gtk.TreeView

Es un widget de tipo contenedor.
Que sirve para mostrar un modelo de datos de tipo árbol (gtk.TreeStore) o
Un modelo de datos de tipo lista (gtk.ListStore).

Antes de explicar, vemos la descripción general de la clase.

Funciones de Clase:

```
class gtk.TreeView(gtk.Container):
    gtk.TreeView(model=None)
    def get_model()
    def set_model(model=None)
    def get_selection()
    def get_hadjustment()
    def set_hadjustment(adjustment)
    def get_vadjustment()
    def set_vadjustment(adjustment)
    def get_headers_visible()
    def set_headers_visible(headers_visible)
    def columns_autosize()
    def set_headers_clickable(active)
    def set_rules_hint(setting)
    def get_rules_hint()
    def append_column(column)
    def remove_column(column)
    def insert_column(column, position)
    def insert_column_with_attributes(position, title, cell, ...)
    def insert_column_with_data_func(position, title, cell, func, data=None)
    def get_column(n)
    def get_columns()
    def move_column_after(column, base_column)
    def set_expander_column(column)
    def get_expander_column()
    def set_column_drag_function(func, user_data)
    def scroll_to_point(tree_x, tree_y)
    def scroll_to_cell(path, column, use_align, row_align, col_align)
    def row_activated(path, column)
    def expand_all()
    def collapse_all()
    def expand_to_path(path)
    def expand_row(path, open_all)
    def collapse_row(path)
    def map_expanded_rows(func, data)
    def row_expanded(path)
    def set_reorderable(reorderable)
```

```
def get_reorderable()
def set_cursor(path, focus_column=None, start_editing=False)
def set_cursor_on_cell(path, focus_column=None, focus_cell=None, start_editing=False)
def get_cursor()
def get_bin_window()
def get_path_at_pos(x, y)
def get_cell_area(path, column)
def get_background_area(path, column)
def get_visible_rect()
def widget_to_tree_coords(wx, wy)
def tree_to_widget_coords(tx, ty)
def enable_model_drag_source(start_button_mask, targets, actions)
def enable_model_drag_dest(targets, actions)
def unset_rows_drag_source()
def unset_rows_drag_dest()
def set_drag_dest_row(path, pos)
def get_drag_dest_row()
def get_dest_row_at_pos(x, y)
def create_row_drag_icon(path)
def set_enable_search(enable_search)
def get_enable_search()
def get_search_column()
def set_search_column(column)
def set_search_equal_func(func=None, user_data=None)
def get_fixed_height_mode()
def set_fixed_height_mode(enable)
def get_hover_selection()
def set_hover_selection(hover)
def get_hover_expand()
def set_hover_expand(expand)
def set_row_separator_func(func=None, user_data=None)
def get_visible_range()
def get_headers_clickable()
def get_search_entry()
def set_search_entry(entry=None)
def set_search_position_func(func, data=None)
def set_rubber_banding(enable)
def get_rubber_banding()
def get_grid_lines()
def set_grid_lines(grid_lines)
def get_enable_tree_lines()
def set_enable_tree_lines(enabled)
def convert_widget_to_bin_window_coords(widget_x, widget_y)
def convert_widget_to_tree_coords(widget_x, widget_y)
def convert_tree_to_widget_coords(tree_x, tree_y)
def convert_tree_to_bin_window_coords(tree_x, tree_y)
def convert_bin_window_to_widget_coords(window_x, window_y)
def convert_bin_window_to_tree_coords(window_x, window_y)
def get_level_indentation()
```

```
def set_level_indentation(indentation)
def get_show_expanders()
def set_show_expanders(enabled)
def get_tooltip_column()
def set_tooltip_column(column)
def is_rubber_banding_active()
def set_tooltip_cell(tooltip, path, column, cell)
def set_tooltip_row(tooltip, path)
```

Clases Ancestras:

```
+-- gobject.GObject
 +-- gtk.Object
  +-- gtk.Widget
   +-- gtk.Container
    +-- gtk.TreeView
```

Interfaces Implementadas:

gtk.Buildable

Propiedades de TreeViewColumn:

(Las de sus ancestros):
gtk.Object Properties
gtk.Widget Properties
gtk.Container Properties

(Las propias):

"enable-grid-lines" Lectura-Escritura
Dibujar o no, una linea entre cada fila.

"enable-search" Lectura-Escritura
Permite la busqueda interactiva entre columnas. Default value: True

"enable-tree-lines" Lectura-Escritura
Dibujar lineas entre nodos (las ramas del arbol)

"expander-column" Lectura-Escritura
Establece la columna que contendrá el ícono expansor.

"fixed-height-mode" Lectura-Escritura
Si es True, asumirá que todas las filas tienen la misma altura lo cual acelera la pantalla.
Valor por defecto: falso. Disponible en GTK + 2.4 y versiones superiores.

"hadjustment" Lectura-Escritura
El ajuste horizontal del widget.

"headers-clickable" Escritura
Si es True, los encabezados de columna responden a los eventos de click. Valor por defecto: False

"headers-visible" Lectura-Escritura
Si es True, muestra los botones de encabezado de columna. Valor por defecto: cierto

"hover-expand" Lectura-Escritura
Si es True, expande o contrae las filas si se mueve el puntero del mouse sobre ellos.
Este modo está pensado principalmente para treeviews en ventanas emergentes, por ejemplo, en gtk.ComboBox o gtk.EntryCompletion. Valor por defecto: falso. Disponible en GTK + 2.6 y superior.

"hover-selection" Lectura-Escritura
Si es True, la fila seleccionada sigue al puntero del mouse. Dicho de otro modo, se mantiene la seleccion en la fila que está debajo del mouse. Actualmente, esto sólo funciona para los modos de selección y gtk.SELECTION_SINGLE gtk.SELECTION_BROWSE. Este modo está pensado principalmente para treeviews en ventanas emergentes, por ejemplo, en gtk.ComboBox o gtk.EntryCompletion. Valor por defecto: falso. Disponible en GTK + 2.6 y superior.

"level-indentation" Lectura-Escritura
Sangría para cada nivel.

"model" Lectura-Escritura
El modelo de la vista en árbol.

"reorderable" Lectura-Escritura
Si es True, la vista es reorderable. Valor por defecto: falso.

"rubber-banding" Lectura-Escritura
Si es True habilita la selección de varios elementos cuando se arrastra el puntero del ratón.

"rules-hint" Lectura-Escritura
Si es True, dibujar las filas de colores alternados. Valor por defecto: falso.

"search-column" Lectura-Escritura
La columna del modelo para buscar cuando se busca a través de código.
Valores posibles:> = -1. Valor por defecto: -1

"show-expanders" Lectura-Escritura
Hacer los expansores si los tiene.

"vadjustment" Lectura-Escritura
A juste vertical para el widget.

Propiedades de Estilo:

"allow-rules" Lectura

Si es True, permiten dibujar filas alternas de color.

"even-row-color" Lectura
el gtk.gdk.Color a utilizar para filas pares. Disponible en GTK + 2.2 y superiores.

"expander-size" Lectura
el tamaño de la flecha de ampliación. Valores posibles:> = 0. Valor por defecto: 12

"grid-line-pattern" Lectura
Patrón Dash utilizado para dibujar las líneas de cuadrícula de vista de árbol

"grid-line-width" Lectura
Anchura, en píxeles, de las líneas de cuadrícula Vista de árbol

"horizontal-separator" Lectura
el espacio horizontal entre las células. Debe ser un número par.
Valores posibles:> = 0. Valor por defecto: 2

"indent-expanders" Lectura
Si es True, los expansores son sangrados.

"odd-row-color" Lectura
el gtk.gdk.Color de usar para las filas impares. Disponible en GTK + 2.2 y superiores.

"row-ending-details" Lectura
Activa las filas de fondo

"tree-line-pattern" Lectura
Dash patrón utilizado para dibujar las líneas de la vista en árbol

"tree-line-width" Lectura
Anchura, en píxeles, de las líneas en vista de árbol

"vertical-separator" Lectura
El espacio vertical entre las filas. Debe ser un número par.
Valores posibles:> = 0. Valor por defecto: 2

Señales que emite:

(Las de sus ancestros):
gobject.GObject Signal Prototypes
gtk.Object Signal Prototypes
gtk.Widget Signal Prototypes
gtk.Container Signal Prototypes

(Las propias con sus retrollamadas para capturarlas):
"columns-changed"

```
def callback(treeview, user_param1, ...)
```

"cursor-changed"
def callback(treeview, user_param1, ...)

"expand-collapse-cursor-row"
def callback(treeview, logical, expand, open_all, user_param1, ...)

"move-cursor"
def callback(treeview, step, count, user_param1, ...)

"row-activated"
def callback(treeview, path, view_column, user_param1, ...)

"row-collapsed"
def callback(treeview, iter, path, user_param1, ...)

"row-expanded"
def callback(treeview, iter, path, user_param1, ...)

"select-all"
def callback(treeview, user_param1, ...)

"select-cursor-parent"
def callback(treeview, user_param1, ...)

"select-cursor-row"
def callback(treeview, start_editing, user_param1, ...)

"set-scroll-adjustments"
def callback(treeview, hadjustment, vadjustment, user_param1, ...)

"start-interactive-search"
def callback(treeview, user_param1, ...)

"test-collapse-row"
def callback(treeview, iter, path, user_param1, ...)

"test-expand-row"
def callback(treeview, iter, path, user_param1, ...)

"toggle-cursor-row"
def callback(treeview, user_param1, ...)

"unselect-all"
def callback(treeview, user_param1, ...)

Descripción General:

Un widget gtk.TreeView se utiliza para mostrar los contenidos de los modelos de aplicación de la

interfaz gtk.TreeModel. Los modelos de árboles proporcionados con GTK y PyGTK son:

* Gtk.ListStore
* Gtk.TreeStore
* Gtk.TreeModelSort

Además, PyGTK proporciona gtk.GenericTreeModel que le permite crear su propio modelo de árbol en su totalidad en Python.

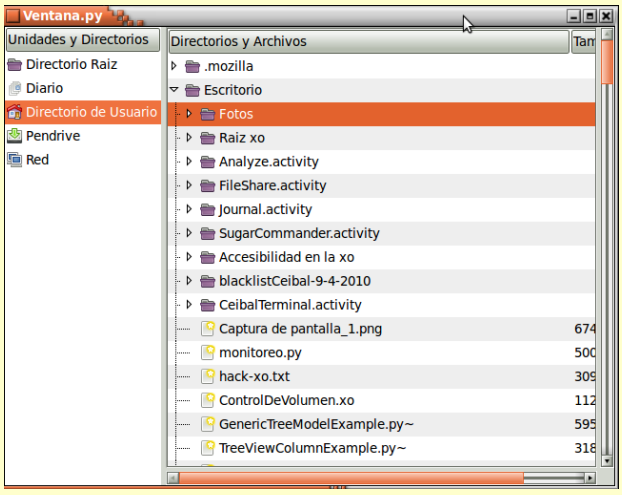
El gtk.TreeView utiliza columnas y renderizadores de celda para mostrar realmente la información del modelo. GTK y PyGTK proporciona la gtk.TreeViewColumn para manejar las columnas del modelo. Los renderizadores de celda pueden ser de los siguientes tipos:

* Gtk.CellRendererPixbuf
* Gtk.CellRendererText
* Gtk.CellRendererToggle

Además, PyGTK proporciona gtk.GenericCellRenderer que le permite crear su propia celda totalmente en Python.

Ejemplo Sencillo de Navegador de Archivos:

El siguiente es un senncillo ejemplo de la interfaz gráfica para un navegador de archivos utilizando TreeView. No se implementa la funcionalidad de copiar, pegar, etc los archivos en el sistema, sino que solo se trata la implementación de los objetos gráficos para la interfaz.



Descripción general:

1. Se crea una ventana.
2. Se crea un gtk.Hpaned y se agrega a la ventana.
3. Se crea un TreeView con modelo gtk.Listore y se agrega a la izquierda del Hpaned.
4. Se crea un TreeView con modelo gtk.TreeStore y se agrega a la derecha del Hpaned.
5. Se implementan funcionalidades para ambos modelos.

Las implementaciones de gtk.Window y gtk.HPaned son necesarias para el ejemplo, pero no se explica nada sobre ellas ya que no es el objeto de esta guia.
Acá sólo se desarrolla la implementación de gtk.TreeView y las diferentes maneras de acceder a sus datos tanto para un modelo de datos gtk.Listore como para gtk.TreeStore.

Paso 1. *(Crear la ventana principal para los demás widget)*

La Ventana Principal:

```
#!/usr/bin/env python

import pygtk
pygtk.require("2.0")
import gtk

from Navegador_de_Archivos import Navegador_de_Archivos

class Ventana():

    def __init__(self):
        self.window=gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_size_request(640, 480)
        self.window.connect("delete_event", self.delete_event)
        self.window.connect("destroy", self.destroy)

        navegador_de_archivos = Navegador_de_Archivos()
        self.window.add(navegador_de_archivos)

        self.window.show_all()

    def delete_event(self, widget, event, data=None):
        print "delete event occurred"
        return False

    def destroy(self, widget, data=None):
        gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    miventana = Ventana()
    miventana.main()
```

Paso 2. *(Crear un Hpaned para contener los TreeView)*

Un Hpaned para los TreeView:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import pygtk
pygtk.require("2.0")
import gtk

from TreeView_Unidades import TreeView_Unidades
from TreeView_Directorios_y_archivos import TreeView_Directorios_y_archivos

class Navegador_de_Archivos(gtk.HPaned):
# Un panel horizontal con el navegador de archivos.
# para agregar directamente a una ventana o al canvas de una ventana sugar.

    def __init__(self):

        gtk.HPaned.__init__(self)

        self.unidadesdealmacenamiento = None
        self.arboldedirectorios = None

        #self.add1(self.area_izquierda_del_panel())
        #self.add2(self.area_derecha_del_panel())
        self.pack1(self.area_izquierda_del_panel(), resize=False, shrink=True)
        self.pack2(self.area_derecha_del_panel(), resize=True, shrink=True)

        # Vinculamos ambos treeview
        self.unidadesdealmacenamiento.asignar_arbol_de_directorios(self.arboldedirectorios)

        # Seleccionamos el primer punto de montaje en la lista para llenar el árbol de directorios
        self.unidadesdealmacenamiento.treeselection.select_path(0)

        self.show_all()

    def area_izquierda_del_panel(self):
    # El widget de la zona izquierda de gtk.HPaned es un ListStore
        self.unidadesdealmacenamiento = TreeView_Unidades()
        return self.unidadesdealmacenamiento

    def area_derecha_del_panel(self):
    # El widget de la zona derecha de gtk.HPaned es un TreeStore
        scrolled_window2 = gtk.ScrolledWindow()
        scrolled_window2.set_policy(gtk.POLICY_AUTOMATIC, gtk.POLICY_AUTOMATIC)

        self.arboldedirectorios = TreeView_Directorios_y_archivos()
        scrolled_window2.add_with_viewport (self.arboldedirectorios)

        return scrolled_window2
```

Paso 3. *(Crear un ListStore para agregar a la izquierda de Hpaned)*

Un ListStore:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import pygtk
pygtk.require("2.0")
import gtk
import gobject
import os

class TreeView_Unidades(gtk.TreeView):
# Una lista de puntos de montaje para agregar en un gtk.HPaned zona izquierda

        def __init__(self):

                self.modelo = None
                self.treeview_arbol = None

                gtk.TreeView.__init__(self)

                # construir ListStore que muestra la lista de archivos en el directorio referido
                self.modelo = self.construir_lista() # Nos devolverá un  ListStore
                self.construir_columnas_de_listas() # Creará las columnas de nuestro ListStore
                self.Llenar_ListStore() # Agrega las filas que hemos definido a nuestro ListStore

                # Creamos un objeto  treeselection para manejar las selecciones de fila
                self.treeselection = self.get_selection() # treeview.get_selection()
                self.treeselection.set_mode(gtk.SELECTION_SINGLE)

                # conectamos el objeto treeselection a una funcion que manejará las selecciones.
                # Cada vez que el usuario seleccione una fila se ejecutará la función  func_selecciones.
                self.treeselection.set_select_function(self.func_selecciones, self.modelo, True)

                self.set_model(self.modelo)

                self.show_all()

        def asignar_arbol_de_directorios(self, treeview_arbol):
        # A traves de esta función, cuando creemos el treestore, los vincularemos para que cuando el usuario
        # selecciones una fila en este liststore, se carguen los directorios y archivos en el treestore.
                self.treeview_arbol = treeview_arbol

        def func_selecciones(self, selection, model, path, is_selected, user_data):
        # Control de selecciones sobre ListTore
                # otener la carpeta almacenada en esta fila
                iter = model.get_iter(path) # iter es un objeto treeiter para itinerar sobre el modelo
                directorio =  model.get_value(iter, 2) # obtengo el valor de la columna 2 del ListStore
                # ahora, con estos datos hay que llenar el arbol de directorios
                self.treeview_arbol.leer_directorio(directorio)

                return True # Debe devolver True para que se realice la selección.
```

```
def construir_lista(self):
# Construye Listore para carpetas y unidades
        modelo = gtk.ListStore (str, str, str) # con 3 columnas de tipo str
        return modelo

def construir_columnas_de_listas(self):
# Columnas para ListStore
        # Nombre de la columna, tipo de cellrender, Numero de columna comenzando en 0
        columna = gtk.TreeViewColumn('Unidades y Directorios') # primera columna de datos

        # crear un CellRenderers para mostrar los datos
        celda_de_imagen = gtk.CellRendererPixbuf() # para el icono
        celda_de_texto = gtk.CellRendererText() # para el texto
        celda_de_direccion = gtk.CellRendererText() # para la direccion en el sistema de archivos
        celda_de_direccion.set_property('visible', False) # la hacemos invisible

        # agregar los cellrenderer a la columna
        # usamos pack_start para agregar todos los cellrenderer a una misma columna del modelo.
        columna.pack_start(celda_de_imagen, False)
        columna.pack_start(celda_de_texto, True)
        columna.pack_start(celda_de_direccion, True)

        self.append_column (columna) # treeview.append_column (columna)

        columna.set_attributes(celda_de_imagen, stock_id=1) # el ícono
        columna.set_attributes(celda_de_texto, text=0) # el texto
        columna.set_attributes(celda_de_direccion, text=2) # la direccion en el sistema de archivos

        # configurar los atributos de las celdas
        # GTK+ 2.0 doesn't support the "stock_id" property
        if gtk.gtk_version[1] < 2:
                columna.set_cell_data_func(celda_de_imagen, self.make_pb)
                # la función  self.make_pb nos devuelve el ícono correcto
        else:
                columna.set_attributes(celda_de_imagen, stock_id=1)
        columna.set_attributes(celda_de_texto, text=0)

def make_pb(self, columna, celda_de_texto, model, iter):
        # Los iconos del ListStore
        stock = model.get_value(iter, 1)
        pb = self.render_icon(stock, gtk.ICON_SIZE_MENU, None)
        celda_de_texto.set_property('pixbuf', pb)
        return

def Llenar_ListStore(self):
# Llenamos la lista con puntos de montaje elegidos
        self.modelo.append([ 'Directorio Raiz', gtk.STOCK_DIRECTORY, "/"])
        self.modelo.append([ 'Diario',
                gtk.STOCK_DND_MULTIPLE,"/home/olpc/.sugar/default/datastore/store"])
        self.modelo.append([ 'Directorio de Usuario', gtk.STOCK_HOME,
                os.path.join(os.environ['HOME'])])
        self.modelo.append([ 'Pendrive', gtk.STOCK_SAVE, "/media"])
```

Nota:
Los íconos utilizados pertenecen a Stock Items: http://library.gnome.org/devel/pygtk/stable/gtk-stock-items.html

Paso 4. *(Crear un TreeStore para agregar a la derecha de Hpaned)*

Un TreeStore:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import pygtk
pygtk.require("2.0")
import gtk
import gobject
import os

class TreeView_Directorios_y_archivos(gtk.TreeView):

        def __init__(self):

                self.modelo = None

                gtk.TreeView.__init__(self)

                self.set_property("rules-hint", True)  # alterna los colores de las filas
                self.set_property("enable-tree-lines", True) # muestra lineas de directorio a  directorio

                self.modelo = self.construir_arbol() # contruye el TreeStore
                self.construir_columnas_de_arbol() # construye las columnas para el TreeStore

                # Capturamos las señales del treeview y las manejamos con nuestras funciones
                self.connect("row-expanded", self.callback_expand, None) # cuando se expande la fila
                self.connect("row-activated", self.callback_activated, None) # cuando se hace doble click sobre la fila
                self.connect("row-collapsed", self.callback_collapsed, None) # cuando se colapsa la fila

                # Detectar eventos del mouse, en particular click derecho para crear menu emergente
                self.add_events(gtk.gdk.BUTTON2_MASK)
                self.connect("button-press-event", self.handler_click)

                self.set_model(self.modelo)

                self.show_all()

        def handler_click(self, widget, event):
        # reacciona a los clicks sobre las filas de tresstore
                boton = event.button # obtenemos el botón que se presionó
                pos = (event.x, event.y) # obtenemos las coordenadas
                tiempo = event.time # obtenemos el tiempo

                # widget es TreeView (widget.get_name())
                # Obteniendo datos a partir de coordenadas de evento
                path, columna, xdefondo, ydefondo = widget.get_path_at_pos(event.x, event.y)
                # TreeView.get_path_at_pos(event.x, event.y) devuelve:
                # * La ruta de acceso en el punto especificado (x, y), en relación con las coordenadas widget
                # * El gtk.TreeViewColumn en ese punto
                # * La coordenada X en relación con el fondo de la celda
                # * La coordenada Y en relación con el fondo de la celda
```

```
            if boton == 1:
                return
            elif boton == 3:
                # Abrir menu – popup, pasando el path de la fila seleccionada
                self.crear_menu_emergente(widget, boton, pos, tiempo, path)
                return
            elif boton == 2:
                return

    def crear_menu_emergente(self, widget, boton, pos, tiempo, path):
    # un menu para agregar o eliminar directorios o archivos

            menu = gtk.Menu()

            # Items del menu
            copiar = gtk.MenuItem("Copiar")
            cortar = gtk.MenuItem("Cortar")
            borrar = gtk.MenuItem("Borrar")
            pegar = gtk.MenuItem("Pegar")

            # Agregar los items al menu
            menu.append(copiar)
            menu.append(cortar)
            menu.append(borrar)
            menu.append(pegar)

            # Se conectan las funciones de retrollamada a la senal "activate"
            copiar.connect_object("activate", self.seleccionar_origen, path, "Copiar")
            cortar.connect_object("activate", self.seleccionar_origen, path, "Cortar")
            borrar.connect_object("activate", self.seleccionar_origen, path, "Borrar")
            pegar.connect_object("activate", self.seleccionar_origen, path, "Pegar")

            menu.show_all()
            menu.popup(None, None, self.posicionar_menu, boton, tiempo, None)

    def seleccionar_origen(self, path, accion):
    # Recibe el path de la fila seleccionada en el modelo y la accion a realizar

            if accion == "Copiar":
                pass
            elif accion == "Cortar":
                pass
            elif accion == "Borrar":
                pass
            elif accion == "Pegar":
                pass

            print "Seleccionado: ", path, accion

    def posicionar_menu(self, widget, pos):
    # Establece la posicion del menu desplegable
            print "Posicionando menu desplegable"

    def construir_arbol(self):
    # Construimos el TreeStore
            modelo = gtk.TreeStore (str, str, str, str) # Cuatro columnas de tipo str
            return modelo
```

```
def construir_columnas_de_arbol(self):
# Columnas para el TreeStore

        columna = gtk.TreeViewColumn('Directorios y Archivos') # primera columna

        celda_de_imagen = gtk.CellRendererPixbuf() # para el ícono
        celda_de_texto = gtk.CellRendererText() # para el texto
        celda_de_direccion = gtk.CellRendererText() # para la dirección

        # agregamos los cellrenderer a la 1º columna
        columna.pack_start(celda_de_imagen, False)
        columna.pack_start(celda_de_texto, True)
        columna.pack_start(celda_de_direccion, True)

        celda_de_direccion.set_property('visible', False) # la hacemos invisible

        columna.set_property('resizable', True) # para que el usuario pueda cambiarle el ancho

        self.append_column (columna)

        columna.set_attributes(celda_de_imagen, stock_id=1) # el ícono
        columna.set_attributes(celda_de_texto, text=0) # el texto
        columna.set_attributes(celda_de_direccion, text=2) # la direccion

        # configurnado cellrenderer
        if gtk.gtk_version[1] < 2:
                columna.set_cell_data_func(celda_de_imagen, self.make_pb)
        else:
                columna.set_attributes(celda_de_imagen, stock_id=1)
        columna.set_attributes(celda_de_texto, text=0)

        # para una segunda columna
        render2 = gtk.CellRendererText()
        columna2 = gtk.TreeViewColumn('Tamaño', render2, text=3)

        self.append_column (columna2)

        # Establecemos la columna donde aparecerá el icono expansor.
        # Por defecto es la primer columna, este metodo permite cambiarla
        self.set_expander_column(columna)

def make_pb(self, columna, celda_de_texto, model, iter):
# Los iconos
        stock = model.get_value(iter, 1)
        pb = self.render_icon(stock, gtk.ICON_SIZE_MENU, None)
        celda_de_texto.set_property('pixbuf', pb)
        return

def leer_directorio(self, directorio):
# Recibe el directorio base desde donde se armará el arbol del treestore.
        self.modelo.clear() # Limpia el TreeStore

        path = 0
        carpeta = (directorio, path)

        self.leer(carpeta) # Agrega directorios y archivos en un nodo del treestore
```

```
def leer(self, carpeta):
# Agrega directorios y archivos en un nodo del treestore

  try:
        directorio = carpeta[0] # direccion de la carpeta que vamos a leer
        path = carpeta[1] # nodo del treestore donde se insertará la carpeta

        if path == 0:
              iter = self.modelo.get_iter_first()
        else:
              iter = self.modelo.get_iter(path)

        archivos = []
        for archivo in os.listdir(os.path.join(directorio)): # lee el directorio

              direccion = directorio + "/" + archivo # crea la direccion del archivo o directorio encontrado

              if os.path.isdir(os.path.join(direccion)):
              # si es un directorio

                    iteractual = self.modelo.append(iter,[archivo, gtk.STOCK_DIRECTORY,
                                direccion, ""])

                    # para mostrar expansor de fila en los directorios aunque estén vacíos.
                    self.agregar_nada(iteractual)

              elif os.path.isfile(os.path.join(direccion)):
              #si es un archivo
                    archivos.append(direccion)

        for x in archivos:
        # los archivos se agregan al final
              archivo = os.path.basename(x)
              self.modelo.append(iter,[archivo, gtk.STOCK_NEW, x, str(os.path.getsize(x))+" bytes"])

  except:
        print "**** Error de acceso a un archivo o carpeta ****"

def agregar_nada(self, iterador):
# para mostrar expansor de fila en los directorios
        self.modelo.append(iterador,["(Vacío)", None, None, None])

def callback_expand (self, treeview, iter, path, user_param1):
# Se ejecuta cuando el usuario expande la fila

        # Obtener los datos del primer hijo en este nodo
        iterdelprimerhijo = treeview.modelo.iter_children(iter) # El primer hijo de esta fila
        valordelprimerhijoenlafila = treeview.modelo.get_value(iterdelprimerhijo, 0)

        # tomar el valor de la direccion almacenada en el item
        valor = treeview.modelo.get_value(iter, 2)
        carpeta = (valor, path)

        # Ver si hay archivos o directorios bajo esta direccion
        if os.listdir(os.path.join(valor)) and valordelprimerhijoenlafila == "(Vacío)":
              #print ". . . Esta direccion contiene carpetas o archivos"
              #print ". . . El modelo contiene", treeview.modelo.iter_n_children(iter), "hijos"
              #print ". . . El valor del primer hijo es:",valordelprimerhijoenlafila
```

```
                self.leer(carpeta)
                treeview.modelo.remove(iterdelprimerhijo)
                #print "Borrando Item: ", valordelprimerhijoenlafila
        else:
                print ". . . Esta direccion está vacía o ya fue llenada"


def callback_activated (self, treeview, path, view_column, user_param1):
# Cuando se hace doble click sobre una fila

        # Obtengo el valor almacenado
        iter = treeview.modelo.get_iter(path)
        valor = treeview.modelo.get_value(iter, 2)

        if os.path.isdir(os.path.join(valor)):
        # Si representa a un directorio

                if treeview.row_expanded(path):
                # Si está expandida, colapsarla
                        treeview.collapse_row(path)
                elif not treeview.row_expanded(path):
                # Si no está expandida, expandirla
                        treeview.expand_to_path(path)

        elif os.path.isfile(os.path.join(valor)):
        # Si representa a un archivo
                pass

def callback_collapsed(self, treeview, iter, path, user_param1):
# Cuando se colapsa una fila, eliminar todos los hijos.
        while treeview.modelo.iter_n_children(iter):
                iterdelprimerhijo = treeview.modelo.iter_children(iter)
                treeview.modelo.remove(iterdelprimerhijo)
        # agregar un hijo vacío
        self.agregar_nada(iter)
```

Resumen *(solo treeview)*:

En el ejemplo, los Treeview o vista de arbol, se han creado heredando de **gtk.TreeView**

```
class TreeView_Directorios_y_archivos(gtk.TreeView):
        def __init__(self):
                self.modelo = None
                gtk.TreeView.__init__(self)
```

Luego se les ha asignado un modelo de datos que puede ser **gtk.ListStore** o **gtk.TreeStore**. En ambos casos, se deben definir la cantidad de columnas de este modelo y el tipo de datos que contendrá cada una de ellas.

```
modelo = gtk.TreeStore (str, str, str, str) # creamos el modelo
nuestroTreeView.set_model(self.modelo) # lo asignamos a nuestro treeview
```

La forma en que se crean las columnas de cada modelo no corresponden a treeview sino a cada modelo en particular por lo cual no lo explicaré en ese resumen.

A través del establecimiento de las propiedades del treeview podemos manejar mucho de su apariencia:

```
nuestroTreeView.set_property("rules-hint", True)
nuestroTreeView.set_property("enable-tree-lines", True)
```

(Al comienzo de este documento tienes la lista completa de propiedades con su explicación.)

Y mucho del trabajo a realizar lo logramos al detectar las señales que emite treeview cuando el usuario interactua con el, sólo es necesario conectar nuestras funciones a esas señales para responder a gusto a ellas:

```
nuestroTreeView.connect("row-expanded", self.callback_expand, None) # callback_expand es nuestra funcion
nuestroTreeView.connect("row-activated", self.callback_activated, None) # callback_ activated es nuestra funcion
nuestroTreeView.connect("row-collapsed", self.callback_collapsed, None) # callback_ collapsed es nuestra funcion
```

(Cada una de estas señales deben capturarse con una función que respete la sintaxis y argumentos devueltos por la señal. Al comienzo de este documento puedes ver la lista completa de señales y funciones para capturarlas.)

Para obtener los datos del modelo cuando el usuario hace click sobre el mismo, tienes 2 opciones. Puedes crear un objeto gtk.TreeSelection y conectarlo a una función para manejar las selecciones sobre el modelo o puedes capturar los eventos del mouse sobre el modelo.

Para construir una función que maneje las selecciones sobre el modelo, primero creamos el objeto treeselection:

```
nuestroTreeView.treeselection = self.get_selection()
nuestroTreeView.treeselection.set_mode(gtk.SELECTION_SINGLE) # Establecemos el modo de seleccion admitido

# y conectamos el objeto a una funcion que manejará las selecciones
nuestroTreeView.treeselection.set_select_function(self.func_selecciones, self.modelo, True)
```

La funcion:

```
        def func_selecciones(self, selection, model, path, is_selected, user_data):
                iter = model.get_iter(path) # objeto iterador en el modelo
                directorio =  model.get_value(iter, 2) # obtengo datos de 2ª columna
                # Aca haces lo que quieras con los datos obtenidos
                return True # Debe devolver True para que se realice la selección
```

Para utilizar la otra forma, detectar los eventos del mouse, primero definimos:

```
nuestroTreeView.add_events(gtk.gdk.BUTTON2_MASK) # capturamos los eventos del mouse sobre nuestro TreeView
nuestroTreeView.connect("button-press-event", self.handler_click) # y los conectamos con nuestra función de control
```

La Función:

```
    def handler_click(self, widget, event):
        boton = event.button
        pos = (event.x, event.y)
        tiempo = event.time

        # widget es TreeView widget.get_name()
        # Obteniendo datos a partir de coordenadas del evento
        path, columna, xdefondo, ydefondo = widget.get_path_at_pos(event.x, event.y)
        # TreeView.get_path_at_pos(event.x, event.y) devuelve:
        # * La ruta de acceso en el punto especificado (x, y), en relación con las coordenadas widget
        # * El gtk.TreeViewColumn en ese punto
        # * La coordenada X en relación con el fondo de la celda
        # * La coordenada Y en relación con el fondo de la celda

        if boton == 1:
            return
        elif boton == 3:
            # con click derecho creo un menú emergente, pasandole el nodo donde se produjo el evento
            self.crear_menu_emergente(widget, boton, pos, tiempo, path)
            return
        elif boton == 2:
            return
```