Introduction to GTK+

Ted Gould

PLUG Developers Meeting

August 4th, 2003

http://gould.cx/ted/projects/gtkintro/

Presentation Outline

- GTK+ and helper libraries
- Glib Object Model
- GTK+ Programming

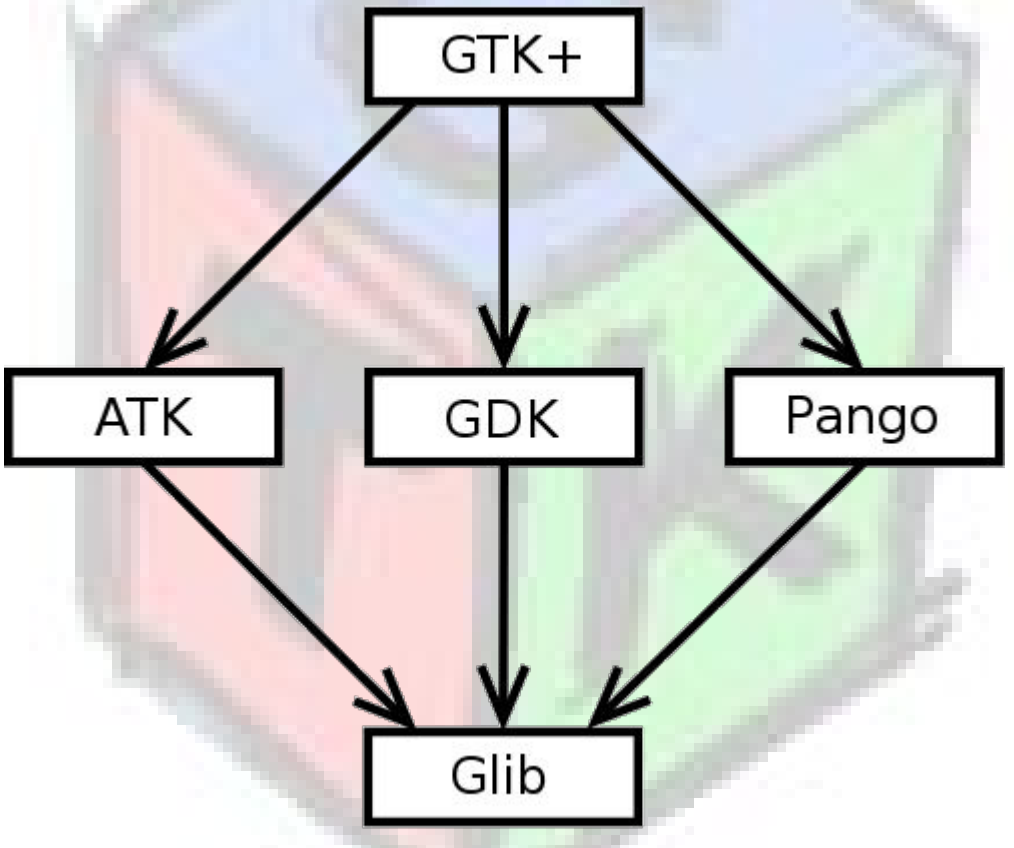
<break>

- GNOME Libraries
- References

GTK+ Overview

- Developed to move The GIMP off of Motif
- Realized that C could be object oriented
- Created in C for compatibility (every modern language can load C libraries)
- Large number of bindings (Effiel, Java, Ruby...)
- GUI Interface designer: Glade
- License: LGPL
- Used in the GNOME project (and many

Helping out GTK+



Pango Text Rendering Library

- Greek 'Pan' for 'All' and Japanese 'Go' for language
- Internationalized text rendering library – not actually GTK+ specific, but used by GTK+
- Uses Unicode internally
- Focus on 'correct' rendering of anything
- Font system and toolkit independent
- For modern Linux uses XFT

Accessibility Toolkit

- Hard to find documentation on :)
- Allows GTK (and other) programs to be used by screen readers, magnifiers, etc.
- Developed by the developers that built the accessibility for Java
- Required for many corporations (especially gov't) to use software

GDK: Gimp Display Kit

- Library to perform the actual rendering to the display
- Abstracts out the display so that it can be X11 or Win32 or Cocoa or whatever
- Also provides some pixmap functions if you need those (not covered today)

Glib C Utility Library

- Makes C 'easy to use'
- Provides much of the functionality that is replicated in many programs
- Has things like: memory allocation, linked lists, hash tables, strings and more
- One key feature: basis of object oriented C with GObject hierarchy

Glib Objects

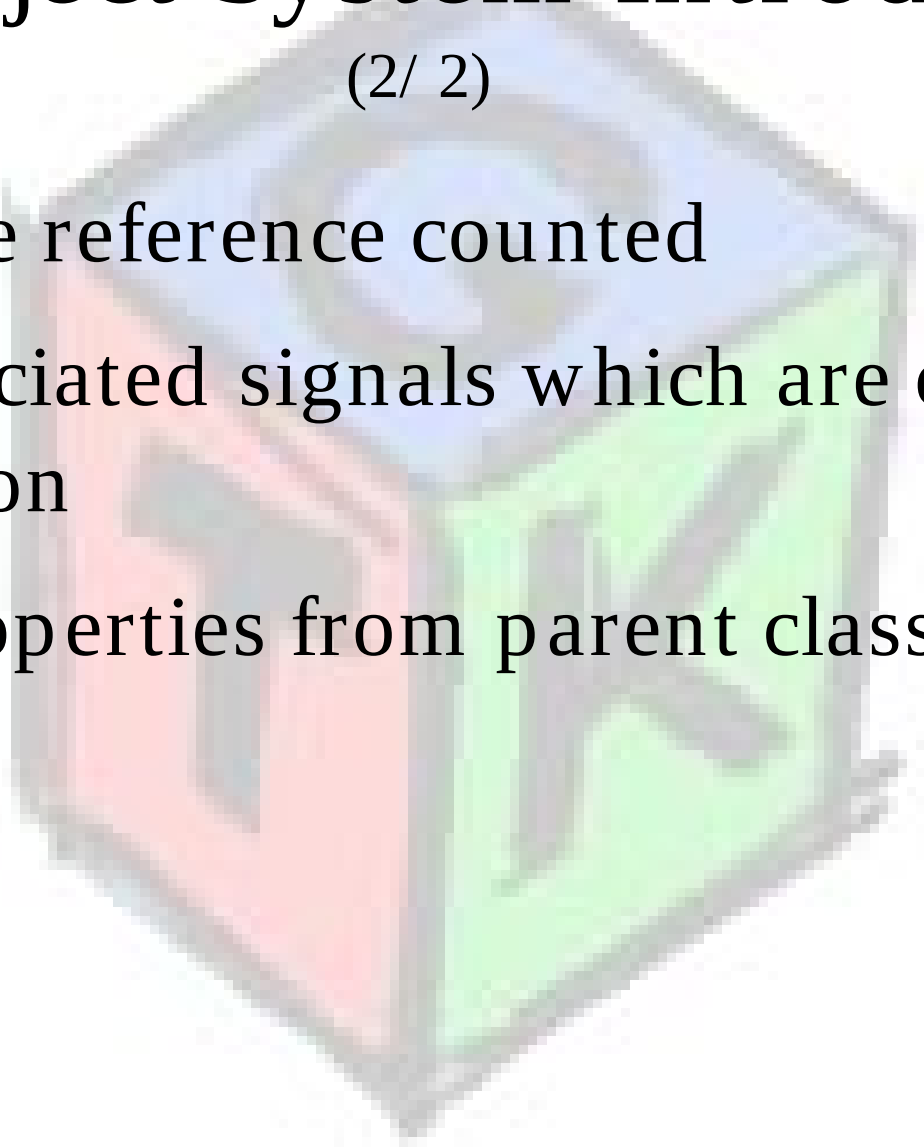
Glib: Object System Introduction

(1/ 2)

- Object oriented programming is about thinking about 'objects' of data with associated functions
- Many languages provide syntactic assistance for this type of programming (C++, Java, etc.)
- Object Oriented programming allows for intellectual separation of code into digestible components
- Glib accomplishes this with C by using

Glib: Object System Introduction

(2/ 2)



- Objects are reference counted
- Have associated signals which are created on construction
- Inherit properties from parent class

Glib: Object Structure

- First element is the parent
- Contains values that are relevant to this object

```
struct MyObject {
   GObject parent;
   int myval;
};
```

- Class structure contains functions that operate on this object

```
struct MyObjectClass {
   GObjectClass parent;
   void (*SetMyVal) (int);
};
```

Glib: Object Macros

- Macros are used for verifying the object type and casting. Common implementation below:

```
#define MY_OBJECT_TYPE   (my_object_get_type())

#define MY_OBJECT(obj)  (G_TYPE_CHECK_INSTANCE_CAST
  ((obj), MY_OBJECT_TYPE, my_object)

#define MY_IS_OBJECT(obj)  (G_TYPE_CHECK_INSTANCE_TYPE
  ((obj), MY_OBJECT_TYPE)
```

Glib: Creating a type

- The get_type function registers a type with Glib, and if it's already registered, just returns it

```
GType my_object_get_type (void) {
  static GType type = 0;
  if (type == 0) {
      GTypeInfo info = {
          sizeof(MyObjectClass), NULL, NULL,
          (GClassInitFunc)my_object_class_init,
          NULL, NULL, sizeof(MyObject), 16,
          (GInstanceInitFunc)my_object_init};

      type = g_type_register_static(
           G_TYPE_OBJECT, “MyObject”, &info, 0);}
  return type;}
```

GObject creation and destruction

- Glib provides space for an initialization function
  - This function allocates all memory for the instance
  - Sets all values to a benign state
- And a destruction function for the object
  - Actually several layers of destruction
  - Free's all memory
  - Gets called when the reference count goes to zero

Glib: Object Creation

- Now everything is really easy ^_^
- One function call to create object

```
g_type_new(MY_OBJECT_TYPE, NULL)
```

- Calls
  - My Object Instance Initialization
  - If the class hasn't been initialized it will do that
  - GObject Instance Initialization
- Many objects also provide a: `my_object_new()`

Other Glib Handy things

- Standardized types (gint, guint, gint32...)
- Singly linked lists, doubly linked lists
- Hash tables
- Heaps and memory allocation
- Dynamic module loading

Programming GTK+

Basic GTK+ Program

```
#include <gtk/gtk.h>

int main(int argc, char * argv[]) {
    GtkWidget * window;

    gtk_init(&argc, &argv);

    window = gtk_window_new(
                          GTK_WINDOW_TOPLEVEL);
    gtk_widget_show (window);

    gtk_main();

    return 0;
}
```

Initialize GTK+

Create new object

GTK+ main loop

GTK+ Main Loop

- Most GTK+ programs run with a single line of execution
- First, this line of execution sets everything up (builds windows, open network connections...)
- Then, it needs to wait for user interaction
- GTK+ main loop waits for the interaction and sends the *events* on to *event handlers*
- Also can include things like timers

Program with an event

```
static gboolean del_ev (GtkWidget * widget,
                GdkEvent * event, gpointer data) {
  g_print ("delete event occured\n");
  return TRUE; /* ERROR */ }

static void destroy (GtkWidget * widget,
                     gpointer data) {
  gtk_main_quit(); }

int main() {
  ....
  g_signal_connect(G_OBJECT(window),
    "delete_event", G_CALLBACK(del_ev), NULL);
  g_signal_connect(G_OBJECT(window), "destroy"
                   G_CALLBACK(destroy), NULL);
  gtk_main(); }
```

Functions that get called when the event happens

Connects events to functions

Creating a Button

```
static void hello (GtkWidget *widget,
                   gpointer data) {
  g_print("Hello World\n");
  }

static void build_button (GtkWidget *window){
  GtkWidget * button;

  button = gtk_button_new_with_label("Hello");
  g_signal_connect(G_OBJECT(button),
          "clicked", G_CALLBACK(hello), NULL);
  gtk_container_add(GTK_CONTAINER(window),
                    button);
  gtk_widget_show(button);
  return;
  }
```

Called on a click of the button

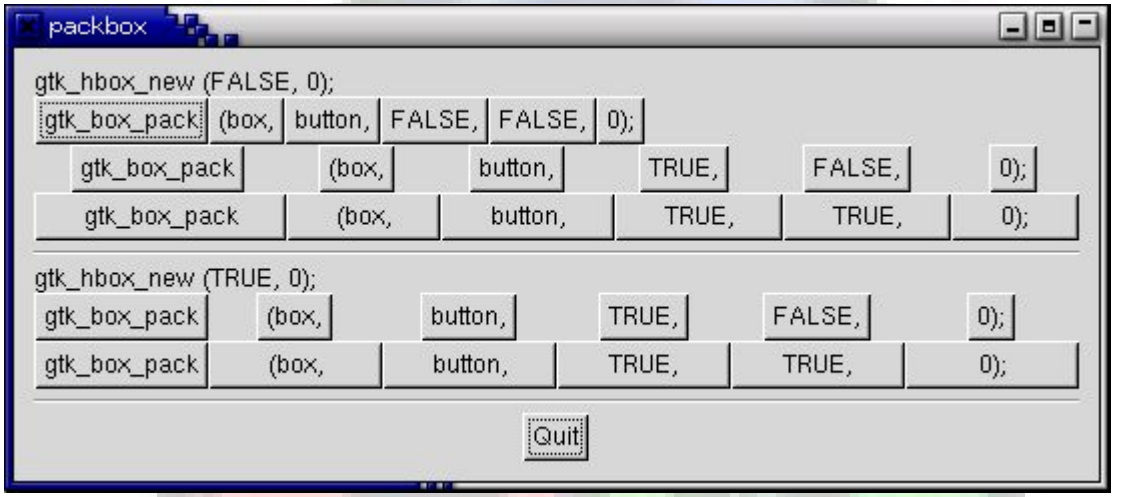
Create Button

Add button to window

Containers in GTK+

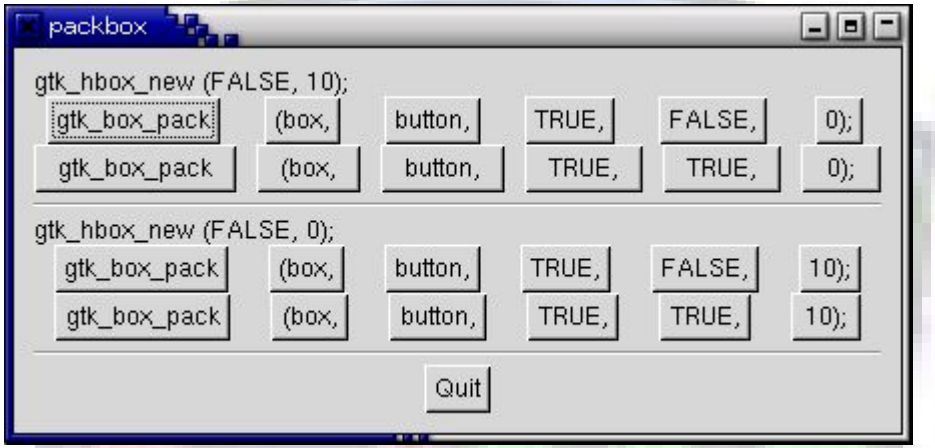
- To put multiple objects in a window a container object is required
- Basic Containers: Horizontal box, Vertical Box and a Table
- Boxes can be packed from the beginning or end
- Tables are done by coordinates
- Lots of flexibility -- leads to some confusion

Layout of Boxes



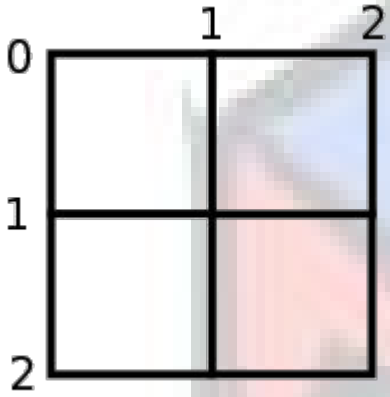
- Three values being changed
  - Homogeneous: make the box take all available space (force expand for all objects)
  - Expand: make objects large enough to use all the space allocated to the box
  - Fill: packed object is allocated space instead of

Spacing the Boxes



- Two forms of spacing (box vs. object)
  - Spacing (box): placed between objects in the box
  - Padding (object): added on either side of the object in the box

Packing Using Tables



- Objects in tables get assigned X and Y ranges

```
gtk_table_attach_defaults(GTK_TABLE(table),
                          button1, 0, 1, 0, 1);

gtk_table_attach_defaults(GTK_TABLE(table),
                          button2, 1, 2, 0, 1);

gtk_table_attach_defaults(GTK_TABLE(table),
                          quit_button, 0, 2, 1, 2);
```

Different Types of Buttons

- Normal buttons (we've done these)
- Toggle Buttons
- Check boxes
- Radio Buttons
- Buttons with graphics (using hbox from before!)

Toggle Buttons

- Toggle buttons preserve their state
- Create: `gtk_toggle_button_new_with_label()`
- Set State: `gtk_toggle_button_set_active(but, TRUE)`
- Callback function: (typical structure)

```
void callback (GtkWidget *widget, gpointer data) {
  if (gtk_toggle_button_get_active(
                        GTK_TOGGLE_BUTTON(widget))) {
      /* button down */
  } else {
      /* button up */
  }}
```

Check Box Buttons

- These are actually subclasses of the toggle buttons! Same functions apply. (Isn't subclassing wonderful)
- Create: `gtk_check_button_new_with_label()`

Radio Buttons

- Radio buttons need to come in sets (so that only one can be active at a time)
- Create: `GtkWidget *gtk_radio_button_new (GSList * group)`
- To get a group: `GSList * gtk_radio_button_get_group (GtkRadioButton * widget)`
- To create a group pass `NULL` to first one
- On selection two events are sent: depressed then pressed

Graphic Button (using an hbox)

```
GtkWidget *box, *label, *image, *button;
  button = gtk_button_new();
  box = gtk_hbox_new(FALSE, 0);
  image = gtk_image_new_from_file(“file.xpm”);
  label = gtk_label_new(“cool button”);
  gtk_box_pack_start(GTK_BOX(box), image,
                     FALSE, FALSE, 3);
  gtk_box_pack_start(GTK_BOX(box), label,
                     FALSE, FALSE, 3);
  gtk_container_add(GTK_CONTAINER(button),
                    box);
  gtk_widget_show(box);
  gtk_widget_show(label);
  gtk_widget_show(image);
  gtk_widget_show(button);
```

Button Object Hierarchy

- GObject
  - GtkObject
    - GtkWidget
      - GtkContainer
        - GtkBin
          - GtkButton
            - GtkToggleButton
              - GtkCheckButton
                - GtkRadioButton
- Small part of the overall GTK+ object hierarchy
- All functions of superclass available on

Using Menus

- Menus can be packed in almost any widget, but they only really make sense in windows
- There are items and shells - shells are lists of items
- Submenus are then created by associating a shell with an item
- Easy to do with ItemFactory (not shown here)

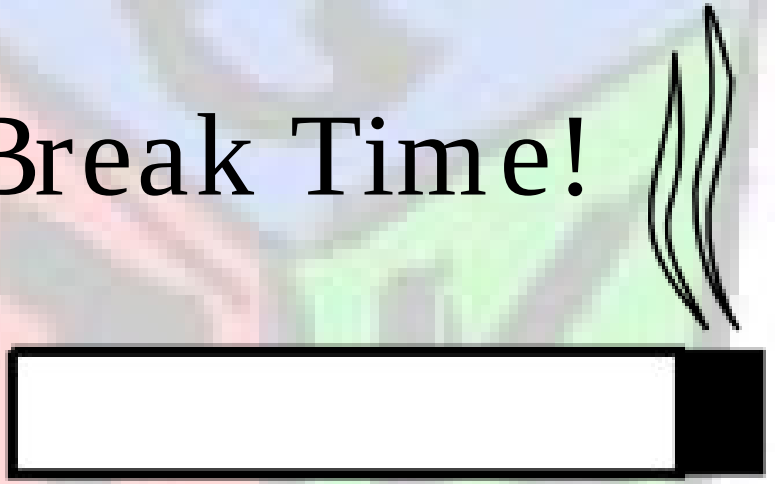
Menu Example

```
menu = gtk_menu_new ();
menuitem = gtk_menu_item_new_with_label("Hey");
gtk_menu_shell_append(
            GTK_MENU_SHELL(menu), menuitem);
root_m = gtk_menu_item_new_with_label("Root");
gtk_menu_item_set_submenu(
                    GTK_MENU_ITEM(root_m), menu);
vbox = gtk_vbox_new (FALSE, 0);
gtk_container_add(GTK_CONTAINER(window), vbox);
menu_bar = gtk_menu_bar_new();
gtk_box_pack_start(GTK_BOX(vbox), menu_bar,
                                FALSE, FALSE, 2);
gtk_menu_shell_append(GTK_MENU_SHELL(menu_bar),
                                        root_m);
gtk_widget_show({menu|menuitem|root_m|menu_bar|
                                    vbox|window});
```

Other things in GTK+

- Scrolled Windows
- Arrows
- Dials and adjustments
- Calendar
- Dialogs
- Text entry boxes

Break Time!



GNOME Libraries of Interest

- ORBit
- Bonobo
- GNOME VFS
- GConf
- Glade
- GStreamer

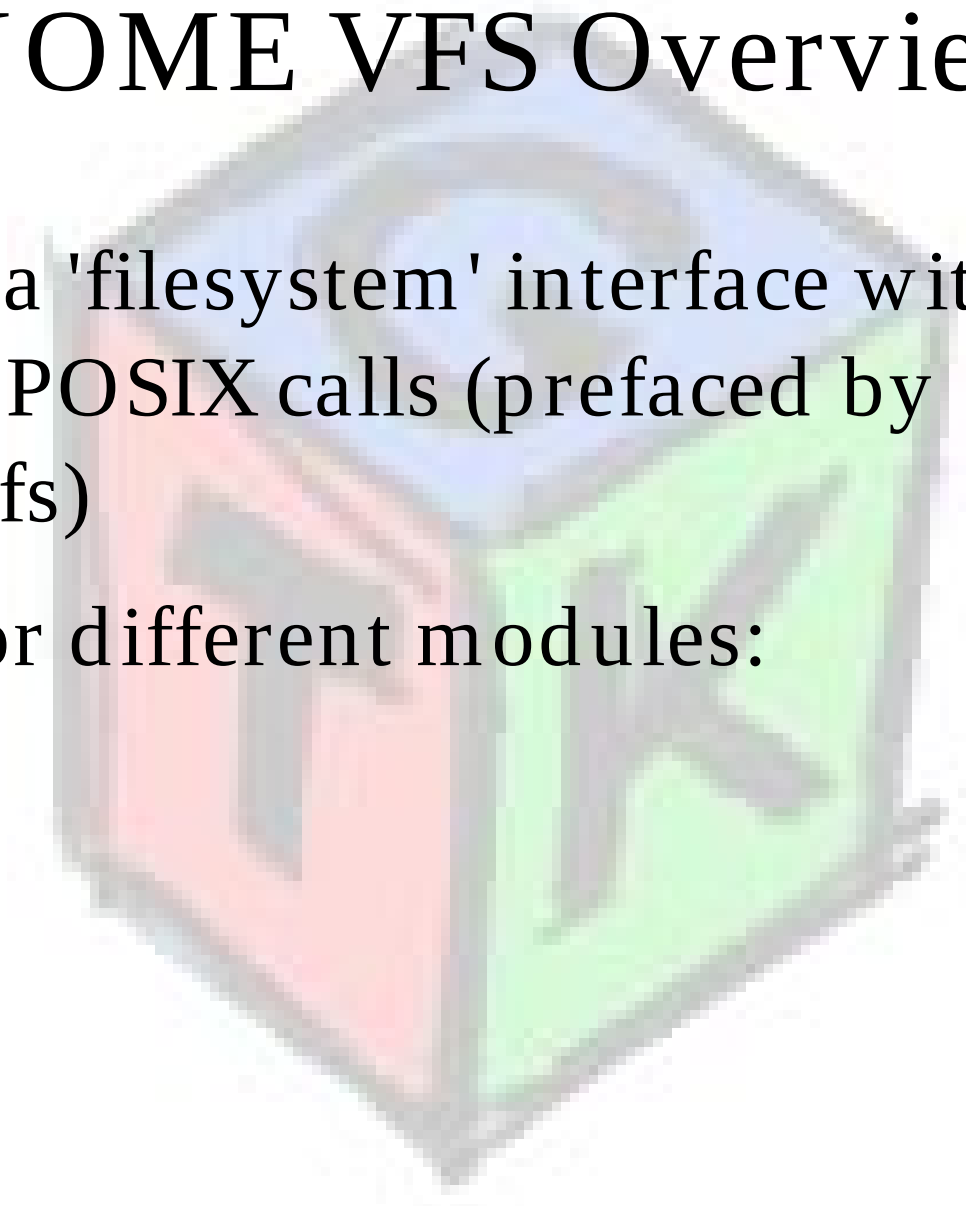
ORBit Overview

- Designed as a fast CORBA implementation
- Provides a C language binding (unusual)
- Provides the basis for Bonobo, Panel, Gconf, CDDB lookup....
- Supposedly as fast as two function calls!
- Allows GNOME applications to be machine independent and still share data

Bonobo Overview

- Named after the Bonobo monkey
- Provides Object Embedding and standard interfaces (similar to MS OLE)
- Common interfaces for objects that are machine and location independent
- Simple interfaces that all inherit from Bonobo::Unknown
- Provides and activation framework that is based on you DISPLAY variable

GNOME VFS Overview



- Provides a 'filesystem' interface with standard POSIX calls (prefaced by gnome_vfs)
- Allows for different modules:
  - burn:/ /
  - nfs:/ /
  - smb:/ /
  - fonts:/ /
- Embedding in all applications allows any

GConf Overview

- Configuration framework similar to Microsoft's Registry (but better, a lot better)
- Global set of defaults
- Per-user settings in $(HOME)/ .gconf
- Allows for schema definitions of variables
- Allows for instant notification of changes for instant apply of variables
- Will be part of the GNOME 2.6 lock down

Glade/ libglade Overview

- Glade is a graphical GUI designer
- Saves design as a XML file
- Can be used to create source code (many lang.)
- Or... can be loaded dynamically using libglade
- Provides a quick and simple way to build GUIs

GStreamer Overview

- Streaming multimedia framework
- Allows for higher level applications to worry about what they really want to do ^_^
- Supports MPEG, MP3, Ogg, Divx, SWF....
- Sources can be any GNOME VFS source (including Internet radio)

References

- GTK site: http:/ / gtk.org/
- GTK tutorial: http:/ / gtk.org/ tutorial/
- GNOME Developer: http:
  / / developer.gnome.org/
- Gstreamer: http:/ / gstreamer.org
- Pango: http:/ / pango.org
- This presentation:
  http:/ / gould.cx/ ted/ projects/
  gtkintro