always drawn on top of its parent and is hidden if the par- ent is iconified or withdrawn.*
* *Un nivel superior transitorio, que siempre se dibuja encima de su padre y se oculta si el padre se iconifica o se retira.*
* ***4*** *A Toplevel which is undecorated by the window manager can be created by setting the overrideredirect flag to a nonzero value. This creates a window that cannot be resized or moved directly.*
* *Se puede crear un nivel superior que no esté decorado por el administrador de ventanas estableciendo el indicador overrideredirect en un valor distinto de cero. Esto crea una ventana que no se puede cambiar de tamaño ni mover directamente.*

*from Tkinter import \**

***root = Tk()***

***root.option\_readfile('optionDB')***

***root.title('Toplevel')***

*Label(root, text='This is the main (default) Toplevel').pack(pady=10) t1 = Toplevel(root)  
 Label(t1, text='This is a child of root').pack(padx=10, pady=10)  
 t2 = Toplevel(root)*

*Label(t2, text='This is a transient window of root').pack(padx=10, pady=10)*

*t2.transient(root)*

*t3 = Toplevel(root, borderwidth=5, bg='blue')*

*Label(t3, text='No wm decorations', bg='blue', fg='white').pack(padx=10,*

*pady=10)*

*t3.overrideredirect(1)*

*t3.geometry('200x70+150+150')*

***root.mainloop()***

***Note***

*The use of the option\_readfile call in each of the examples to set application-wide defaults for colors and fonts is explained in “Setting application-wide default fonts and colors” on page 49. This call is used to ensure that most examples have consistent fonts and predictable field sizes.*

*El uso de la llamada option\_readfile en cada uno de los ejemplos para establecer valores predeterminados de colores y fuentes en toda la aplicación se explica en “Configuración de fuentes y colores predeterminados en toda la aplicación” en la página 49. Esta llamada se utiliza para garantizar que la mayoría de los ejemplos tengan fuentes consistentes. y tamaños de campo predecibles.*

*Documentation for the Toplevel widget starts on page 539.*

***4.1.2 Frame***

*Frame widgets are containers for other widgets. Although you can bind mouse and keyboard events to callbacks, frames have limited options and no methods other than standard widget options.*

*Los widgets de marco son contenedores para otros widgets. Aunque puede vincular eventos de mouse y teclado a devoluciones de llamada, los marcos tienen opciones limitadas y no tienen más métodos que las opciones de widget estándar.*

*One of the most common uses for a frame is as a master for a group of widgets which will be handled by a geometry manager. This is shown in figure 4.2. The second frame example, shown in figure 4.3 below, uses one frame for each row of the display.*

*Uno de los usos más comunes de un marco es como maestro para un grupo de widgets que serán manejados por un administrador de geometría. Esto se muestra en la figura 4.2. El segundo ejemplo de cuadro, que se muestra en la figura 4.3 a continuación, utiliza un cuadro para cada fila de la pantalla.*

*for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:*

*f = Frame(root, borderwidth=2, relief=relief)*

*Label(f, text=relief, width=10).pack(side=LEFT)*

*f.pack(side=LEFT, padx=5, pady=5)*

*In a similar manner to buttons and labels, the appearance of the frame can be modified by choosing a relief type and applying an appropriate borderwidth. (See figure 4.3.) In fact, it can be hard to tell the difference between these widgets. For this reason, it may be a good idea to reserve particular decorations for single widgets and not allow the decoration for a label to be used for a button, for example:*

*De manera similar a los botones y etiquetas, la apariencia del marco se puede modificar eligiendo un tipo de relieve y aplicando un ancho de borde apropiado. (Consulte la figura 4.3.) De hecho, puede resultar difícil distinguir entre estos widgets. Por esta razón, puede ser una buena idea reservar decoraciones particulares para widgets individuales y no permitir que la decoración de una etiqueta se use para un botón, por ejemplo:*

*class GUI:*

*def \_\_init\_\_(self):*

*of = [None]\*5*

*for bdw in range(5):*

*of[bdw] = Frame(self.root, borderwidth=0)  
 Label(of[bdw], text='borderwidth = %d ' % bdw).pack(side=LEFT) ifx = 0  
 iff = []  
 for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:*

*TKINTER WIDGET TOUR 33*

*iff.append(Frame(of[bdw], borderwidth=bdw, relief=relief)) Label(iff[ifx], text=relief, width=10).pack(side=LEFT) iff[ifx].pack(side=LEFT, padx=7-bdw, pady=5+bdw)  
 ifx = ifx+1*

*of[bdw].pack()*

*34*

*CHAPTER 4 TKINTER WIDGETS*

***Figure 4.3 Frame styles combining relief type with varying borderwidths***

*A common use of the GROOVE relief type is to provide a labelled frame (sometimes called a panel) around one or more widgets. There are several ways to do this; figure 4.4 illustrates just one example, using two frames. Note that the outer frame uses the Placer geometry man- ager to position the inner frame and label. The widgets inside the inner frame use the Packer geometry manager.*

*Un uso común del tipo de relieve GROOVE es proporcionar un marco etiquetado (a veces llamado panel) alrededor de uno o más widgets. Hay varias formas de hacer esto; La figura 4.4 ilustra sólo un ejemplo, utilizando dos marcos. Tenga en cuenta que el marco exterior utiliza el administrador de geometría Placer para colocar el marco interior y la etiqueta. Los widgets dentro del marco interior utilizan el administrador de geometría Packer.*

***Figure 4.4 Using a Frame widget to construct a panel***

*f = Frame(root, width=300, height=110)*

*xf = Frame(f, relief=GROOVE, borderwidth=2)*

*Label(xf, text="You shot him!").pack(pady=10)*

*Button(xf, text="He's dead!", state=DISABLED).pack(side=LEFT, padx=5,*

*pady=8) Button(xf, text="He's completely dead!", command=root.quit).pack(side=RIGHT,*

*padx=5, pady=8)*

*xf.place(relx=0.01, rely=0.125, anchor=NW)  
 Label(f, text='Self-defence against fruit').place(relx=.06, rely=0.125,*

*anchor=W)*

*f.pack()*

***Documentation for the Frame widget starts on page 491.***

***4.1.3 Label***

*Label widgets are used to display text or images. Labels can contain text spanning multiple lines, but you can only use a single font. You can allow the widget to break a string of text fit- ting the available space or you can embed linefeed characters in the string to control breaks. Several labels are shown in figure 4.5.*

*Los widgets de etiquetas se utilizan para mostrar texto o imágenes. Las etiquetas pueden contener texto que abarque varias líneas, pero solo puedes usar una única fuente. Puede permitir que el widget rompa una cadena de texto que se ajuste al espacio disponible o puede incrustar caracteres de salto de línea en la cadena para controlar los saltos. En la figura 4.5 se muestran varias etiquetas.*

***Figure 4.5 Label widget***

*Although labels are not intended to be used for interacting with users, you can bind mouse and keyboard events to callbacks. This may be used as a “cheap” button for certain applications.*

*Aunque las etiquetas no están diseñadas para interactuar con los usuarios, puede vincular eventos de mouse y teclado a devoluciones de llamada. Esto puede usarse como un botón "barato" para ciertas aplicaciones.*

*Label(root, text="I mean, it's a little confusing for me when you say "*

*"'dog kennel' if you want a mattress. Why not just say 'mattress'?",*

*wraplength=300, justify=LEFT).pack(pady=10)*

*f1=Frame(root)*

*Label(f1, text="It's not working, we need more!",*

*relief=RAISED).pack(side=LEFT, padx=5)  
 Label(f1, text="I'm not coming out!", relief=SUNKEN).pack(side=LEFT,*

*f1.pack()*

*f2=Frame(root)*

*for bitmap,rlf in [ ('woman',RAISED),('mensetmanus',SOLID),*

*('terminal',SUNKEN), ('escherknot',FLAT),*

*('calculator',GROOVE),('letters',RIDGE)]:*

*Label(f2, bitmap='@bitmaps/%s' % bitmap, relief=rlf).pack(side=LEFT,*

*f2.pack()*

***Documentation for the Label widget starts on page 495.***

*TKINTER WIDGET TOUR*

*35*

*padx=5)*

*padx=5)*

***4.1.4 Button***

*Strictly, buttons are labels that react to mouse and key- board events. You bind a method call or callback that is invoked when the button is activated. Buttons may be disabled to prevent the user from activating a button. Button widgets can contain text (which can span mul- tiple lines) or images. Buttons can be in the tab group, which means that you can navigate to them using the TAB key. Simple buttons are illustrated in figure 4.6.*

*Estrictamente, los botones son etiquetas que reaccionan a los eventos del mouse y del teclado. Vincula una llamada a un método o una devolución de llamada que se invoca cuando se activa el botón. Los botones pueden desactivarse para evitar que el usuario active un botón. Los widgets de botones pueden contener texto (que puede abarcar varias líneas) o imágenes. Los botones pueden estar en el grupo de pestañas, lo que significa que puede navegar hasta ellos usando la tecla TAB. Los botones simples se ilustran en la figura 4.6.*

*Label(root, text="You shot him!").pack(pady=10)  
 Button(root, text="He's dead!", state=DISABLED).pack(side=LEFT) Button(root, text="He's completely dead!",*

*command=root.quit).pack(side=RIGHT)*

*Not all GUI programmers are aware that the relief option may be used to create buttons with different appearances. In particular, FLAT and SOLID reliefs are useful for creating toolbars where icons are used to convey functional information. However, some care must be exercised when using some relief effects. For example, if you define a button with a SUNKEN relief, the widget will not have a different appearance when it is activated, since the default behavior is to show the button with a SUNKEN relief; alternative actions must be devised such as changing the background color, font or wording within the button. Figure 4.7 illustrates the effect of com- bining the available relief types with increasing borderwidth. Note that increased borderwidth can be effective for some relief types (and RIDGE and GROOVE don’t work unless borderwidth is 2 or more). However, buttons tend to become ugly if the borderwidth is too great.*

*No todos los programadores de GUI saben que la opción de relieve se puede utilizar para crear botones con diferentes apariencias. En particular, los relieves PLANOS y SÓLIDOS son útiles para crear barras de herramientas donde se utilizan iconos para transmitir información funcional. Sin embargo, se debe tener cierto cuidado al utilizar algunos efectos de alivio. Por ejemplo, si defines un botón con un relieve HUNDIDO, el widget no tendrá una apariencia diferente cuando se active, ya que el comportamiento predeterminado es mostrar el botón con un relieve HUNDIDO; Se deben idear acciones alternativas, como cambiar el color de fondo, la fuente o el texto dentro del botón. La Figura 4.7 ilustra el efecto de combinar los tipos de relieve disponibles con un ancho de frontera creciente. Tenga en cuenta que aumentar el ancho del borde puede ser efectivo para algunos tipos de relieve (y RIDGE y GROOVE no funcionan a menos que el ancho del borde sea 2 o más). Sin embargo, los botones tienden a volverse feos si el ancho del borde es demasiado grande.*

***Figure 4.7 Combining relief and varying borderwidth***

*class GUI:*

*def \_\_init\_\_(self):*

*of = [None] \*5*

*for bdw in range(5):*

*of[bdw] = Frame(self.root, borderwidth=0)*

*Label(of[bdw], text='borderwidth = %d' % bdw).pack(side=LEFT)*

*for relief in [RAISED, SUNKEN, FLAT, RIDGE, GROOVE, SOLID]:*

*Button(of[bdw], text=relief,*

*borderwidth=bdw, relief=relief, width=10,*

*command=lambda s=self, r=relief, b=bdw: s.prt(r,b))\*

*.pack(side=LEFT, padx=7-bdw, pady=7-bdw)*

*of[bdw].pack()*

*def prt(self, relief, border):*

*print '%s:%d' % (relief, border)*

*Documentation for the Button widget starts on page 453.*

***4.1.5 Entry***

*Entry widgets are the basic widgets used to collect input from a user. They may also be used to display information and may be disabled to prevent a user from changing their values.*

*Entry widgets are limited to a single line of text which can be in only one font. A typical entry widget is shown in figure 4.8. If the text entered into the widget is longer than the avail- able display space, the widget scrolls the contents. You may change the visible position using the arrow keys. You may also use the widget’s scrolling methods to bind scrolling behavior to the mouse or to your application.*

***Figure 4.8 Entry widget***

*Label(root, text="Anagram:").pack(side=LEFT, padx=5, pady=10)*

*e = StringVar()*

*Entry(root, width=40, textvariable=e).pack(side=LEFT)*

*e.set("'A shroe! A shroe! My dingkom for a shroe!'")*

*Documentation for the Entry widget starts on page 484.* ***4.1.6 Radiobutton***

***Figure 4.9 Radiobutton widget***

*The Radiobutton widget may need renaming soon! It is becoming unusual to see car radios with mechanical button selectors, so it might be difficult to explain the widget to future GUI designers. However, the idea is that all selections are exclusive, so that selecting one button deselects any button already selected.*

*In a similar fashion to Button widgets, Radiobuttons can dis- play text or images and can have text which spans multiple lines, although in one font only. Figure 4.9 illustrates typical Radiobuttons.*

*You normally associate all of the radiobuttons in a group to a single variable.*

*var = IntVar()*

*for text, value in [('Passion fruit', 1), ('Loganberries', 2),*

*('Mangoes in syrup', 3), ('Oranges', 4),*

*('Apples', 5),('Grapefruit', 6)]: Radiobutton(root, text=text, value=value, variable=var).pack(anchor=W)*

*var.set(3)*

*TKINTER WIDGET TOUR 37*

*If the indicatoron flag is set to FALSE, the radiobutton group behaves as a button box, as shown in figure 4.10. The selected button is normally indicated with a SUNKEN relief.*

*var = IntVar()*

*for text, value in [('Red Leicester', 1), ('Tilsit', 2), ('Caerphilly', 3),*

*('Stilton', 4), ('Emental', 5),*

*('Roquefort', 6), ('Brie', 7)]:*

*Radiobutton(root, text=text, value=value, variable=var,*

*var.set(3)*

***indicatoron=0****).pack(anchor=W, fill=X, ipadx=18)*

*38*

*CHAPTER 4 TKINTER WIDGETS*

***Figure 4.10 Radiobuttons: indicatoron=0***

*Documentation for the Radiobutton widget starts on page 519.* ***4.1.7 Checkbutton***

*Checkbutton widgets are used to provide on/off selections for one or more items. Unlike radiobuttons (see “Radiobutton” on page 37) there is no interaction between checkbuttons. You may load checkbuttons with either text or images. Checkbuttons should normally have a variable (IntVar) assigned to the variable option which allows you to determine the state of the checkbutton. In addition (or alternately) you may bind a callback to the button which will be called whenever the button is pressed.*

*Note that the appearance of checkbuttons is quite different on UNIX and Windows; UNIX normally indicates selection by using a fill color, whereas Windows uses a checkmark. The Windows form is shown in figure 4.11.*

***Figure 4.11 Checkbutton widget***

*for castmember, row, col, status in [*

*('John Cleese', 0,0,NORMAL), ('Eric Idle', 0,1,NORMAL),*

*('Graham Chapman', 1,0,DISABLED), ('Terry Jones', 1,1,NORMAL),*

*('Michael Palin',2,0,NORMAL), ('Terry Gilliam', 2,1,NORMAL)]: setattr(var, castmember, IntVar())  
 Checkbutton(root, text=castmember, state=status, anchor=W, variable = getattr(var, castmember)).grid(row=row, col=col, sticky=W)*

*Documentation for the Checkbutton widget starts on page 481.* ***4.1.8 Menu***

*Menu widgets provide a familiar method to allow the user to choose operations within an application. Menus can be fairly cumbersome to construct, especially if the cascades walk out several levels (it is usually best to try design menus so that you do not need to walk out more than three levels to get to any functionality).*

*Tkinter provides flexibility for menu design, allowing multiple fonts, images and bit- maps, and checkbuttons and radiobuttons. It is possible to build the menu in several schemes. The example shown in figure 4.12 is one way to build a menu; you will find an alternate scheme to build the same menu online as altmenu.py.*

***Figure 4.12 Menu widget***

*Figure 4.13 illustrated adding Button commands to menu.*

***Figure 4.13 Menu: Button commands***

*mBar = Frame(root, relief=RAISED, borderwidth=2)*

*mBar.pack(fill=X)*

*CmdBtn = makeCommandMenu()*

*CasBtn = makeCascadeMenu()*

*ChkBtn = makeCheckbuttonMenu()*

*RadBtn = makeRadiobuttonMenu()*

*NoMenu = makeDisabledMenu()*

*mBar.tk\_menuBar(CmdBtn, CasBtn, ChkBtn, RadBtn, NoMenu)*

*def makeCommandMenu():*

*CmdBtn = Menubutton(mBar, text='Button Commands', underline=0)*

*TKINTER WIDGET TOUR 39*

*CmdBtn.pack(side=LEFT, padx="2m")*

*CmdBtn.menu = Menu(CmdBtn)*

*CmdBtn.menu.add\_command(label="Undo")*

*CmdBtn.menu.entryconfig(0, state=DISABLED)*

*CmdBtn.menu.add\_command(label='New...', underline=0, command=new\_file) CmdBtn.menu.add\_command(label='Open...', underline=0, command=open\_file) CmdBtn.menu.add\_command(label='Wild Font', underline=0,*

*font=('Tempus Sans ITC', 14), command=stub\_action)*

*CmdBtn.menu.add\_command(bitmap="@bitmaps/RotateLeft")*

*CmdBtn.menu.add('separator')*

*CmdBtn.menu.add\_command(label='Quit', underline=0,*

*background='white', activebackground='green',*

*command=CmdBtn.quit)*

*CmdBtn['menu'] = CmdBtn.menu*

*return CmdBtn*

*Figure 4.14 shows the appearance of Cascade menu entries.*

*def makeCascadeMenu():*

*CasBtn = Menubutton(mBar, text='Cascading Menus', underline=0)*

*CasBtn.pack(side=LEFT, padx="2m")*

*CasBtn.menu = Menu(CasBtn)*

*CasBtn.menu.choices = Menu(CasBtn.menu)*

*CasBtn.menu.choices.wierdones = Menu(CasBtn.menu.choices)*

*CasBtn.menu.choices.wierdones.add\_command(label='Stockbroker') CasBtn.menu.choices.wierdones.add\_command(label='Quantity Surveyor') CasBtn.menu.choices.wierdones.add\_command(label='Church Warden') CasBtn.menu.choices.wierdones.add\_command(label='BRM') CasBtn.menu.choices.add\_command(label='Wooden Leg') CasBtn.menu.choices.add\_command(label='Hire Purchase') CasBtn.menu.choices.add\_command(label='Dead Crab') CasBtn.menu.choices.add\_command(label='Tree Surgeon') CasBtn.menu.choices.add\_command(label='Filing Cabinet')*

***Figure 4.14 Menu: Cascade***

*40*

*CHAPTER 4 TKINTER WIDGETS*

*CasBtn.menu.choices.add\_command(label='Goldfish') CasBtn.menu.choices.add\_cascade(label='Is it a...', menu=CasBtn.menu.choices.wierdones) CasBtn.menu.add\_cascade(label='Scipts', menu=CasBtn.menu.choices) CasBtn['menu'] = CasBtn.menu*

*return CasBtn*

*Check buttons may be used within a menu, as shown in figure 4.15.*

***Figure 4.15 Menu: Checkbuttons***

*def makeCheckbuttonMenu():*

*ChkBtn = Menubutton(mBar, text='Checkbutton Menus', underline=0)*

*ChkBtn.pack(side=LEFT, padx='2m')*

*ChkBtn.menu = Menu(ChkBtn)*

*ChkBtn.menu.add\_checkbutton(label='Doug')*

*ChkBtn.menu.add\_checkbutton(label='Dinsdale')*

*ChkBtn.menu.add\_checkbutton(label="Stig O'Tracy")*

*ChkBtn.menu.add\_checkbutton(label='Vince')*

*ChkBtn.menu.add\_checkbutton(label='Gloria Pules')*

*ChkBtn.menu.invoke(ChkBtn.menu.index('Dinsdale'))*

*ChkBtn['menu'] = ChkBtn.menu*

*return ChkBtn*

*An alternative is to use Radiobuttons in a menu, as illustrated in figure 4.16.*

*def makeRadiobuttonMenu():*

*RadBtn = Menubutton(mBar, text='Radiobutton Menus', underline=0)*

*RadBtn.pack(side=LEFT, padx='2m')*

*RadBtn.menu = Menu(RadBtn)*

*RadBtn.menu.add\_radiobutton(label='metonymy')*

*RadBtn.menu.add\_radiobutton(label='zeugmatists')*

*RadBtn.menu.add\_radiobutton(label='synechdotists')*

*RadBtn.menu.add\_radiobutton(label='axiomists')*

*RadBtn.menu.add\_radiobutton(label='anagogists')*

*RadBtn.menu.add\_radiobutton(label='catachresis')*

*RadBtn.menu.add\_radiobutton(label='periphrastic')*

*RadBtn.menu.add\_radiobutton(label='litotes')*

*RadBtn.menu.add\_radiobutton(label='circumlocutors')*

*RadBtn['menu'] = RadBtn.menu*

*return RadBtn*

*TKINTER WIDGET TOUR 41*

*def makeDisabledMenu():*

*Dummy\_button = Menubutton(mBar, text='Disabled Menu', underline=0)*

*Dummy\_button.pack(side=LEFT, padx='2m')*

*Dummy\_button["state"] = DISABLED*

*return Dummy\_button*

*Documentation for the Menu widget starts on page 501. Documentation for the Menubutton widget starts on page 506. Documentation for the OptionMenu class starts on page 510.*

***Figure 4.16 Menu: Radiobuttons***

***4.1.9 Message***

*The Message widget provides a convenient way to present multi-line text. You can use one font and one foreground/background color combination for the complete message. An exam- ple using this widget is shown in figure 4.17.*

*The widget has the standard widget methods.*

*Message(root, text="Exactly. It's my belief that these sheep are laborin' " "under the misapprehension that they're birds. Observe their " "be'avior. Take for a start the sheeps' tendency to 'op about "  
 "the field on their 'ind legs. Now witness their attempts to "*

*"fly from tree to tree. Notice that they do not so much fly "*

*"as...plummet.", bg='royalblue', fg='ivory',*

*relief=GROOVE).pack(padx=10, pady=10)*

*Documentation for the Message widget starts on page 508.*

*42*

*CHAPTER 4 TKINTER WIDGETS*

***Figure 4.17 Message widget***

***4.1.10 Text***

*The Text widget is a versatile widget. Its primary purpose is to display text, of course, but it is capable of multiple styles and fonts, embedded images and windows, and localized event binding.*

*The Text widget may be used as a simple editor, in which case defining multiple tags and markings makes implementation easy. The widget is complex and has many options and meth- ods, so please refer to the full documentation for precise details. Some of the possible styles and embedded objects are shown in figure 4.18.*

***Figure 4.18 Text widget with several embedded objects***

*text = Text(root, height=26, width=50)*

*scroll = Scrollbar(root, command=text.yview)*

*text.configure(yscrollcommand=scroll.set)*

*text.tag\_configure('bold\_italics', font=('Verdana', 12, 'bold', 'italic')) text.tag\_configure('big', font=('Verdana', 24, 'bold')) text.tag\_configure('color', foreground='blue', font=('Tempus Sans ITC', 14)) text.tag\_configure('groove', relief=GROOVE, borderwidth=2)*

*text.tag\_bind('bite', '<1>',*

*lambda e, t=text: t.insert(END, "I'll bite your legs off!"))*

*text.insert(END, 'Something up with my banter, chaps?\n')*

*text.insert(END, 'Four hours to bury a cat?\n', 'bold\_italics')*

*text.insert(END, 'Can I call you "Frank"?\n', 'big')*

*text.insert(END, "What's happening Thursday then?\n", 'color')*

*text.insert(END, 'Did you write this symphony in the shed?\n', 'groove')*

*button = Button(text, text='I do live at 46 Horton terrace')*

*text.window\_create(END, window=button)*

*TKINTER WIDGET TOUR 43*

*44*

*CHAPTER 4 TKINTER WIDGETS*

*photo=PhotoImage(file='lumber.gif')*

*text.image\_create(END, image=photo)*

*text.insert(END, 'I dare you to click on this\n', 'bite')*

*text.pack(side=LEFT)*

*scroll.pack(side=RIGHT, fill=Y)*

*Documentation for the Text widget starts on page 528.* ***4.1.11 Canvas***

*Canvases are versatile widgets. Not only can you use them to draw complex objects, using lines, ovals, polygons and rectangles, but you can also place images and bitmaps on the canvas with great precision. In addition to these features you can place any widgets within a canvas (such as buttons, listboxes and other widgets) and bind mouse or keyboard actions to them.*

*You will see many examples in this book where Canvas widgets have been used to provide a free-form container for a variety of applications. The example shown in figure 4.19 is a some- what crude attempt to illustrate most of the available facilities.*

*One property of Canvas widgets, which can be either useful or can get in the way, is that objects are drawn on top of any objects already on the canvas. You can change the order of can- vas items later, if necessary.*

***Figure 4.19 Canvas widget***

*canvas = Canvas(root, width =400, height=400) canvas.create\_oval(10,10,100,100, fill='gray90') canvas.create\_line(105,10,200,105, stipple='@bitmaps/gray3') canvas.create\_rectangle(205,10,300,105, outline='white', fill='gray50') canvas.create\_bitmap(355, 53, bitmap='questhead')*

*xy = 10, 105, 100, 200*

*canvas.create\_arc(xy, start=0, extent=270, fill='gray60')*

*canvas.create\_arc(xy, start=270, extent=5, fill='gray70')*

*canvas.create\_arc(xy, start=275, extent=35, fill='gray80')*

*canvas.create\_arc(xy, start=310, extent=49, fill='gray90')*

*canvas.create\_polygon(205,105,285,125,166,177,210,199,205,105, fill='white') canvas.create\_text(350,150, text='text', fill='yellow', font=('verdana', 36))*

*img = PhotoImage(file='img52.gif')*

*canvas.create\_image(145,280, image=img, anchor=CENTER)*

*frm = Frame(canvas, relief=GROOVE, borderwidth=2)*

*Label(frm, text="Embedded Frame/Label").pack()*

*canvas.create\_window(285, 280, window=frm, anchor=CENTER)*

*canvas.pack()*

*Documentation for the Canvas widget starts on page 456. Documentation for the Bitmap class starts on page 452. Documentation for the PhotoImage class starts on page 512.*

***4.1.12 Scrollbar***

***Figure 4.20 Scrollbar widget***

*Scrollbar widgets can be added to any widget that supports scrolling such as Text, Canvas and Listbox widgets.*

*Associating a Scrollbar widget with another widget is as simple as adding callbacks to each widget and arranging for them to be dis- played together. Of course, there is no requirement for them to be co- located but you may end up with some unusual GUIs if you don’t! Fig- ure 4.20 shows a typical application.*

*list = Listbox(root, height=6, width=15)*

*scroll = Scrollbar(root, command=list.yview)*

*list.configure(yscrollcommand=scroll.set)*

*list.pack(side=LEFT)*

*scroll.pack(side=RIGHT, fill=Y)*

*for item in range(30):*

*list.insert(END, item)  
 Documentation for the Scrollbar widget starts on page 525.*

***4.1.13 Listbox***

*Listbox widgets display a list of values that may be chosen by the user. The default behavior of the widget is to allow the user to select a single item in the list. A simple example is shown in figure 4.21. You may add additional bindings and use the selectmode option of the widget to allow multiple-item and other properties.*

*See “Scrollbar” above, for information on adding scrolling capability to the listbox.*

*TKINTER WIDGET TOUR 45*

***Figure 4.21 List box widget***

***4.1.14 Scale***

*list = Listbox(root, width=15)*

*list.pack()*

*for item in range(10):*

*list.insert(END, item)  
 Documentation for the Listbox widget starts on page 497.*

*The Scale widget allows you to set linear values between selected lower and upper values and it displays the current value in a graphical manner. Optionally, the numeric value may be displayed.*

*The Scale widget has several options to control its appearance and behavior; otherwise it is a fairly simple widget.*

*The following example, shown in figure 4.22, is an adaptation of one of the demonstra- tions supplied with the Tcl/Tk distribution. As such, it may be useful for programmers in Tcl/ Tk to see how a conversion to Tkinter can be made.*

*46*

*CHAPTER 4 TKINTER WIDGETS*

*def setHeight(canvas, heightStr):*

*height = string.atoi(heightStr)*

*height = height + 21*

*y2 = height - 30*

*if y2 < 21:*

*y2 = 21*

*canvas.coords('poly',*

*15,20,35,20,35,y2,45,y2,25,height,5,y2,15,y2,15,20)*

***Figure 4.22 Scale widget: application***

*canvas.coords('line',*

*15,20,35,20,35,y2,45,y2,25,height,5,y2,15,y2,15,20)*

*canvas = Canvas(root, width=50, height=50, bd=0, highlightthickness=0) canvas.create\_polygon(0,0,1,1,2,2, fill='cadetblue', tags='poly') canvas.create\_line(0,0,1,1,2,2,0,0, fill='black', tags='line')*

*scale = Scale(root, orient=VERTICAL, length=284, from\_=0, to=250,*

*tickinterval=50, command=lambda h, c=canvas:setHeight(c,h))*

*scale.grid(row=0, column=0, sticky='NE')*

*canvas.grid(row=0, column=1, sticky='NWSE')*

*scale.set(100)*

*Documentation for the Scale widget starts on page 522.* ***4.2 Fonts and colors***

*The purpose of this section is to present the reader with an overview of fonts and colors as they apply to Tkinter. This will provide sufficient context to follow the examples that will be presented throughout the text.*

***4.2.1 Font descriptors***  *Those of us that have worked with X Window applications have become accustomed to the awkward and precise format of X window font descriptors. Fortunately, with release 8.0 and above of Tk, there is a solution: Tk defines font descriptors. Font descriptors are architecture independent. They allow the programmer to select a font by creating a tuple containing the family, pointsize and a string containing optional styles. The following are examples:  
   
 ('Arial', 12, 'italic')*

*('Helvetica', 10)*

*('Verdana', 8, 'medium')*

*If the font family does not contain embedded spaces, you may pass the descriptor as a sin- gle string, such as:  
   
 'Verdana 8 bold italic'*

* ***4.2.2 X Window System font descriptors***  *Of course, the older font descriptors are available if you really want to use them. Most X Win- dow fonts have a 14-field name in the form:****-foundry-family-weight-slant-setwidth-style-pixelSize-pointSize- Xresolution-Yresolution-spacing-averageWidth-registry-encoding***  *Normally, we only care about a few of the fields:  
     
  -\*-family-weight-slant-\*-\*-\*-pointSize-\*-\*-\*-\*-registry-encoding  
     
  These fields are defined as follows:  
   • family A string that identifies the basic typographic style for example, helvetica,  
     
  arial, etc.).*

*FONTS AND COLORS 47*

* *weight A string that identifies the nominal blackness of the font, according to the foundry's judgment (for example, medium, bold, etc.).*
* *slant A code string that indicates the overall posture of the typeface design used in the font—one of roman (R), italic (I) or oblique (0).*
* *pointSize An unsigned integer-string typographic metric in device-independent units which gives the body size for which the font was designed.*

*encoding A registered name that identifies the coded character set as defined by the specified registry.  
   
An example of an X font descriptor might be:* ***'-\*-verdana-medium-r-\*-\*-8-\*-\*-\*-\*-\*-\*-\*'***

* *This describes an 8-point Verdana font, medium weight and roman (upright). Although the descriptor is somewhat ugly, most programmers get used to the format quickly. With X- servers, not all fonts scale smoothly if a specific pointsize is unavailable in a font; unfortunately it is a trial-and-error process to get exactly the right combination of font and size for optimal screen appearance.****4.2.3 Colors***  *Tkinter allows you to use the color names defined by the X-server. These names are quite florid, and do not always fully describe the color: LavenderBlush1, LemonChiffon, LightSalmon, MediumOrchid3 and OldLace are just a few. Common names such as red, yellow, blue and black may also be used. The names and the corresponding RGB values are maintained in a Tk include file, so the names may be used portably on any Tkinter platform.\*  
     
  It is often easier to precisely define colors using color strings:*

*#RGB for 4-bit values (16 levels for each color) #RRGGBB for 8-bit values (256 levels for each color) #RRRRGGGGBBBB for 16-bit values (65526 levels for each color)*

*Here is an example of how one might set up part of a color definition table for an appli- cation (incomplete code):*

*# These are the color schemes for xxx and yyy front panels  
 # Panel LED off ON Active Warning COLORS = [('#545454','#656565','LawnGreen', 'ForestGreen','DarkOrange',\*

*Alarm Display Inside Chrome InsideP Chassis*

*'#ff342f','#747474','#343434','#efefef','#444444','#a0a0a0',\*

*DkChassis LtChassis VDkChassis VLtChassis Bronze*

*'#767600','#848400','#6c6c00','#909000','#7e5b41'),*

*etc.*

*# #*

*\* XwindowcolornamesarepresentinthestandardX11distributionbutarenotspecifiedbytheX11 Protocol or Xlib. It is permissible for X-server vendors to change the names or alter their intepretation. In rare cases you may find an implementation that will display different colors with Tkinter and X Window applications using the same color name.*

*48*

*CHAPTER 4 TKINTER WIDGETS*

***4.2.4 Setting application-wide default fonts and colors***

*When designing an application, you may find that the default colors, fonts and font-sizes sup- plied by the system are not appropriate for the particular layout that you have in mind. At such times you must set their values explicitly. The values could be put right in the code (you will see several examples in the book where this has been done). However, this prevents end users or system administrators from tailoring an application to their particular requirements or business standards. In this case the values should be set in an external option database. For X window programmers this is equivalent to the resource database which is usually tailored using a .Xdefaults file. In fact the format of the Tk option database is exactly like the .Xde- faults file:*

*\*font:*

*\*Label\*font:*

*\*background:*

*\*Entry\*background:*

*\*foreground:*

*\*Listbox\*foreground:*

*Verdana 10*

*Verdana 10 bold*

*Gray80*

*white*

*black*

*RoyalBlue*

*The purpose of these entries is to set the font for all widgets except Labels to Verdana 10 (regular weight) and Labels to Verdana 10 bold. Similarly we set the default colors for background and foreground, modifying Entry backgrounds and Listbox foregrounds. If we place these entries in a file called optionDB, we can apply the values using an option\_readfile call:*

*root = Tk()*

*root.option\_readfile('optionDB')*

*This call should be made early in the code to ensure that all widgets are created as intended.*

***4.3 Pmw Megawidget tour***

*Python megawidgets, Pmw, are composite widgets written entirely in Python using Tkinter widgets as base classes. They provide a convenient way to add functionality to an application without the need to write a lot of code. In particular, the ComboBox is a useful widget, along with the Entry field with several built-in validation schemes.*

*In a similar fashion to the Tkinter tour, above, the following displays show typical Pmw widget appearance and usage. The code is kept short and it illustrates some of the options avail- able for the widgets. If you need to look up a particular method or option, refer to appendix C. Each widget also has references to the corresponding section in the appendix.*

*Pmw comes with extensive documentation in HTML format. Consequently this chapter will not repeat this information here. Additionally, there is example code for all of the widgets in the demos directory in the Pmw distribution. Most of the examples shown are simplifica- tions derived from that code.*

*With the exception of the first example, the code examples have been stripped of the boil- erplate code necessary to import and initialize Tkinter. The common code which is not shown in any sequences after the first is shown in bold. The full source code for all of the displays is available online.*

*PMW MEGAWIDGET TOUR 49*

*50*

*CHAPTER 4 TKINTER WIDGETS*

***4.3.1 AboutDialog***

*The AboutDialog widget provides a convenience dialog to present version, copyright and developer information. By providing a small number of data items the dialog can be displayed with minimal code. Figure 4.23 shows a typical AboutDialog.*

***Figure 4.23 Pmw AboutDialog widget***

***from Tkinter import \****

***import Pmw***

***root = Tk()***

***root.option\_readfile('optionDB')***

***Pmw.initialise()***

*Pmw.aboutversion('1.5')*

*Pmw.aboutcopyright('Copyright Company Name 1999\nAll rights reserved')*

*Pmw.aboutcontact(*

*'For information about this application contact:\n' +*

*' Sales at Company Name\n' +*

*' Phone: (401) 555-1212\n' +*

*' email: info@company\_name.com'*

*)*

*about = Pmw.AboutDialog(root, applicationname='About Dialog')*

*root.mainloop()*

*This widget is used in the AppShell class which will be presented in “A standard appli- cation framework” on page 155 and it is used in several examples later in the book.*

*Documentation for the AboutDialog widget starts on page 542.* ***4.3.2 Balloon***

*The Balloon widget implements the now somewhat familiar balloon help motif (this is some- times called Tool Tips). The purpose of the widget is to display help information when the cursor is placed over a widget on the screen, normally after a short delay. Additionally (or alternatively) information may be displayed in a status area on the screen. The information in this area is removed after a short delay. This is illustrated in figure 4.24.*

*Although balloon help can be very helpful to novice users, it may be annoying to experts. If you provide balloon help make sure that you provide an option to turn off output to the*

*balloon and the status area, and make such choices persistent so that the user does not have to turn off the feature each time he uses the application.*

*balloon = Pmw.Balloon(root)*

*frame = Frame(root)*

*frame.pack(padx = 10, pady = 5)*

*field = Pmw.EntryField(frame, labelpos=W, label\_text='Name:')*

*field.setentry('A.N. Other')*

*field.pack(side=LEFT, padx = 10)*

*balloon.bind(field, 'Your name', 'Enter your name')*

*check = Button(frame, text='Check')*

*check.pack(side=LEFT, padx=10)*

*balloon.bind(check, 'Look up', 'Check if name is in the database')*

*frame.pack()*

*messageBar = Pmw.MessageBar(root, entry\_width=40,*

*entry\_relief=GROOVE,*

*labelpos=W, label\_text='Status:')*

*messageBar.pack(fill=X, expand=1, padx=10, pady=5)*

*balloon.configure(statuscommand = messageBar.helpmessage)*

*... After a few seconds*

*Documentation for the Balloon widget starts on page 545.* ***4.3.3 ButtonBox***

***Figure 4.24 Pmw Balloon widget***

***Figure 4.25  
 Pmw ButtonBox widget***

*The ButtonBox widget provides a convenient way to imple- ment a number of buttons and it is usually used to provide a command area within an application. The box may be laid out either horizontally or vertically and it is possible to define a default button. A simple ButtonBox is shown in figure 4.25.*

*PMW MEGAWIDGET TOUR 51*

*52*

*CHAPTER 4 TKINTER WIDGETS*

*def buttonPress(btn):*

*print 'The "%s" button was pressed' % btn*

*def defaultKey(event):*

*buttonBox.invoke()*

*buttonBox = Pmw.ButtonBox(root, labelpos='nw', label\_text='ButtonBox:') buttonBox.pack(fill=BOTH, expand=1, padx=10, pady=10)*

*buttonBox.add('OK', command = lambda b='ok': buttonPress(b)) buttonBox.add('Apply', command = lambda b='apply': buttonPress(b)) buttonBox.add('Cancel', command = lambda b='cancel': buttonPress(b))*

*buttonBox.setdefault('OK')*

*root.bind('<Return>', defaultKey)*

*root.focus\_set()*

*buttonBox.alignbuttons()*

*Documentation for the Buttonbox widget starts on page 546.* ***4.3.4 ComboBox***

*The ComboBox widget is an important widget, originally found on Macintosh and Windows interfaces and later on Motif. It allows the user to select from a list of options, which, unlike an OptionMenu, may be scrolled to accommodate large numbers of selections. The list may be displayed permanently, such as the example at the left of figure 4.26 or as a dropdown list, shown at the right of figure 4.26. Using the dropdown form results in GUIs which require much less space to implement complex interfaces.*

*choice = None*

*def choseEntry(entry):*

*print 'You chose "%s"' % entry*

*choice.configure(text=entry)*

*asply = ("The Mating of the Wersh", "Two Netlemeng of Verona", "Twelfth*

*Thing", "The Chamrent of Venice", "Thamle", "Ring Kichard the Thrid")*

*choice = Label(root, text='Choose play', relief='sunken', padx=20, pady=20)*

*choice.pack(expand=1, fill='both', padx=8, pady=8)*

*combobox = Pmw.ComboBox(root, label\_text='Play:', labelpos='wn',*

*listbox\_width=24, dropdown=0,*

*selectioncommand=choseEntry,*

*scrolledlist\_items=asply)*

*combobox.pack(fill=BOTH, expand=1, padx=8, pady=8)*

*combobox.selectitem(asply[0])*

*# ===========*

*combobox = Pmw.ComboBox(root, label\_text='Play:', labelpos='wn',*

*listbox\_width=24,* ***dropdown=1****,*

*...  
 Documentation for the ComboBox widget starts on page 549.*

***Figure 4.26 Pmw ComboBox widget***

***4.3.5 ComboBoxDialog***

*The ComboBoxDialog widget provides a convenience dialog to allow the user to select an item from a ComboBox in response to a question. It is similar to a SelectionDialog widget except that it may allow the user to type in a value in the EntryField widget or select from a perma- nently displayed list or a dropdown list. An example is shown in figure 4.27.*

*choice = None*

*def choseEntry(entry):*

*print 'You chose "%s"' % entry*

*choice.configure(text=entry)*

*plays = ("The Taming of the Shrew", "Two Gentelmen of Verona", "Twelfth*

*Night", "The Merchant of Venice", "Hamlet", "King Richard the Third")*

*dialog = Pmw.ComboBoxDialog(root, title = 'ComboBoxDialog',*

*buttons=('OK', 'Cancel'), defaultbutton='OK',*

*combobox\_labelpos=N, label\_text='Which play?',*

*scrolledlist\_items=plays, listbox\_width=22)*

*dialog.tkraise()*

*result = dialog.activate()*

*print 'You clicked on', result, dialog.get()*

*PMW MEGAWIDGET TOUR 53*

***Figure 4.27 Pmw ComboBoxDialog widget***

*Documentation for the ComboBoxDialog widget starts on page 551.* ***4.3.6 Counter***

*The Counter widget is a versatile widget which allows the user to cycle through a sequence of available values. Pmw provides integer, real, time and date counters and it is possible to define your own function to increment or decrement the displayed value. There is no limita- tion on the value that is displayed as the result of incrementing the counter, so there is no reason that the counter cannot display “eine, zwei, drei” or whatever sequence is appropriate for the application. Some examples are shown in figure 4.28.*

***Figure 4.28 Pmw Counter widget***

*def execute(self):*

*print 'Return pressed, value is', date.get()*

*date = Pmw.Counter(root, labelpos=W,*

*label\_text='Date (4-digit year):',*

*entryfield\_value=time.strftime('%d/%m/%Y',*

*time.localtime(time.time())),*

*entryfield\_command=execute,*

*entryfield\_validate={'validator' : 'date', 'format' : 'dmy'},*

*datatype = {'counter' : 'date', 'format' : 'dmy', 'yyyy' : 1})*

*real = Pmw.Counter(root, labelpos=W,*

*label\_text='Real (with comma):',*

*54*

*CHAPTER 4 TKINTER WIDGETS*

*entryfield\_value='1,5',*

*datatype={'counter' : 'real', 'separator' : ','},*

*entryfield\_validate={'validator' : 'real',*

*'min' : '-2,0', 'max' : '5,0',*

*'separator' : ','},*

*increment= .1)*

*int = Pmw.Counter(root, labelpos=W,*

*label\_text='Integer:',*

*orient=VERTICAL,*

*entry\_width=2,*

*entryfield\_value=50,*

*entryfield\_validate={'validator' : 'integer',*

*'min' : 0, 'max' : 99})*

*counters = (date, real)*

*Pmw.alignlabels(counters)*

*for counter in counters:*

*counter.pack(fill=X, expand=1, padx=10, pady=5)*

*int.pack(padx=10, pady=5)*

*Documentation for the Counter widget starts on page 553.* ***4.3.7 CounterDialog***

*The CounterDialog widget provides a convenience dialog requesting the user to select a value from a Counter widget. The counter can contain any data type that the widget is capa- ble of cycling through, such as the unlikely sequence shown in figure 4.29.*

***Figure 4.29 Pmw CounterDialog widget***

*choice = None*

*dialog = Pmw.CounterDialog(root,*

*label\_text='Enter the number of twits (2 to 8)\n', counter\_labelpos=N, entryfield\_value=2, counter\_datatype='numeric',  
 entryfield\_validate={'validator': 'numeric', 'min': 2, 'max': 8}, buttons=('OK', 'Cancel'), defaultbutton='OK',*

*title='Twit of the Year')*

*dialog.tkraise()*

*result = dialog.activate()*

*print 'You clicked on', result, dialog.get()*

*Documentation for the CounterDialog widget starts on page 556.*

*PMW MEGAWIDGET TOUR 55*

***4.3.8 Dialog***

*The Dialog widget provides a simple way to create a toplevel containing a ButtonBox and a child site area. You may populate the child site with whatever your application requires. Figure 4.30 shows an example of a Dialog.*

***Figure 4.30 Pmw Dialog widget***

*dialog = Pmw.Dialog(root, buttons=('OK', 'Apply', 'Cancel', 'Help'),*

*defaultbutton='OK', title='Simple dialog')*

*w = Label(dialog.interior(), text='Pmw Dialog\nBring out your dead!',*

*background='black', foreground='white', pady=20)*

*w.pack(expand=1, fill=BOTH, padx=4, pady=4)*

*dialog.activate()*

*Documentation for the Dialog widget starts on page 558.* ***4.3.9 EntryField***

*The EntryField widget is an Entry widget with associated validation methods. The built- in validation provides validators for integer, hexadecimal, alphabetic, alphanumeric, real, time and date data formats. Some of the controls that may be placed on the validation include checking conformity with the selected data format and checking that entered data is between minimum and maximum limits. You may also define your own validators. A few examples are shown in figure 4.31.*

***Figure 4.31 Pmw EntryField widget***

*noval = Pmw.EntryField(root, labelpos=W, label\_text='No validation',*

*validate = None)*

*real = Pmw.EntryField(root, labelpos=W,value = '98.4',*

*label\_text = 'Real (96.0 to 107.0):',*

*validate = {'validator' : 'real',*

*'min' : 96, 'max' : 107, 'minstrict' : 0})*

*56*

*CHAPTER 4*

*TKINTER WIDGETS*

*int = Pmw.EntryField(root, labelpos=W, label\_text = 'Integer (5 to 42):',*

*validate = {'validator' : 'numeric',*

*'min' : 5, 'max' : 42, 'minstrict' : 0},*

*value = '12')*

*date = Pmw.EntryField(root, labelpos=W,label\_text = 'Date (in 2000):',*

*value = '2000/1/1', validate = {'validator' : 'date',*

*'min' : '2000/1/1', 'max' : '2000/12/31',*

*'minstrict' : 0, 'maxstrict' : 0,*

*'format' : 'ymd'})*

*widgets = (noval, real, int, date)*

*for widget in widgets:*

*widget.pack(fill=X, expand=1, padx=10, pady=5)*

*Pmw.alignlabels(widgets)*

*real.component('entry').focus\_set()  
 Documentation for the EntryField widget starts on page 559.*

***4.3.10 Group***

*The Group widget provides a convenient way to place a labeled frame around a group of wid- gets. The label can be any reasonable widget such as a Label but it can also be an Entry- Field, RadioButton or CheckButton depending on the application requirements. It is also possible to use the widget as a graphic frame with no label. These examples are shown in figure 4.32.*

***Figure 4.32 Pmw Group widget***

*w = Pmw.Group(root, tag\_text='place label here')*

*w.pack(fill=BOTH, expand=1, padx=6, pady=6)*

*cw = Label(w.interior(), text='A group with a\nsimple Label tag')*

*cw.pack(padx=2, pady=2, expand=1, fill=BOTH)*

*w = Pmw.Group(root, tag\_pyclass=None)*

*w.pack(fill=BOTH, expand=1, padx=6, pady=6)*

*cw = Label(w.interior(), text='A group\nwithout a tag')*

*cw.pack(padx=2, pady=2, expand=1, fill=BOTH)*

*w = Pmw.Group(root, tag\_pyclass=Checkbutton,*

*tag\_text='checkbutton', tag\_foreground='blue')*

*PMW MEGAWIDGET TOUR 57*

*58*

*CHAPTER 4 TKINTER WIDGETS*

*w.pack(fill=BOTH, expand=1, padx=6, pady=6)*

*cw = Frame(w.interior(),width=150,height=20)*

*cw.pack(padx=2, pady=2, expand=1, fill=BOTH)*

*Documentation for the Group widget starts on page 564.* ***4.3.11 LabeledWidget***

*The LabeledWidget widget is a convenience container which labels a widget or collection of widgets. Options are provided to control the placement of the label and control the appear- ance of the graphic border. The child site can be populated with any combination of widgets. The example shown in figure 4.33 uses the widget as a frame which requires less code than using individual components.*

***Figure 4.33 Pmw LabeledWidget widget***

*frame = Frame(root, background = 'gray80')*

*frame.pack(fill=BOTH, expand=1)*

*lw = Pmw.LabeledWidget(frame, labelpos='n',*

*label\_text='Sunset on Cat Island')*

*lw.component('hull').configure(relief=SUNKEN, borderwidth=3)*

*lw.pack(padx=10, pady=10)*

*img = PhotoImage(file='chairs.gif')*

*cw = Button(lw.interior(), background='yellow', image=img)*

*cw.pack(padx=10, pady=10, expand=1, fill=BOTH)*

*Documentation for the LabeledWidget widget starts on page 565.*

***4.3.12 MenuBar***

*The MenuBar widget is a manager widget which provides methods to add menu buttons and menus to the menu bar and to add menu items to the menus. One important convenience is that it is easy to add balloon help to the menus and menu items. Almost all of the menu options available with Tkinter Menu widgets (see “Menu” on page 39) are available through the Pmw MenuBar. Figure 4.34 illustrates a similar menu to the one shown in figure 4.13 using discrete Tkinter widgets.*

***Figure 4.34 Pmw MenuBar widget***

*balloon = Pmw.Balloon(root)*

*menuBar = Pmw.MenuBar(root, hull\_relief=RAISED,hull\_borderwidth=1,*

*balloon=balloon)*

*menuBar.pack(fill=X)*

*menuBar.addmenu('Buttons', 'Simple Commands')*

*menuBar.addmenuitem('Buttons', 'command', 'Close this window',*

*font=('StingerLight', 14), label='Close')*

*menuBar.addmenuitem('Buttons', 'command',*

*bitmap="@bitmaps/RotateLeft", foreground='yellow')*

*menuBar.addmenuitem('Buttons', 'separator')*

*menuBar.addmenuitem('Buttons', 'command',*

*'Exit the application', label='Exit')*

*menuBar.addmenu('Cascade', 'Cascading Menus')*

*menuBar.addmenu('Checkbutton', 'Checkbutton Menus')*

*menuBar.addmenu('Radiobutton', 'Radiobutton Menus')*

*Documentation for the MenuBar widget starts on page 572.* ***4.3.13 MessageBar***

*The MessageBar widget is used to implement a status area for an application. Messages in several discrete categories may be displayed. Each message is displayed for a period of time which is determined by its category. Additionally, each category is assigned a priority so the message with the highest priority is displayed first. It is also possible to specify the number of times that the bell should be rung on receipt of each message category. Figure 4.35 shows how a system error would appear.*

*messagebar = box = None*

*def selectionCommand():*

*sels = box.getcurselection()*

*if len(sels) > 0:*

*PMW MEGAWIDGET TOUR 59*

*60*

*CHAPTER 4*

*TKINTER WIDGETS*

*messagetype = sels[0]*

*if messagetype == 'state':*

*messagebar.message('state', 'Change of state message')*

*else:*

*text = messages[messagetype]*

*messagebar.message(messagetype, text)*

*messages = { 'help' : 'Save current file', 'userevent' : 'Saving file "foo"',*

*'busy' : 'Busy deleting all files from file system ...',*

*'systemevent': 'File "foo" saved',*

*'usererror' : 'Invalid file name "foo/bar"',*

*'systemerror': 'Failed to save file: file system full',*

*}*

*messagebar = Pmw.MessageBar(root, entry\_width=40, entry\_relief=GROOVE, labelpos=W, label\_text='Status:')*

*messagebar.pack(side=BOTTOM, fill=X, expand=1, padx=10, pady=10)*

*box = Pmw.ScrolledListBox(root,listbox\_selectmode=SINGLE,*

*items=('state', 'help', 'userevent', 'systemevent',*

*'usererror', 'systemerror', 'busy',),*

*label\_text='Message type', labelpos=N,*

*selectioncommand=selectionCommand)*

*box.pack(fill=BOTH, expand=1, padx=10, pady=10)*

*Documentation for the MessageBar widget starts on page 574.*

***Figure 4.35 Pmw MessageBar widget***

***4.3.14 MessageDialog***

*The MessageDialog widget is a convenience dialog which displays a single message, which may be broken into multiple lines, and a number of buttons in a ButtonBox. It is useful for creating simple dialogs “on-the-fly.” Figure 4.36 shows an example.*

***Figure 4.36 Pmw MessageDialog widget***

*dialog = Pmw.MessageDialog(root, title = 'Simple Dialog',*

*defaultbutton = 0,*

*buttons = ('OK', 'Apply', 'Cancel', 'Help'), message\_text = 'This dialog box was constructed on demand')*

*dialog.iconname('Simple message dialog')*

*result = dialog.activate()*

*print 'You selected', result*

*Documentation for the MessageDialog widget starts on page 576.* ***4.3.15 NoteBookR***

*The NoteBookR widget implements the popular property sheet motif. Methods allow a num- ber of pages or panes to be created. Any content may then be added to the panels. The user selects a panel by clicking on the tab at its top. Alternatively panels may be raised or lowered through instance methods. An example is shown in figure 4.37.*

*nb = Pmw.NoteBookR(root)*

*nb.add('p1', label='Page 1')*

*nb.add('p2', label='Page 2')*

*nb.add('p3', label='Page 3')*

*p1 = nb.page('p1').interior()*

*p2 = nb.page('p2').interior()*

*p3 = nb.page('p3').interior()*

*nb.pack(padx=5, pady=5, fill=BOTH, expand=1)*

*Button(p1, text='This is text on page 1', fg='blue').pack(pady=40)*

*c = Canvas(p2, bg='gray30')*

*w = c.winfo\_reqwidth()*

*h = c.winfo\_reqheight()*

*c.create\_oval(10,10,w-10,h-10,fill='DeepSkyBlue1')*

*c.create\_text(w/2,h/2,text='This is text on a canvas', fill='white',*

*font=('Verdana', 14, 'bold'))*

*c.pack(fill=BOTH, expand=1)*

*PMW MEGAWIDGET TOUR 61*

***Figure 4.37 Pmw NoteBookR widget***

*Documentation for the NotebookR widget starts on page 580.* ***4.3.16 NoteBookS***

*The NoteBookS widget implements an alternative style of NoteBook. NoteBookS provides additional options to control the color, dimensions and appearance of the tabs. Otherwise it is quite similar to NoteBookR. Figure 4.38 illustrates a similar layout using NotebookS.*

*62*

*CHAPTER 4 TKINTER WIDGETS*

*nb = Pmw.NoteBookS(root)*

*nb.addPage('Page 1')*

*nb.addPage('Page 2')*

*nb.addPage('Page 3')*

***Figure 4.38 Pmw NoteBookS widget***

*f1 = nb.getPage('Page 1')*

*f2 = nb.getPage('Page 2')*

*f3 = nb.getPage('Page 3')*

*nb.pack(pady=10, padx=10, fill=BOTH, expand=1)*

*Button(f1, text='This is text on page 1', fg='blue').pack(pady=40)*

*c = Canvas(f2, bg='gray30')*

*w = c.winfo\_reqwidth()*

*h = c.winfo\_reqheight()*

*c.create\_oval(10,10,w-10,h-10,fill='DeepSkyBlue1')*

*c.create\_text(w/2,h/2,text='This is text on a canvas', fill='white',*

*font=('Verdana', 14, 'bold'))*

*c.pack(fill=BOTH, expand=1)*

*Documentation for the NotebookS widget starts on page 582.* ***4.3.17 NoteBook***

*Release 0.8.3 of Pmw replaces NoteBookR and NoteBookS with Notebook. While it is quite similar to the previous notebooks, there are some small changes. In fact, you will have to make changes to your code to use NoteBook with existing code. However, the changes are minor and the new form may be a little easier to use. Figure 4.39 illustrates the new widget.*

*from Tkinter import \**

*import Pmw*

*root = Tk()*

*root.option\_readfile('optionDB')*

*root.title('Notebook')*

*Pmw.initialise()*

*nb = Pmw.NoteBook(root)*

*p1 = nb.add('Page 1')*

*PMW MEGAWIDGET TOUR*

*63*

***Figure 4.39 Pmw NoteBook widget (version 0.8.3)***

*p2 = nb.add('Page 2')*

*p3 = nb.add('Page 3')*

*nb.pack(padx=5, pady=5, fill=BOTH, expand=1)*

*Button(p1, text='This is text on page 1', fg='blue').pack(pady=40)*

*c = Canvas(p2, bg='gray30')*

*w = c.winfo\_reqwidth()*

*h = c.winfo\_reqheight()*

*c.create\_oval(10,10,w-10,h-10,fill='DeepSkyBlue1')*

*c.create\_text(w/2,h/2,text='This is text on a canvas', fill='white',*

*font=('Verdana', 14, 'bold'))*

*c.pack(fill=BOTH, expand=1)*

*nb.setnaturalpagesize()*

*root.mainloop()*

*Documentation for the Notebook widget starts on page 578.* ***4.3.18 OptionMenu***

*The OptionMenu widget implements a classic popup menu motif familiar to Motif program- mers. However, the appearance of the associated popup is a little different, as shown in figure 4.40. OptionMenus should be used to select limited items of data. If you populate the widget with large numbers of data the popup may not fit on the screen and the widget does not scroll.*

***Figure 4.40 Pmw OptionMenu widget***

*var = StringVar()*

*var.set('Quantity Surveyor')*

*opt\_menu = Pmw.OptionMenu(root, labelpos=W,*

*label\_text='Choose profession:', menubutton\_textvariable=var,*

*64*

*CHAPTER 4 TKINTER WIDGETS*

*items=('Stockbroker', 'Quantity Surveyor', 'Church Warden', 'BRM'),*

*menubutton\_width=16)*

*opt\_menu.pack(anchor=W, padx=20, pady=30)*

*Documentation for the OptionMenu widget starts on page 584.* ***4.3.19 PanedWidget***

*The PanedWidget widget creates a manager containing multiple frames. Each frame is a con- tainer for other widgets and may be resized by dragging on its handle or separator line. The area within each pane is managed independently, so a single pane may be grown or shrunk to modify the layout of its children. Figure 4.41 shows an example.*

***Figure 4.41 Pmw PanedWidget widget***

*pane = Pmw.PanedWidget(root, hull\_width=400, hull\_height=300)*

*pane.add('top', min=100)*

*pane.add('bottom', min=100)*

*topPane = Pmw.PanedWidget(pane.pane('top'), orient=HORIZONTAL)*

*for num in range(4):*

*if num == 1:*

*name = 'Fixed\nSize'*

*topPane.add(name, min=.2, max=.2)*

*else:*

*name = 'Pane\n' + str(num)*

*topPane.add(name, min=.1, size=.25)*

*button = Button(topPane.pane(name), text=name)*

*button.pack(expand=1)*

*topPane.pack(expand=1, fill=BOTH) pane.pack(expand=1, fill=BOTH)  
 Documentation for the PanedWidget widget starts on page 586.*

*PMW MEGAWIDGET TOUR 65*

***4.3.20 PromptDialog***

*The PromptDialog widget is a convenience dialog which displays a single EntryField and a number of buttons in a ButtonBox. It is useful for creating a simple dialog on-the-fly. The example shown in figure 4.42 collects a password from a user.*

***Figure 4.42 Pmw PromptDialog widget***

*dialog = Pmw.PromptDialog(root, title='Password', label\_text='Password:',*

*entryfield\_labelpos=N, entry\_show='\*', defaultbutton=0,*

*buttons=('OK', 'Cancel'))*

*result = dialog.activate()*

*print 'You selected', result*

*Documentation for the PromptDialog widget starts on page 587.* ***4.3.21 RadioSelect***

*The RadioSelect widget implements an alternative to the Tkinter RadioButton widget. RadioSelect creates a manager that contains a number of buttons. The widget may be con- figured to operate either in single-selection mode where only one button at a time may be activated, or multiple selection mode where any number of buttons may be selected. This is illustrated in figure 4.43.*

***Figure 4.43 Pmw RadioSelect widget***

*horiz = Pmw.RadioSelect(root, labelpos=W, label\_text=HORIZONTAL,*

*frame\_borderwidth=2, frame\_relief=RIDGE)*

*horiz.pack(fill=X, padx=10, pady=10)*

*for text in ('Passion fruit', 'Loganberries', 'Mangoes in syrup',*

*'Oranges', 'Apples', 'Grapefruit'):*

*horiz.add(text)*

*66*

*CHAPTER 4 TKINTER WIDGETS*

*horiz.invoke('Mangoes in syrup')*

*multiple = Pmw.RadioSelect(root, labelpos=W, label\_text='Multiple\nselection', frame\_borderwidth=2, frame\_relief=RIDGE, selectmode=MULTIPLE)*

*multiple.pack(fill=X, padx=10)*

*for text in ('Doug', 'Dinsdale', "Stig O'Tracy", 'Vince', 'Gloria Pules'):*

*multiple.add(text)*

*multiple.invoke('Dinsdale')  
 Documentation for the RadioSelect widget starts on page 589.*

***4.3.22 ScrolledCanvas***

*The ScrolledCanvas widget is a convenience widget providing a Canvas widget with asso- ciated horizontal and vertical scrollbars. An example is shown in figure 4.44.*

***Figure 4.44 Pmw ScrolledCanvas widget***

*sc = Pmw.ScrolledCanvas(root, borderframe=1, labelpos=N,*

*label\_text='ScrolledCanvas', usehullsize=1,*

*for i in range(20):*

*x = -10 + 3\*i*

*y = -10*

*for j in range(10):*

*hull\_width=400,hull\_height=300)*

*sc.create\_rectangle('%dc'%x,'%dc'%y,'%dc'%(x+2),'%dc'%(y+2),*

*fill='cadetblue', outline='black')*

*sc.create\_text('%dc'%(x+1),'%dc'%(y+1),text='%d,%d'%(i,j),*

*anchor=CENTER, fill='white')*

*y=y+3*

*sc.pack()*

*sc.resizescrollregion()*

*Documentation for the ScrolledCanvas widget starts on page 592.*

*PMW MEGAWIDGET TOUR 67*

***4.3.23 ScrolledField***

*68*

*CHAPTER 4 TKINTER WIDGETS*

*The ScrolledField widget provides a labeled EntryField widget with bindings to allow the user to scroll through data which is too great to be displayed within the available space. This widget should be reserved for very special uses, since it contravenes many of the com- monly considered human factors for GUI elements. Figure 4.45 shows the effect of scrolling the field using the keyboard arrow keys.*

***Figure 4.45 Pmw ScrolledField widget***

*lines = (*

*"Mount Everest. Forbidding, aloof, terrifying. This year, this",*

*"remote Himalayan mountain, this mystical temple, surrounded by the",*

*"most difficult terrain in the world, repulsed yet another attempt to",*

*"conquer it. (Picture changes to wind-swept, snowy tents and people)",*

*"This time, by the International Hairdresser's Expedition. In such",*

*"freezing, adverse conditions, man comes very close to breaking",*

*"point. What was the real cause of the disharmony which destroyed",*

*"their chances at success?")*

*global index*

*field = index = None*

*def execute():*

*global index*

*field.configure(text=lines[index % len(lines)])*

*index = index + 1*

*field = Pmw.ScrolledField(root, entry\_width=30,*

*entry\_relief=GROOVE, labelpos=N,*

*label\_text='Scroll the field using the\nmiddle mouse button')*

*field.pack(fill=X, expand=1, padx=10, pady=10)*

*button = Button(root, text='Change field', command=execute)*

*button.pack(padx=10, pady=10)*

*index = 0*

*execute()*

*Documentation for the ScrolledField widget starts on page 594.*

***4.3.24 ScrolledFrame***

*The ScrolledFrame widget is a convenience widget providing a Frame widget with associ- ated horizontal and vertical scrollbars. An example is shown in figure 4.46.*

***Figure 4.46 Pmw ScrolledFrame widget***

*global row, col*

*row = col = 0*

*sf = frame = None*

*def addButton():*

*global row, col*

*button = Button(frame, text = '(%d,%d)' % (col, row))*

*button.grid(row=row, col=col, sticky='nsew')*

*frame.grid\_rowconfigure(row, weight=1)*

*frame.grid\_columnconfigure(col, weight=1)*

*sf.reposition()*

*if col == row:*

*col = 0*

*row = row + 1*

*else:*

*col = col + 1*

*sf = Pmw.ScrolledFrame(root, labelpos=N, label\_text='ScrolledFrame',*

*usehullsize=1, hull\_width=400, hull\_height=220)*

*sf.pack(padx=5, pady=3, fill='both', expand=1)*

*frame = sf.interior()*

*for i in range(250):*

*addButton()*

*Documentation for the ScrolledFrame widget starts on page 595.*

*PMW MEGAWIDGET TOUR 69*

***4.3.25 ScrolledListbox***

*The ScrolledListbox widget is a convenience widget providing a ListBox widget with associated horizontal and vertical scrollbars. Figure 4.47 shows a typical ScrolledListbox.*

***Figure 4.47 Pmw ScrolledListbox widget***

*box = None*

*def selectionCommand():*

*sels = box.getcurselection()*

*if len(sels) == 0:*

*print 'No selection'*

*else:*

*print 'Selection:', sels[0]*

*box = Pmw.ScrolledListBox(root, listbox\_selectmode=SINGLE,*

*items=('John Cleese', 'Eric Idle', 'Graham Chapman',*

*'Terry Jones', 'Michael Palin', 'Terry Gilliam'),*

*labelpos=NW, label\_text='Cast Members',*

*listbox\_height=5, vscrollmode='static',*

*selectioncommand=selectionCommand,*

*dblclickcommand=selectionCommand,*

*usehullsize=1, hull\_width=200, hull\_height=200,)*

*box.pack(fill=BOTH, expand=1, padx=5, pady=5)*

*Documentation for the ScrolledListbox widget starts on page 598.* ***4.3.26 ScrolledText***

*The ScrolledText widget is a convenience widget providing a Text widget with associated horizontal and vertical scrollbars, as shown in figure 4.48.*

*st = Pmw.ScrolledText(root, borderframe=1, labelpos=N,*

*label\_text='Blackmail', usehullsize=1,*

*hull\_width=400, hull\_height=300,*

*text\_padx=10, text\_pady=10,*

*text\_wrap='none')*

*st.importfile('blackmail.txt')*

*st.pack(fill=BOTH, expand=1, padx=5, pady=5) Documentation for the ScrolledText widget starts on page 600.*

*70*

*CHAPTER 4*

*TKINTER WIDGETS*

***Figure 4.48 Pmw ScrolledText widget***

***4.3.27 SelectionDialog***

*The SelectionDialog widget provides a convenience dialog to allow the user to select an item from a ScrolledList in response to a question. It is similar to a ComboBoxDialog except that there is no provision for the user to type in a value. Figure 4.49 shows an example.*

***Figure 4.49 Pmw SelectionDialog widget***

*dialog = None*

*def execute(result):*

*sels = dialog.getcurselection()*

*if len(sels) == 0:*

*print 'You clicked on', result, '(no selection)'*

*PMW MEGAWIDGET TOUR 71*

*else:*

*print 'You clicked on', result, sels[0]*

*dialog.deactivate(result)*

*dialog = Pmw.SelectionDialog(root, title='String',*

*buttons=('OK', 'Cancel'), defaultbutton='OK',*

*scrolledlist\_labelpos=N, label\_text='Who sells string?',*

*scrolledlist\_items=('Mousebat', 'Follicle', 'Goosecreature',*

*'Mr. Simpson', 'Ampersand', 'Spong', 'Wapcaplet',*

*'Looseliver', 'Vendetta', 'Prang'),*

*command=execute)*

*dialog.activate()  
 Documentation for the SelectionDialog widget starts on page 603.*

***4.3.28 TextDialog***

*The TextDialog widget provides a convenience dialog used to display multi-line text to the user. It may also be used as a simple text editor. It is shown in figure 4.50.*

***Figure 4.50 Pmw TextDialog widget***

*sketch = """Doctor: Mr. Bertenshaw?*

*Mr. B: Me, Doctor.*

*# ------Lines removed----------*

*Jane, you Trent, you Trillo...me doctor!"""*

*dialog = Pmw.TextDialog(root, scrolledtext\_labelpos='n',*

*72*

*CHAPTER 4 TKINTER WIDGETS*

*title='Sketch',*

*defaultbutton=0,*

*label\_text='The Hospital')*

*dialog.insert(END, sketch)*

*dialog.configure(text\_state='disabled')*

*dialog.activate()*

*dialog.tkraise()*

*Documentation for the TextDialog widget starts on page 605.* ***4.3.29 TimeCounter***

*The TimeCounter widget implements a device to set hours, minutes and seconds using up and down arrows. The widget may be configured to autorepeat so that holding down a button will slew the value displayed in the widget. Figure 4.51 shows the widget’s appearance.*

***Figure 4.51 Pmw TimeCounter widget***

*time = Pmw.TimeCounter(root, labelpos=W, label\_text='HH:MM:SS',*

*min='00:00:00', max='23:59:59')*

*time.pack(padx=10, pady=5)  
 Documentation for the TimeCounter widget starts on page 607.*

***4.4 Creating new megawidgets***

*In addition to supplying useful widgets, Pmw provides a simple mechanism to allow you to develop new megawidgets. The documentation supplied with Pmw describes the process of coding a megawidget. This description is an adaptation of that material.*

***4.4.1 Description of the megawidget***

*This widget will implement a simple gauge which tracks an integer value supplied by a Scale widget, which selects a number from a range. The gauge indicates the setting as a percentage of the range. The completed megawidget will look like the one shown in figure 4.52.*

*The scale widget will be a component of the megawidget since the range may be set by the programmer; the size and color of the gauge may similarly be changed, as appropriate for the application, so we make this a component, too.*

***Figure 4.52 Gauge widget***

*CREATING NEW MEGAWIDGETS 73*

***4.4.2 Options***

***4.4.3***

*In addition to the options for the scale and gauge components, we will need to define some options for the megawidget. First, we define min and max to allow the programmer the range supported by the widget. Secondly, we define fill and size to control the color and size of the gauge. Lastly, we define value to allow us to set the initial value of the megawidget.*

***Creating the megawidget class***

*Pmw megawidgets inherit from either Pmw.MegaWidget, Pmw.MegaToplevel or Pmw.Dia- log. The gauge widget is intended to be used within other code widgets so it inherits from Pmw.MegaWidget. Here is the code for the megawidget.*

***pmw\_megawindget.py***

*from Tkinter import \**

*import Pmw*

*class Gauge(Pmw.MegaWidget):*

*def \_\_init\_\_(self, parent=None, \*\*kw):*

*# Define the options for the megawidget*

*optiondefs = (*

*('min', 0,*

*('max', 100,*

*('fill', 'red',*

*('size', 30,*

*('value', 0,*

*('showvalue', 1,*

*Pmw.INITOPT),*

*Pmw.INITOPT),*

*None),*

*Pmw.INITOPT),*

*None),*

*None),*

*1*

*2 3*

*4*

*5*

*)*

*self.defineoptions(kw, optiondefs)*

*# Initialize the base class*

*Pmw.MegaWidget.\_\_init\_\_(self, parent)*

*interior = self.interior()*

*# Create the gauge component*

*self.gauge = self.createcomponent('gauge',*

*(), None,*

*Frame, (interior,),*

*borderwidth=0)*

*self.canvas = Canvas(self.gauge,*

*width=self['size'], height=self['size'],*

*background=interior.cget('background'))*

*self.canvas.pack(side=TOP, expand=1, fill=BOTH)*

*self.gauge.grid()*

*# Create the scale component*

*self.scale = self.createcomponent('scale',*

*(), None,*

*Scale, (interior,),*

*command=self.\_setGauge,*

*length=200,*

*74*

*CHAPTER 4*

*TKINTER WIDGETS*

*from\_ = self['min'],*

*to = self['max'],*

*showvalue=self['showvalue'])*

*self.scale.grid()*

*value=self['value']*

*if value is not None:*

*self.scale.set(value)*

*# Check keywords and initialize options*

*self.initialiseoptions(Gauge)*

*def \_setGauge(self, value):*

*self.canvas.delete('gauge')*

*ival = self.scale.get()*

*ticks = self['max'] - self['min']*

*arc = (360.0/ticks) \* ival*

*xy = 3,3,self['size'],self['size']*

*start = 90-arc*

*if start < 0:*

*5*

*6*

*start = 360 + start*

*self.canvas.create\_arc(xy, start=start, extent=arc-.001,*

*fill=self['fill'], tags=('gauge',))*

*Pmw.forwardmethods(Gauge, Scale, 'scale')*

*root = Tk()*

*root.option\_readfile('optionDB')*

*root.title('Gauge')*

*Pmw.initialise()*

*g1 = Gauge(root, fill='red', value=56, min=0, max=255)*

*g1.pack(side=LEFT, padx=1, pady=10)*

*g2 = Gauge(root, fill='green', value=60, min=0, max=255)*

*g2.pack(side=LEFT, padx=1, pady=10)*

*g3 = Gauge(root, fill='blue', value=36, min=0, max=255)*

*g3.pack(side=LEFT, padx=1, pady=10)*

*root.mainloop()*

*Code comments*

*7*

* *1 Options for the megawidget are specified by a three-element sequence of the option name, default value and a final argument. The final argument can be either a callback function, Pmw.INITOPT or None. If it is Pmw.INITOPT then the option may only be provided as an initialization option and it cannot be set by calling configure. Calling self.defineop- tions includes keyword arguments passed in the widget’s constructor. These values may over- ride any default values.*
* *2 Having set the options we call the constructor of the base class, passing the parent widget as the single argument.*
* *3 By convention, Pmw defines an interior attribute which is the container for components.*

*CREATING NEW MEGAWIDGETS 75*

*4*

*5 6*

*7*

*We then create the gauge’s indicator, which is going to be drawn on a canvas contained in a frame. The createcomponent method has five standard arguments (name, aliases, group, class and arguments to the constructor) followed by any number of keyword argu- ments.*

*Then, we construct the scale component in a similar manner.*

*Having completed the constructor, we first call initialiseoptions to check that all of the keyword arguments we supplied have been used. It then calls any option callbacks that have been defined.*

*Once the megawidget’s class has been defined we call the Pmw.forwardmethods method to direct any method calls from other widgets to the scale component.*

***Figure 4.53 Using the gauge megawid- get as a color mixer***

*Figure 4.53 illustrates a possible application of the gauge megawidget as a color mixer. The widget may be reconfigured to show or hide the current value of each slider. It is an easy task to add more options to the widget.*