Beginning GTK+ Programming

Owen Taylor
otaylor@redhat.com

GU4DEC
18 June 2003
Dublin, Ireland
Plan of Tutorial

Basic Terminology
Object Oriented Programming in C
A first program
The Glade GUI builder
Geometry management
libglade

Program organization
GtkDialog
GtkTextView
GtkTreeView
GdkPixbuf
UTF-8, Pango, internationalization
The Libraries

- **GLib** - data structures (hash tables, linked lists), portability, main loop
- **GModule** - dynamic object loading
- **GObject** - object system for C libraries
The Libraries (2)

- **ATK** - connect toolkit to screen readers, etc.
- **Pango** - internationalized text layout, rendering
- **gdk-pixbuf** - Image loading and manipulation
- **GDK** - drawing, connection to window system

Diagram:

- Application
  - GTK+
    - ATK
    - Pango
    - gdk-pixbuf
    - GDK
      - Window System
      - GLib
      - GModule
      - GObject
The Libraries (3)

- **GTK+**
  - Widget system (layout, focusing, key shortcuts, ...)
  - Simple widgets (push buttons, labels)
  - Complex widgets (text editor, tree/list view)
Widgets, Containers

• **Widget** - a control
  – GtkEntry - Single line text entry field
  – GtkButton - Push Button
  – GtkLabel - text label

• **Container** - a widget that contains other widgets
  – GtkWindow - toplevel frame with titlebar
  – GtkHBox - arrange a row of widgets
  – GtkButton - just a container for a GtkLabel
### Addressbook Example

<table>
<thead>
<tr>
<th>Last</th>
<th>First</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doe</td>
<td>John</td>
<td>February 29</td>
</tr>
<tr>
<td>Smith</td>
<td>Jane</td>
<td>December 1</td>
</tr>
<tr>
<td>Taylor</td>
<td>Owen</td>
<td>June 10</td>
</tr>
</tbody>
</table>

#### Doe, John
- **Name:** Doe, John
- **Birthday:** February 29

- **Address:**
  101 Green Knolls Dr
  Middletown, MD 12345
Object Oriented Programming

• "Object" represented by a structure

```c
typedef struct AddressEntry AddressEntry;

struct AddressEntry {
    char *firstname;
    char *lastname;
    int day; /* Birth day */
    int month; /* Birth month */
};
```

• Always use structures by reference (pointer)

```c
AddressEntry *entry = address_entry_new();
```
Constructor

- Allocates memory block, fills in initial field values

```c
AddressEntry *
address_entry_new (void)
{
    AddressEntry *entry;

    entry = g_malloc (sizeof (AddressEntry));
    entry->firstname = g_strdup (""");
    entry->lastname = g_strdup (""");
[...]

    return entry;
}
```
Methods

• Methods are functions
  – Object passed as first argument

```c
void
address_entry_setFirstname (AddressEntry *entry,
                            const char  *firstname)
{
    g_free (entry->firstname);
    entry->firstname = g_strdup (firstname);
}

const char *
address_entry_getFirstname (AddressEntry *entry)
{
    return entry->firstname;
}
```
Memory Management

- How do we know when to free an object?
- Reference counting - keep track of number of parties interested

Diagram:

- Application
- List Widget
- Print Routine

AddressEntry: ref_count: 3
Reference Counting

/* ref_count set to 1 in address_entry_new() */

AddressEntry *entry
address_entry_ref (AddressEntry *entry)
{
    entry->ref_count++;

    return entry;
}

void
address_entry_unref (AddressEntry *entry)
{
    entry->ref_count--;
    if (entry->ref_count == 0)
    {
        /* No longer in use, free contents and entry */
        g_free (entry->firstname);
        [...]
        g_free (entry);
    }
}
Inheritance

- Inheritance done by nesting

```c
struct ModernAddress {
    AddressEntry address;
    char *email;
}

- Same pointer value used for both

address_entry_set_name (entry, "John Doe");
modern_address_set_email ((ModernAddress *)entry,
    "john.doe@acme.com");
```

AddressEntry *address

ModernAddress *modern
Class Structures

- How does address_entry_unref() know how to free entry if it might be an ModernEntry?
- Class structure holds information about the type

```c
struct AddressEntry {
    AddressEntryClass *klass;
    char *firstname;
    [...]  
};
```

```c
struct AddressEntryClass {
    enum { ADDRESS, MODERN_ADDRESS } type;
    void (*free)(Address *entry);
};
```
Class Structures (2)
Cast Macros

- What if we try to use a non-ModernAddress as a ModernAddress? Compiles, but crashes.
- Replace (ModernAddress *) with cast macro
  ```c
  modern_address_set_email (MODERN_ADDRESS (address),
   "john.doe@acme.com");
  ```
- Get a runtime warning
  ```
  ***Warning***: Invalid cast to modern address
  ```
- Run program with --g-fatal-warnings to stop in debugger on failure
#ifdef G_DISABLE_CAST_CHECKS
#define MODERN_ADDRESS(a) ((ModernAddress *)(a))
#else
#define MODERN_ADDRESS(a) (modern_address_check_cast(a))
#endif

ModernAddress *
modern_address_check_cast (AddressEntry *a)
{
    if (a->klass->type == MODERN_ADDRESS)
        return (ModernAddress *)a;
    else
    {
        g_warning ("Invalid cast to ModernAddress");
        return NULL;
    }
}
A First Program

```c
int
main (int argc, char **argv)
{
    GtkWidget *window, *button;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    g_signal_connect (window, "destroy",
        G_CALLBACK (gtk_main_quit), NULL);

    button = gtk_button_new_with_label ("Hello");
    g_signal_connect (button, "clicked",
        G_CALLBACK (hello_clicked), window);

    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show_all (window);

    gtk_main ();

    return 0;
}
```
Compiling GTK+ programs

- pkg-config command line utility gives proper compiler, linker flags.

```bash
$ gcc -o hello `pkg-config --cflags --libs gtk+-2.0`

gcc -o hello -I/usr/include/gtk-2.0
  -I/usr/lib/gtk-2.0/include -I/usr/include/atk-1.0
  -I/usr/include/pango-1.0 -I/usr/X11R6/include
  -I/usr/include/freetype2 -I/usr/include/glib-2.0
  -I/usr/lib/glib-2.0/include -Wl,--export-dynamic
  -lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0 -lm
  -lpangoxtf-1.0 -lpangox-1.0 -lpango-1.0 -lgobject-2.0
  -lgmodule-2.0 -ldl -lglib-2.0
```

- Note backticks ``
OOP in GTK+

- Object represented by a pointer
  
  GtkWidget *window;

- Naming convention for methods:
  
  gtk_window_set_title (GTK_WINDOW (window), ...);
  gtk_container_add (GTK_CONTAINER (window), ...);
  gtk_widget_show_all (window);

- Cast macros

  GTK_WINDOW (window);

- Reference counting - in base GObject class

  g_object_ref (object);
  g_object_unref (object)
Signals

• Notification essential part of GUI
  – Need to find out when, e.g., a button is clicked
• “connect” a callback function to “signal”

```c
void
g_signal_connect (button, "clicked",
  G_CALLBACK (hello_clicked), window);

static void
hello_clicked (GtkWidget *button,
  gpointer    user_data)
{
  GtkWidget *window = user_data;

  g_print ("Hello");
  gtk_widget_destroy (window);
}
```
Signals (2)

- Set of signals associated with each class
  - Button has "pressed", "released", "clicked", ...
  - Also inherits "focus_in_event", "focus_out_event" from GtkWidget, etc, etc.

- Different signals, different signatures:

  ```c
  void clicked_cb (GtkWidget *object, gpointer user_data);
  void row_activated_cb (GtkWidget *object, gpointer user_data);
  ```

- Listed in reference documentation
- User data always last argument
Boolean signal returns

- Some signals have boolean return values
  - Example: "delete_event", emitted when user clicks on close button in title bar:
    ```c
    gboolean delete_event_cb (GtkWidget  *widget,
                             GdkEventAny *event,
                             gpointer       user_data);
    ```
  - Convention is TRUE return stops signal emission
  - TRUE: I handled it, don't do anything more
  - FALSE: I didn't handle it, do whatever you would normally do. (Here, destroy the window)
Main Loop

- Event Driven Programming:
  - After initial setup, program waits for “events” and reacts for them.

- Low level events (button presses, keystrokes) handled by GTK+ and converted into high-level callbacks.

- gtk_main() starts even loop and runs until gtk_main_quit() is called.
Main Loop (2)

- `gtk_main()`
- `g_main_iterate()`
- `gtk_main_quit()`

Actions:
- Dispatch events
- Check for events
GLib Types

- GLib contains various typedefs for integer types:
  - gboolean: FALSE/TRUE boolean
  - guint: Unsigned integer
  - gint32: 32 bit integer
  - gint64: 64 bit integer
- Also contains 'g' names for standard types
  - gint, gdouble, glong, ...
  - These are exactly the same as the standard types
  - Just used for visual consistency
  - No need to cast
Containership

- Container is a widget that contains other widgets
  - Contained widgets are called children
- Heirarchy of containers define layout
Container examples

- **GtkBin** - container with one child
  - **GtkWidget**: toplevel window
  - **GtkButton**: just a container for its label
  - **GtkAlignment**: align, add padding to child

- **Other containers have multiple children**
  - **GtkHBox**: horizontal row of widgets
  - **GtkVBox**: vertical row of widgets
  - **GtkTable**: 2D layout of widgets (like HTML table)
  - **GtkHPaned/GtkVPaned**: two children with user-adjustable division.
Allocation and Requisition

• Requisition
  – Each widget says how much space it needs
  – Parent widget adds up space of children
  – Figure out how much space is needed for toplevel

• Allocation
  – Start from toplevel
  – Each container divides allocated space among its children
Allocation and Requisition (2)
Using Glade/libglade

• Glade
  – GUI editor for visual layouts
  – Saves layouts as XML files
  – Also generates code: DONT USE

• libglade
  – Loads XML layouts into application
Boxes

• Layout out widgets in a line:
  – GtkHBox: Horizontally
  – GtkVBox: Vertically

• Two boolean properties for each child:
  – Expand: If there is excess space allocated to the box, does it go to this child?
  – Fill: Should excess space actually go to the child, or simply be used as padding.
Tables

- GtkTable allows for grid based layout
- Specify child position by left, right, top, bottom edges.
- Also specify options (expand, fill) and padding for each dimension

```c
gtk_table_attach (GTK_TABLE (table), child
    /* x */ 0, 1,
    /* y */ 0, 1,
    /* options */ GTK_EXPAND | GTK_FILL, GTK_FILL,
    /* padding */ 0,
);```
Glade Demo

• Adding widgets ... toplevel window, vbox, labels
• Adjusting properties - label alignment
• GtkScrolledWindow automatically added for GtkTextView
• Selecting widgets with the widget tree
• Setting box spacing for name/widget pairs
• Copying a portion of the widget tree / using the clipboard.
Glade Demo (2)

- Adding an alignment to control extra space
- Adjusting packing properties
- Setting widget names
- Signals and autoconnect
GtkScrolledWindow

- Many widgets allow scrolling larger area
  - GtkTextView
  - GtkTreeView
  - GtkViewport (scroll any widget)
- Any such *scrollable* widget can be put inside GtkScrolledWindow to add scrollbars
- Set policy for each scrollbar:
  - NEVER
  - ALWAYS
  - AUTOMATIC
• Often need to position widget in larger space
• Set how much of available space to use:
  – xscale: 0.0 => 1.0
  – yscale: 0.0 => 1.0
• Where to put the widget in available space
  – xposition: 0.0 => 1.0
  – yposition: 0.0 => 1.0
libglade example

GladeXML *glade;
GtkWidget *main_window;
GtkWidget *address_list;

/* Create dialog from XML file */
glade = glade_xml_new ("addressbook.glade",
          "addressbook-window",
          NULL); /* translation domain */

/* Extract widgets for future reference */
main_window = glade_xml_get_widget (glade, "addressbook-window");
address_list = glade_xml_get_widget (glade, "addressbook-treeview");

/* Get rid of file object */
g_object_unref (G_OBJECT (glade));
Autoconnect

• Can specify callback names in Glade, have libglade look them them up in program

GladeXML *glade;

    glade = glade_xml_new ("addressbook.glade",
        "addressbook-window",
        NULL);

    glade_xml_signal_autoconnect (glade);

• Functions need to be exported:

  - Have to be public, not static
  - pkg-config --libs libglade gives proper linker flags
Part II

10 Minute Break

Program Organization
GtkDialog
GtkTextView
GtkTreeView
Organizing a Program

• Base around application data structures

    typedef struct AddressBook AddressBook;
    struct AddressBook {
        gchar *filename;
        GtkWidget *name_entry;
        GtkWidget *address_text_view;
    };

• Often have one structure for toplevel window
Object Data

• Way of attaching application data to a widget
• String key identifies each piece of data
• Store data:
  
g_object_set_data (G_OBJECT (window),
  "address-book", address_book);
• Get data:
  
  address_book = g_object_get_data (G_OBJECT (window),
  "address-book");
• Storing object data on top-level widget avoids having to store it on each individual widget:

```c
AddressBook *
get_address_book (GtkWidget *widget)
{
    GtkWidget *toplevel;
    toplevel = gtk_widget_get_toplevel (widget);
    return g_object_get_data (toplevel, "address-book");
}
```
GtkDialog

- Window widget with buttons
- Used for temporary interaction
- Convenience: `gtk_dialog_run()`
  - Starts main loop, waits until button pressed
  - Returns `response id` for chosen button
GtkWidget *dialog;
int response;

dialog =
gtk_dialog_new_with_buttons ("Print document", parent_window,
GTK_DIALOG_DESTROY_WITH_PARENT,
"Print", GTK_RESPONSE_OK,
GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
NULL);

/* Add contents to dialog */

response = gtk_dialog_run (GTK_DIALOG (dialog));

if (response == GTK_RESPONSE_OK)
{
    /* Print */
}

gtk_widget_destroy (dialog);
GtkMessageDialog

• GtkDialog that just holds a message

```c
gtk_message_dialog_new (parent_window,
    GTK_DIALOGDESTROY_WITH_PARENT,
    GTK_MESSAGE_WARNING,
    GTK_BUTTONSONOK,
    "Error when printing: %s",
    error->message);
```

• Trap: string is format string
  – Arbitrary message string might contain '%'
  – So, use "%s", message, not "message"
Model/View

- Widgets so far are small and simple
- GTK+ contains two widget for display of large amounts of information
  - GtkTextView: multiple line text display
  - GtkTreeView: trees and lists (a list is just a flat tree)
- Split apart widget (view) from data store (model)
  - GtkTextView <=> GtkTextBuffer
  - GtkTreeView <=> GtkTreeModel (GtkListStore, GtkTreeStore, ...)
  - Can have multiple views of same model
GtkWidget *view;
GtkTextBuffer *buffer;
GtkTextIter start, end;
char *text;

view = gtk_text_view_new (NULL); /* NULL - create new buffer */
buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (view));

/* Set text */
gtk_text_buffer_set_text (buffer, "Some Text");

/* Get text */
gtk_text_buffer_get_start_iter (buffer, &start);
gtk_text_buffer_get_end_iter (buffer, &end);

text = gtk_text_buffer_get_text (buffer,
                                   &start, &end, /* range */
                                   FALSE); /* include invisible? */
g_print ("The text is %s\n", text);
g_free (text);
Iterators

• Refer to a track of a place in data (GtkTextBuffer)
• Methods to navigate (iterate) through the data
  
  ```
  GtkTextIter iter;
  gtk_text_buffer_get_start_iter (buffer, &iter);
  gtk_text_iter_forward_line (&iter);
  ```

• Temporary
  
  – Changes in buffer invalidate iter
  – Use GtkTextMark for permanent “bookmark”
Object Properties

- Properties are attributes of object

- Set:
  ```c
  g_object_set (entry,
    "text", "Hello World",
    "cursor_position", 3,
    NULL);
  ```

- Get:
  ```c
  g_object_get (entry
    "text", &text,
    "cursor_position", &cursor_position,
    NULL);
  ```
Interfaces

• Sometimes inheritance not enough
• Example from GTK+-2.4:
  – GtkFileChooserDialog is both a GtkWidget and a
    GtkWidget
  – Inherits from GtkWidget
  – Has GtkWidget as an interface
• Cast macros in same way as inheritance:
  
  GtkWidget *chooser = GTK_FILE_CHOOSER (dialog);
Introducing GtkTreeView

- Handles both lists and trees
- Model/View: Data stored separately from widget
- GtkTreeView: The widget
- GtkTreeModel: data interface
  - GtkListStore: Flat data
  - GtkTreeStore: Hierarchical data
  - Can also create custom models (but difficult)
GtkTreeView Column

- Lists and trees have multiple columns of information
- Each column drawn by one or more renderers

<table>
<thead>
<tr>
<th>Country</th>
<th>Representative</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>J. Dupont</td>
</tr>
<tr>
<td>Italy</td>
<td>F. Smith</td>
</tr>
<tr>
<td>Japan</td>
<td>Tanaka A.</td>
</tr>
</tbody>
</table>

- `GtkCellRendererText` for text columns
- `GtkCellRendererPixbuf` for image columns
Cell Renderer Properties

- Properties for each cell renderer are set, row by row, from the GtkTreeModel

Properties

- "text": G_TYPE_STRING
  - J. Dupont: black
  - F. Smith: black
  - Tanaka A.: red

- "foreground": G_TYPE_STRING
  - J. Dupont: black
  - F. Smith: black
  - Tanaka A.: red

- "strikethrough": G_TYPE_BOOLEAN
  - J. Dupont: FALSE
  - F. Smith: TRUE
  - Tanaka A.: FALSE
GtkTreeModel

- Each GtkTreeModel has some number of columns, each with a type
  - `G_TYPE_STRING`: a string
  - `G_TYPE_BOOLEAN`: boolean
  - `G_TYPE_INTEGER`: integer
- Every row in model stores one item in each column
- GtkTreeIter points to a single row in model
enum {
    NAME_COLUMN,
    COLOR_COLUMN,
    STRIKETHROUGH_COLUMN
};

GtkWidget *list_store;
GtkTreeIter iter;

list_store = gtk_list_store_new (3, /* number of columns,
    G_TYPE_STRING,
    G_TYPE_STRING,
    G_TYPE_BOOLEAN);

/* Append a row, iter is set to point to it */
gtk_list_store_append (list_store, &iter);

gtk_list_store_set (list_store, &iter,
    NAME_COLUMN, "Tanaka A.",
    COLOR_COLUMN, "red",
    STRIKETHROUGH_COLUMN, FALSE,
    -1);  /* Terminates arguments */
Data Functions

- Can use *data function* to set all properties from a single column

```
MY_TYPE_REPRESENTATIVE
<representative1>
<representative2>
<representative3>
```

```
J. Dupont
F. Smith
Tanaka A.
```
static void
name_data_func (GtkTreeViewColumn *tree_column,
    GtkWidgetRenderer   *cell,
    GtkWidgetModel   *tree_model,
    GtkWidgetIter     *iter,
    gpointer          data)
{
    Representative *rep;

gtk_tree_model_get (tree_model, iter,
    REPRESENTATIVE_COLUMN, &rep,
    -1);

g_object_set (cell,
    "text", representative_get_name (rep),
    "strikethrough", !representative_made_quota (rep),
    NULL);

    representative_unref (rep);
}
Boxed Types

• Need to tell GTK+ about AddressEntry in order to use in a GtkListStore

#define ADDRESS_TYPE_ENTRY (address_entry_get_type ()
GType
address_entry_get_type (void)
{
    static GType our_type = 0;
    if (our_type == 0)
        our_type = g_boxed_type_register_static ("AddressEntry",
            (GBoxedCopyFunc) address_entry_ref,
            (GBoxedFreeFunc) address_entry_unref);

    return our_type;
}
GtkTreeSelection

- Auxiliary object representing current selection
- Has:
  - Methods for retrieving currently selected rows
  - Signal "selection_changed" for reacting to changes
  - Selection mode setting
    (gtk_tree_selection_set_mode())
  - GTK_SELECTION_SINGLE: 0 or 1 row selected
  - GTK_SELECTION_BROWSE: 1 row selected
  - GTK_SELECTION_MULTIPLE: 0-N rows selected
GtkTreeSelection Example

GtkTreeSelection *selection;

selection = gtk_tree_view_get_selection (treeview);
g_signal_connect (selection, "changed",
               G_CALLBACK (selection_changed), book);

static void
selection_changed (GtkTreeSelection *selection,
                      AddressBook    *book)
{
    GtkWidget *model;
    AddressEntry *entry;
    GtkWidget iter;

    if (gtk_tree_selection_get_selected (selection, &model, &iter))
    {
        gtk_tree_model_get (model, &iter, 0, &entry, -1);
        /* do something with entry */
        address_entry_unref (entry);
    }
}
GtkTreeView Sorting

• Idea: User clicks on column, sort on that column
• But how does GtkTreeView now how to sort on a view column?
  – Might have multiple renderers, custom renderer, etc.
• Idea application sets *sort column ID* for each column in the view.
• Default action for sort column ID N is to sort on *model column N*
• Possible to also set custom sorting for a sort ID (useful for a data func)
Sorting Example

gtk_tree_sortable_set_sort_func (GTK_TREE_SORTABLE (list_store),
SORT_BIRTHDAY,
compare_birthday, NULL, NULL);

int
compare_firstname (GtkTreeModel *model,
GtkTreeIter *a, GtkTreeIter *b,
gpointer user_data)
{
    AddressEntry *entry_a, *entry_b;
    int result;

gtk_tree_model_get (model, a, 0, &entry_a, -1);
gtk_tree_model_get (model, b, 0, &entry_b, -1);

    result = strcmp (entry_a->firstname, entry_b->firstname);

    address_entry_unref (entry_a);
    address_entry_unref (entry_b);

    return result;
}
GdkPixbuf

- GdkPixbuf represents an image
- Can load many image types
  - PNG, JPEG, GIF, TIFF, BMP, ICO, TGA, ANI, XBM, XPM, WBMP, PCX, RAS
  - Also extensible: librsvg installs SVG loader
- Direct client side pixel access. Compare:
  - GdkPixbuf - client-side image, directly manipulate pixels
  - GdkPixmap - server-side image, GDK drawing primitives, no direct pixel access
GdkPixbuf example

GdkPixbuf *pixbuf;
GError *error = NULL;

pixbuf = gdk_pixbuf_new_from_file ("my.png", &error);
if (!pixbuf)
{
    g_printerr ("Unable to load image: %s\n",
                error->message);
    g_error_free (error);
}
else
{
    /* use pixbuf */

    g_object_unref (pixbuf);
}
Unicode

- Character set including characters for almost all languages in the world
- Standardized as ISO-10646
- All text in GTK+ is in Unicode
UTF-8

• Easiest encoding of Unicode is 4 bytes per character ... inefficient.

• Instead use multibyte encoding:
  – Each character one or more bytes
    
    0xxxxxxxx
    110xxxx  10xxxxxx
    1110xxxx  10xxxxxx  10xxxxxx
    11110xxx  10xxxxxx  10xxxxxx  10xxxxxx

• ASCII compatible
Pango

• Text layout engine
  - Input: Unicode text
  - Output: positioned *glyphs*

• Handles most of the world's languages
  - Right to left text (and mixed LTR/RTL)
  - Complex joins, ligatures

• Hides the complexity from application developer
PangoLayout

• Holds a paragraph of text

```c
PangoLayout *layout;
int width, height;
layout = gtk_widget_create_pango_layout (widget, "Hi There");
pango_layout_get_pixel_size (layout,
    &width, &height);

gdk_draw_layout (widget->window,
    widget->style->black_gc,
    x, y,
    layout);
g_object_unref (layout);
```
Pango Markup

- Usually don't use Pango directly; hidden by widgets.
- Mini-XML-like language allows using more features of Pango

```c
const char *markup = "This is <big>big text</big>";
gtk_label_set_markup (label, markup);
```
GList

GList *books;

- Linked list of data items
  - List represented by pointer to first item
  - Empty list represented by NULL
GList example

static GList *books;

books = g_list_prepend (books, book);

[...]

books = g_list_remove (books, book);
if (books == NULL)
    gtk_main_quit ();
More information:

• These slides:

• Main GTK+ site:
  http://www.gtk.org

• API documentation
  http://developer.gnome.org/doc/API/