Matthias Clasen

mclasen@redhat.com

Table of Contents

Introduction

The current stable branch of the GTK+ stack, consisting of GLib 2.6, Pango 1.8 and GTK+ 2.6 was released last December. This may seem like a long time in the world of 6 month release cycles, but 2.6 contains a considerable number of new APIs, ranging from whole new widgets to small integration features or miscellaneous API gaps which have been closed. Many of the new features will need some more time to become used throughout the GNOME stack.

The intention of this talk is to help the adoption of the new GTK+ APIs, by making them better known. I will also peek ahead at what GTK+ 2.8 will offer, if time permits.

New or improved widgets

The new widgets in GTK+ 2.6 are for the most part ports or reimplementations of widgets which previously existed higher up in the GNOME stack. They have been integrated into GTK+, since there is really no reason to force applications to link against the whole GNOME stack to use these generally useful widgets. The trend of moving widgets down to GTK+ has started earlier, when GtkFontButton and GtkColorButton made their appearance in GTK+ 2.4, and it will likely continue in the future, since there are some more GNOME widgets on the "wishlist" for future GTK+ releases.

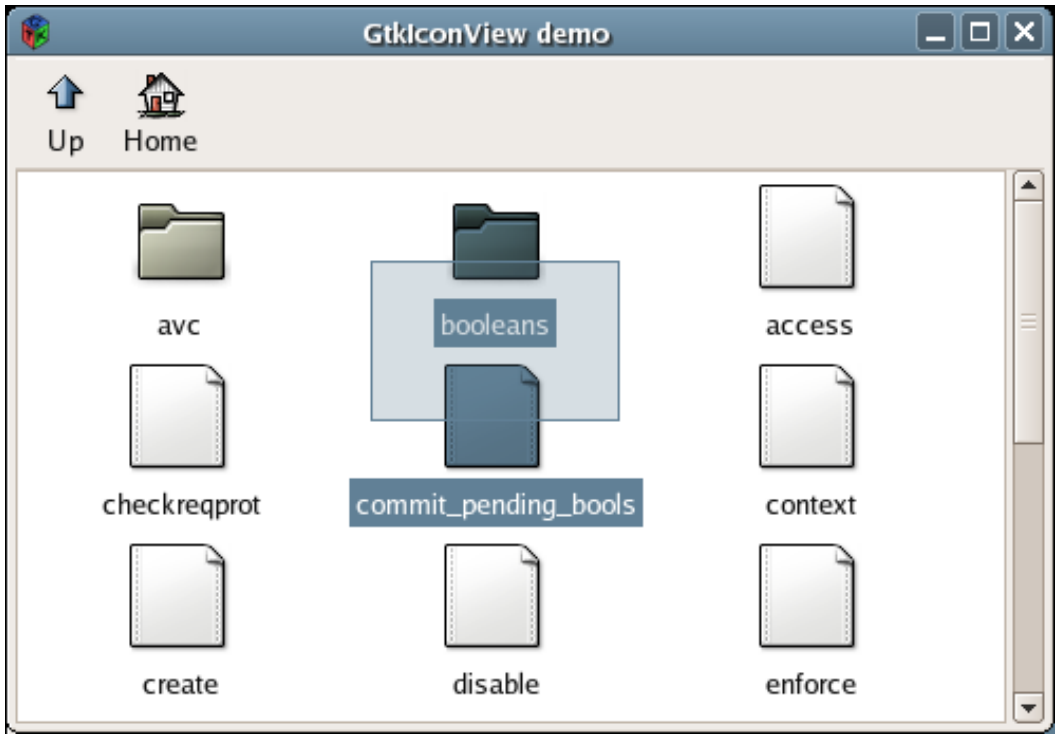
In some cases, widgets have gained new functionality while being ported from libgnomeui to GTK+. Some features which are complicated or hard to implement in GTK+ (like editing or drag-and-drop in the icon view) have been temporarily dropped, and some rarely used or misdesigned features (like dithering support in the color picker button) have been dropped altogether.

GtkIconView

GtkIconView is a widget which displays labeled icons in a grid layout. It is similar to GnomeIconList and at the same time more flexible and more restricted this widget.

It is more flexible, since it uses a model-view architecture and takes icons and labels to display from a tree model. This may look strange at first, as the icon view does only support flat models, not real trees. But the advantage of reusing the tree models is that the infrastructure for sorting and filtering is already there with GtkTreeModelSort and GtkTreeModelFilter. Also, we have the flexibility to associate not just pixbufs and strings, but arbitrary data with items.

The icon view example in gtk-demo nicely demonstrates the power of this concept. It realizes a very simple file system browser. In addition to the name and the icon, it stores a boolean for each item indicating wether it is a directory or or a regular file. It also sets a sort function on the tree model which uses the extra data to sort directories before files.



A GtkIconView with a "rubberband" selection

The areas in which GtkIconView is currently more restricted than GnomeIconList are editing and drag-and-drop. We hope to address these limitations in future GTK+ releases.

The model-view separation makes the GtkIconView API a bit more heavy to use than the GnomeIconList. There is some extra setup to create the model and associate it with the view

```
model = gtk_list_store_new (2, G_TYPE_STRING, GDK_TYPE_PIXBUF);

gtk_icon_view_set_model (view, model);
gtk_icon_view_set_text_column (view, 0);
gtk_icon_view_set_pixbuf_column (view, 1);
```

The single call to append an item to a GnomeIconList

```
gnome_icon_list_append (list, "file.png", "text");
```

has to be replaced by the following

```
pixbuf = gdk_pixbuf_new_from_file ("file.png");
gtk_list_store_append (model, &iter);
gtk_list_store_set (model, &iter,
                    0, pixbuf,
                    1, "text",
                    -1);
```

A more thorough discussion of porting from GnomeIconList to GtkIconView can be found in the GTK+ API documentation, which contains a chapter called "Migrating from GnomeIconList to GtkIconView".

GtkAboutDialog

GtkAboutDialog is a port of the GnomeAbout widget to GTK+, with some extensions. The extensions include clickable hyperlinks and email addresses, a separate license window and support for artists' credits.

To make use of the hyperlink support, you have to call the functions gtk_about_dialog_set_email_hook() and gtk_about_dialog_set_url_hook() to set up suitable callbacks to open the URL that has been clicked. When you are using the GNOME stack, you probably want to use gnome_url_show() in your callback, which will automatically take care of opening the user's preferred browser or email client.

A GtkAboutDialog

GtkAboutDialog offers a convenience function gtk_show_about_dialog() which makes it very easy to setup and associate an about dialog with a toplevel window. Repeated calls to gtk_show_about_dialog() will reuse the previously create about dialog. The code below demonstrates how to use gtk_show_about_dialog() to construct the dialog shown above.

```
void
open_link (GtkAboutDialog *dialog,
           const gchar    *link,
           gpointer        data)
{
  gnome_url_show (link, NULL);
}

const gchar *authors[] =
{
  "Peter Mattis",
  "Spencer Kimball",
  "Josh MacDonald",
  "and many more...",
  NULL
};
```

```
const gchar *documentors[] =
{
  "Owen Taylor",
  "Tony Gale",
  "Matthias Clasen <clasen@redhat.com>",
  "and many more...",
  NULL
};

const gchar *license = "...";

GdkPixbuf *pixbuf = gdk_pixbuf_new_from_file (filename, NULL);

gtk_about_dialog_set_email_hook (open_link, NULL, NULL);
gtk_about_dialog_set_url_hook (open_link, NULL, NULL);
gtk_show_about_dialog (GTK_WINDOW (window),
                  "name", "GTK+ Code Demos",
                       "version", "2.6.3",
                       "copyright", "(C) 1997-2004 The GTK+ Team",
                       "license", license,
                       "website", "http://www.gtk.org",
                       "comments", "Program to demonstrate GTK+ functions.",
                       "authors", authors,
                       "documenters", documentors,
                       "logo", pixbuf,
                       NULL);
```

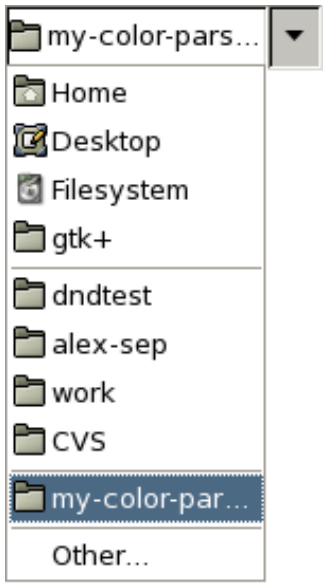
The corresponding code using GnomeAbout would be slightly different, since you have to construct and show the dialog in two steps. Also note that gtk_show_about_dialog() expects a list of key-value pairs as arguments where gnome_about_new() just expects the value.

```
GtkWidget *about_box = gnome_about_new ("GTK+ Code Demos",
                                        "2.6.3",
                                        "(C) 1997-2004 The GTK+ Team",
                                        "Program to demonstrate GTK+ functions.",
                                        authors,
                                        documentors,
                                        "Translation credits",
                                        pixbuf);
gtk_widget_show (about_box);
```

Again, the GTK+ API docs have a chapter called "Migrating from GnomeAbout to GtkAboutDialog", which contains further hints.

GtkFileChooserButton

The GtkFileChooserButton complements the GtkColorButton and GtkFontButton which appeared in GTK+ 2.4. These three widgets are primarily used in preference dialogs. The GtkFileChooserButton is meant to replace GnomeFileEntry. Since it implements the GtkFileChooser interface, it has only a very minimal API, most of the GnomeFileEntry functions are replaced by GtkFileChooser functions.



Choosing a directory

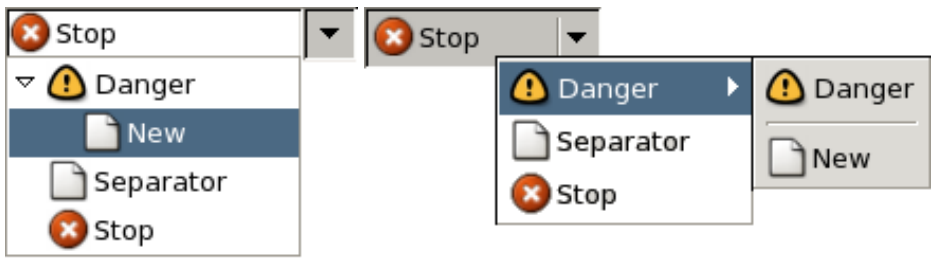
GtkMenuToolButton

GtkMenuToolButton provides a convenient API for the dropdown menus from toolbar buttons which are commonly seen on the "Forward" and "Back" buttons of browsers. The code in GTK+ 2.6 has been derived from similar widgets in Galeon, Epiphany and GEdit. Creating a menu tool button is very straighforward:

```
menu = gtk_menu_new ();

item = gtk_menu_tool_button_new_from_stock (GTK_STOCK_OPEN);
gtk_menu_tool_button_set_menu (GTK_MENU_TOOL_BUTTON (item), menu);
gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
```

GtkComboBox

The GtkComboBox widget made its first appearance in GTK+ 2.4. A somewhat unusual feature of this widget is that it is also follows the model-view pattern and uses tree models. In 2.4, the models had to be flat lists. The major new feature in 2.6 is that trees are now supported. They are displayed as nested menus or in a treeview, depending on the GtkComboBox::appears-as-list style property, which switches between option menu style and Windows combo box style.



GtkTreeView has acquired several new features in GTK+ 2.6 specifically to make the popup in this example work as expected: The selection follows the pointer, and the rows expand and collapse on mouse-over.

Note how the menuitems with submenus are repeated as the first item in the submenu. This is necessary, since GTK+ does not allow menuitems with submenus to be activated. The following code was used to create the example shown above.

```
  /* Create the model */
  store = gtk_tree_store_new (3, GDK_TYPE_PIXBUF, G_TYPE_STRING, G_TYPE_BOOLEAN);

  /* Create the combo box */
  combo = gtk_combo_box_new ();

  pixbuf = gtk_widget_render_icon (combo, GTK_STOCK_DIALOG_WARNING,
       GTK_ICON_SIZE_BUTTON, NULL);
  gtk_tree_store_append (store, &iter, NULL);
  gtk_tree_store_set (store, &iter, 0, pixbuf, 1, "Danger", 2, TRUE, -1);

  pixbuf = gtk_widget_render_icon (combo, GTK_STOCK_NEW,
       GTK_ICON_SIZE_BUTTON, NULL);
  gtk_tree_store_append (store, &iter2, &iter);
  gtk_tree_store_set (store, &iter2, 0, pixbuf, 1, "New", 2, TRUE, -1);

  gtk_tree_store_append (store, &iter, NULL);
  gtk_tree_store_set (store, &iter, 0, pixbuf, 1, "Separator", 2, FALSE, -1);

  pixbuf = gtk_widget_render_icon (combo, GTK_STOCK_STOP,
       GTK_ICON_SIZE_BUTTON, NULL);

  gtk_tree_store_append (store, &iter, NULL);
  gtk_tree_store_set (store, &iter, 0, pixbuf, 1, "Stop", 2, TRUE, -1);

  /* Set the model */
  gtk_combo_box_set_model (GTK_COMBO_BOX (combo), GTK_TREE_MODEL (store));
  gtk_combo_box_set_active_iter (GTK_COMBO_BOX (combo), &iter);

  /* Set up cell renderers */
  renderer1 = gtk_cell_renderer_pixbuf_new ();
  gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (combo), renderer1, FALSE);
  gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combo),
      renderer1, "pixbuf", 0, NULL);

  renderer2 = gtk_cell_renderer_text_new ();
  gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (combo), renderer2, TRUE);
  gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combo),
      renderer2, "text", 1, NULL);
```

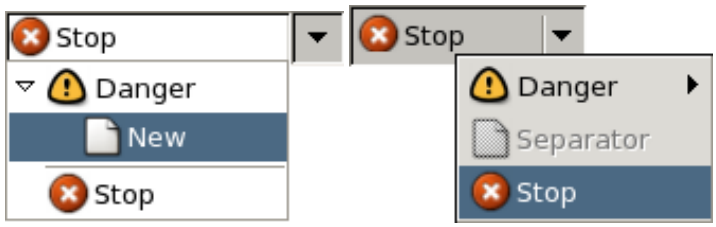
Another new feature of GtkComboBox (and tree views in general) are separators. They are used by setting up a callback function which determines wether a row should be renderered as separator or not. The callback function has access to the model and the row that is being renderered, and can e.g. look at a boolean column or check for a specific string, e.g. "--". In order to turn the "Separator" row in the example above into a separator, we add the call

```
gtk_combo_box_set_row_separator_func (GTK_COMBO_BOX (combo),
                                      is_separator, NULL, NULL);
```

and define the callback as follows:

```
static gboolean
is_separator (GtkTreeModel *model,
       GtkTreeIter  *iter,
       gpointer      data)
{
  gboolean result;

  gtk_tree_model_get (model, iter, 2, &result, -1);
```

```
  return !result;
}
```



GtkComboBox items (and tree view rows in general) can be insensitive, which means they can't be selected, and are grayed out. This is implemented as by the GtkCellRenderer::sensitive property; a row is treated as insensitive for selection purposes if *all* cells have the sensitive property set to FALSE. To make the "Separator" row in the previous example insensitive, we can simply map the sensitive property to column 2 in the gtk_cell_layout_set_attributes calls:

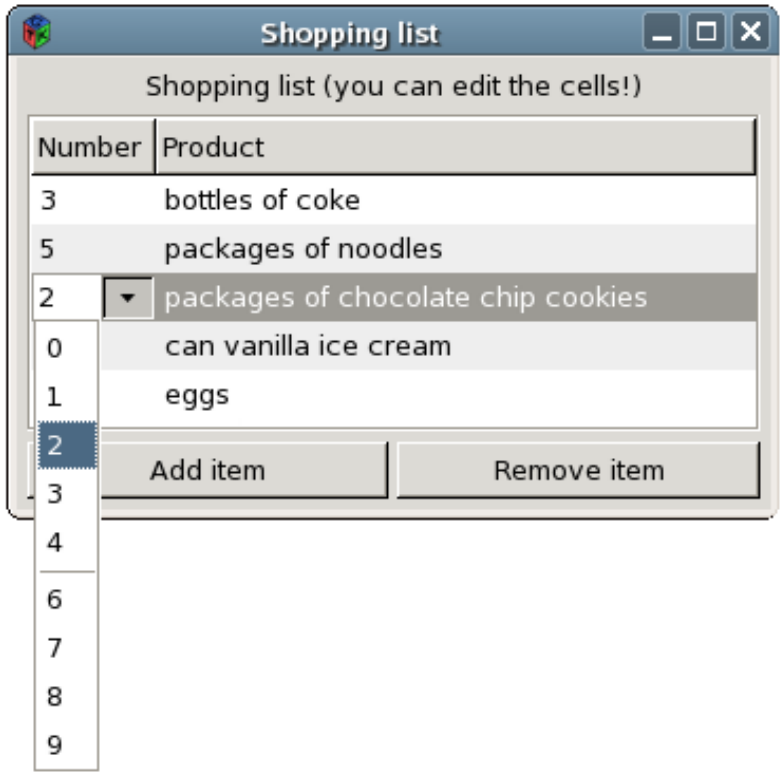
```
  gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combo), renderer1,
      "pixbuf", 0,
      "sensitive", 2, NULL);

  gtk_cell_layout_set_attributes (GTK_CELL_LAYOUT (combo), renderer2,
      "text", 1,
      "sensitive", 2, NULL);
```

New cell renderers

Cell renderers are responsible for drawing the contents of treeview cells. They are also responsible for setting up a suitable widget for editing the contents of a cell, if it is editable. Two new cell renderers made their debut in 2.6. GtkCellRendererProgress displays a numeric value as a progress bar. GtkCellRendererCombo displays text, but uses a GtkComboBox or GtkComboBoxEntry to edit the value, where GtkCellRendererText uses an entry.

One conceptual difficulty with using GtkCellRendererCombo is that there are two tree models involved. The cell renderer displays a cell, by rendering a value from the model which backs the tree view. When editing the cell, a GtkComboBox is instantiated whose initial value is the one displayed in the cell. The possible values of the GtkComboBox come from a *different* tree model, lets call it the "value model", which can be set using the GtkCellRendererCombo::model property. The normal case is probably to use the same value model for all rows, but it is possible to choose a different one for each row.

An example for using a combo cell renderer has recently been added to gtk-demo. It also shows how to use the GtkCellRenderer::editing-started signal to do custom setup of the editable widget. This signal is also a 2.6 addition.



GtkCellRendererCombo in action

The code to set up the GtkCellRendererCombo in this example looks as follows:

```
numbers_model = gtk_list_store_new (1, G_TYPE_STRING);

for (i = 0; i < 10; i++)
  {
    char str[2] = { '0' + i, '\0' };

    gtk_list_store_append (numbers_model, &iter);
    gtk_list_store_set (numbers_model, &iter, COLUMN_NUMBER_TEXT, str, -1);
  }

renderer = gtk_cell_renderer_combo_new ();
g_object_set (renderer,
              "model", numbers_model,
              "text-column", COLUMN_NUMBER_TEXT,
              "has-entry", FALSE,
              "editable", TRUE,
              NULL);
g_signal_connect (renderer, "editing-started", G_CALLBACK (editing_started), NULL);
g_signal_connect (renderer, "edited", G_CALLBACK (cell_edited), items_model);

gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW (treeview),
                                             -1, "Number", renderer,
                                             "text", 0,
                                             NULL);
```

The code is using the following callback to setup the combo box before editing starts. For demonstration purposes, we're turning row 5 into a separator.

```
static gboolean
separator_row (GtkTreeModel *model,
               GtkTreeIter  *iter,
```

```
               gpointer      data)
{
  GtkTreePath *path;
  gint idx;

  path = gtk_tree_model_get_path (model, iter);
  idx = gtk_tree_path_get_indices (path)[0];
  gtk_tree_path_free (path);

  return idx == 5;
}

static void
editing_started (GtkCellRenderer *cell,
                 GtkCellEditable *editable,
                 const gchar     *path,
                 gpointer         data)
{
  gtk_combo_box_set_row_separator_func (GTK_COMBO_BOX (editable),
                                        separator_row, NULL, NULL);
}
```

Other new features

In the second part of my talk, I want to mention a number of smaller new features in GTK+, which are easily overlooked.

Clipboard improvements

GTK+ can cooperate with a clipboard manager to persistently store the clipboard contents beyond the lifetime of the application. The API for this consists of the two functions gtk_clipboard_set_can_store() and gtk_clipboard_store().

The protocol used for transferring the data to the clipboard manager works in a lazy way, ie. no data is transferred unless the application is about to exit while it owns the clipboard. A prototype clipboard manager implementation has been made by Anders Carlsson.

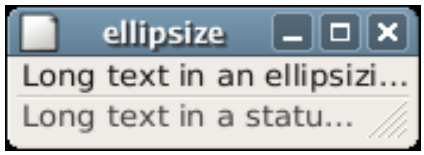
Another extension of the clipboard API in 2.6 is better support for storing images and uris in the clipboard. There is a whole family of new functions for this:

```
gtk_clipboard_set_image()
gtk_clipboard_request_image()

gtk_selection_data_set_pixbuf()
gtk_selection_data_get_pixbuf()

gtk_drag_source_add_image_targets()
gtk_drag_dest_add_image_targets()
```

These functions support all image formats supported by gdk-pixbuf. Similar functions exist to transfer text in any supported format and to transfer URIs. URIs are transferred using the text/uri-list MIME type which is specified in RFC 2483. GLib 2.6 offers the function g_uri_list_extract_uris() to extract the uris from such a list.

Ellipsization and rotated text

Pango supports ellipsization (i.e. typographically correct insertion of ellipsis characters to shorten long texts) and rotation of text since 1.8. GTK+ supports this in the relevant places now. Labels can be rotated using the GtkLabel::angle property. Ellipsization can be turned on for labels and progress bars using the gtk_label_set_ellipsize() and gtk_progress_bar_set_ellipsize() functions. Status bars are ellipsized by default. GtkCellRendererText has an ellipsize property which can be set to turn on ellipsization in treeview cells.



One complication in connection with ellipsization is to ensure that labels keep a reasonable size, and don't collapse to their minimum size, which is the width of three dots, if the label is ellipsized. Another new function, gtk_label_set_max_width_chars(), helps in this case. It allows to specify the desired maximum width of an ellipsized label. The label will be given its natural width, unless that would exceed the maximum with, in which case the label is ellipsized to the the maximum width. Another function that should be mentioned in this context is gtk_file_chooser_button_set_width_chars(), which allows to specify how many characters a GtkFileChooserButton should display before ellipsization kicks in.

Named icons

GTK+ supports named icons in a lot more places in 2.6. These are icons which are specified by name, and change their appearance according to the icon theme of the desktop. This is an important aspect to ensure that applications integrate visually in the desktop. To create a GtkImage widget whose image changes with the theme use gtk_image_set_from_icon_name(). To set a themed window icon, use gtk_window_set_icon_name().

Better Desktop integration

An important aspect of being a good desktop application is to respect global preferences set by the user. The GTK+ mechanism for this is called "settings". These are managed by the gnome-settings-daemon, and applications are notified about changes in the settings, in order to allow settings to change on-the-fly. This is how e.g. theme switching works. GTK+ 2.6 introduces a number of new settings, which allow desktop-wide control of more UI details:

Images in menus

The gtk-menu-images setting allows the user to suppress images in menu items. Applications don't have to do anything special; GtkImageMenuItem automatically reacts to this setting.

Images in buttons

The gtk-button-images setting allows the user to suppress images in buttons. This works automatically for buttons constructed with gtk_button_new_from_stock(). To make it work for manually constructed buttons, images should be added with gtk_button_set_image(). Doing that is not only easier than fiddling with boxes and alignment, but also guarantees that your buttons appear in the same way as stock buttons.

Button order in dialogs

The gtk-alternative-button-order setting allows to select an "alternative button order", as opposed to the normal button order specified by the GNOME HIG. In order to respect this setting, applications should call gtk_dialog_set_alternative_button_order() to set up alternative button orders for their custom dialogs. GTK+ automatically does this for the included dialogs like the color selection dialog and the font selection dialog.

This setting is *not* not controlled by the gnome-settings-daemon, since it is not meant as configurable parameter inside a desktop environment. It is set in the rc file of the ms-windows theme, to make GTK+ applications adapt to the native button order on Windows.

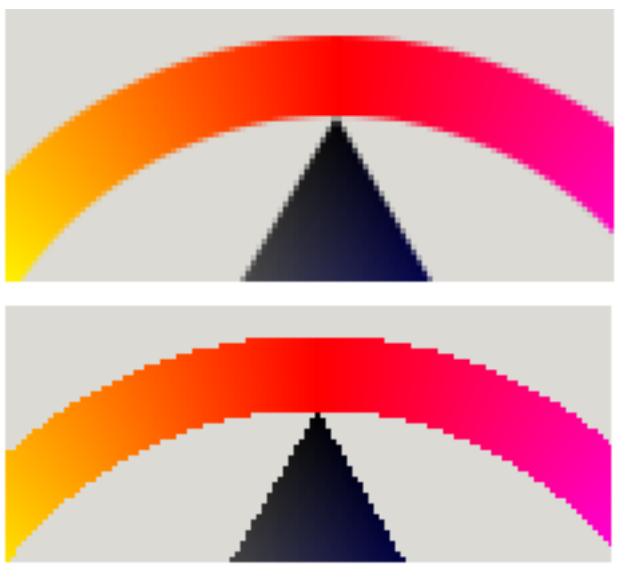
Here is an example which shows how to set up a dialog with the regular button order "Button", "Cancel", "OK", and alternative button order "Button", "OK, "Cancel":

```
dialog = gtk_dialog_new_with_buttons ("Dialog", NULL, 0,
                                      "Button", GTK_RESPONSE_APPLY,
                                      GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                                      GTK_STOCK_OK, GTK_RESPONSE_OK,
                                      NULL);

gtk_dialog_set_alternative_button_order (GTK_DIALOG (dialog),
                                         GTK_RESPONSE_APPLY,
                                         GTK_RESPONSE_OK,
                                         GTK_RESPONSE_CANCEL,
                                         -1);
```

What will be new in GTK+ 2.8 ?

Cairo rendering

This is the topic of Owens talk, so I won't go into detail here. In short, it will be possible to obtain a Cairo context from a GDK drawable with gdk_drawable_create_cairo_context(), and then use the Cairo API to draw on the drawable. Apart from a modern drawing API, this gives us nice antialiasing:

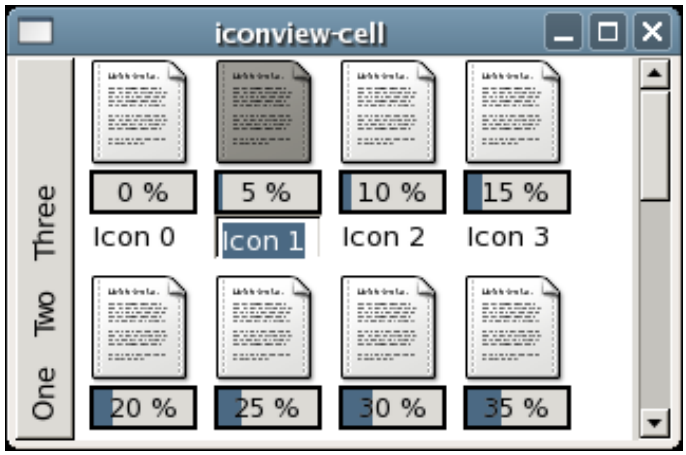


RGBA visuals

Visuals with an alpha channel will allow GTK+ applications to create translucent windows.

Icon view improvements

The GtkIconView will reuse more of the tree view infrastructure by using cell renderers to draw the items. This will remove the restriction to text+icon, and will also make it possible to edit icon view cells.



This screenshot shows a GtkCellRendererProgress used to render a numeric column in the underlying model. One of the text cells is being edited. You can also see a vertical menubar with rotated labels.

Reference material for porting to GTK+ 2.6 APIs

The GTK+ migration checklist.

http://developer.gnome.org/doc/API/2.0/gtk/gtk-migrating-checklist.html

Migrating from GnomeIconList to GtkIconView.

http://developer.gnome.org/doc/API/2.0/gtk/gtk-migrating-GtkIconView.html

Migrating from GnomeAbout to GtkAboutDialog.

http://developer.gnome.org/doc/API/2.0/gtk/gtk-migrating-GtkAboutDialog.html

Migrating from GnomeColorPicker to GtkColorButton.

http://developer.gnome.org/doc/API/2.0/gtk/gtk-migrating-GtkColorButton.html

Lists of new symbols in GTK+ 2.2, 2.4, and 2.6.

http://developer.gnome.org/doc/API/2.0/gtk/index.html