EggMenu

James Henstridge

<james@daa.com.au>

EggMenu is a new menu and toolbar API being developed for inclusion in GTK 2.4. It aims to provide a simple but powerful API that can be used by both simple and complex applications.

The API is also designed to be extensible, so that it can be used in component systems like Bonobo, and extended to handle new types of menu items or toolbar items.

Table of Contents

Gnome Development Process

The methodology used to extend the Gnome Development Platform has changed a lot since the inception of the project. Whereas almost anything could be added to the platform in the early days, we now have a more structured process that allows new APIs to be tested and reviewed before migrating into the development platform.

History

In the pre-1.0 days of Gnome, the evolution of the development platform was driven by what individual hackers were interested in. While this lead to a lot of good code that formed the foundation of early versions of Gnome, it also produced a fair amount of poor quality widgets and APIs.

This was not a large problem back then because the APIs could be fixed as easily as they had been added. As the Gnome platform stabilised, we had less and less opportunity to fix mistakes in the API.

Luckily, a lot of the immature or poor quality APIs were removed or fixed before the Gnome 1.0 release. There was another round of janitorial work for the Gnome 2.0 development platform release, but the job certainly isn't over.

Present

Because of Gnome's guarantees of stability in the development platform, it makes a lot of sense to make sure that immature or badly designed interfaces are not added to stable libraries. Things that go in the stable development library will generally need to stay there for a number of years, which can be a big problem for some APIs.

Since mature, stable interfaces do not spring out of nothing, we needed a process to allow new code to be developed, reviewed and tested in applications. This has included the introduction of the GEP process, and the testing of new code in unstable prototype libraries.

GEP stands for "Gnome Enhancement Proposal". The process is modelled after "Python Enhancement Proposals" (PEPs) and "TCL Improvement Proposals" (TIPs). This involves writing up a document detailing the requirements of an interface and posting it for public review. People who are interested provide feedback which is used to update the GEP. This document is then used as a guide for development of the implementation. Further updates can be made as the code matures.

The `libegg` library has also been created to prototype code and interfaces that will move to the development platform when mature. The guidelines for inclusion in libegg are designed to encourage the maturation of the code:

- All features will move down to a library in the stable development platform when ready. The code must only depend on existing dependencies of the target platform library.
- All features must be "sponsored" by the maintainer of the target library. Features without support do not go in `libegg`.
- As soon as the code has been shipped in the stable development platform, it gets removed from `libegg`.
- Interfaces use an `egg_` prefix, rather than the one of the target platform library. When the code is moved, the functions get renamed.
- The API and/or ABI of `libegg` can change at any point.
- All of `libegg` will always depend on the latest stable branches of platform libraries, rather than the development branches.

EggMenu

Development of some new features targeted at GTK and Gnome 2.4 are currently moving through this process. This paper is about a new menu and toolbar API called *EggMenu*.

Existing Menu/Toolbar API

When writing applications against the existing menu API in GTK, the code would follow the structure of your menus. For example, the code may be structured as follows:

- create `GtkMenuBar`.
- create “File” `GtkMenuItem`.
- create a `GtkMenu` and attach it to the “File” menu item.
- create menu items and add them to the “File” `GtkMenu`.
- attach callbacks to each menu item

The `GtkItemFactory` interface is also available that streamlines the above steps, but it is essentially the same procedure. Building toolbars is a similar process.

This setup has a number of problems though:

- How do I disable the “Save” item? Often a user interface will have multiple ways to perform an action (menu item, toolbar button, etc). If you want to disable the action, you will want to disable every widget that can be used to perform the action.
- Rearranging menus is fairly involved, as it requires code modifications. This is a problem if you want to support rearranging menus.

The libbonoboui library provides many of these features but has a few disadvantages:

- Requires the application to use CORBA, which puts some developers off.
- API is quite different to the GTK one, which means that developers wishing to move their applications from GTK only to GNOME will need to rewrite their menu code. This is even more of a problem for developers who wish to support both GNOME and GTK only compiles (possibly for Windows support).

EggMenu is intended as an API that provides the features needed by advanced applications, only depends on other parts of GTK, and should be extensible enough to be usable by component systems like Bonobo.

Actions

Actions are one of the core concepts of *EggMenu*. An action is an object that bundles the following:

- a name
- a menu label
- a toolbar label (optional)
- an icon
- a keyboard shortcut
- a callback
- state (sensitive, visible, etc)

An action essentially represents something that the user can perform, along with some information about how it should be presented in the interface. Interfaces are provided to create menu items and toolbar items:

```
#include <libegg/menu/egg-action.h>

GtkWidget *egg_action_create_menu_item   (EggAction *action);
GtkWidget *egg_action_create_tool_item   (EggAction *action);
```

A program can create any number of menu items and toolbar items for an action. The menu/toolbar item will mirror the label, icon, keyboard shortcut and state of the action. When you change any of the properties on the action, the menu/toolbar items will change to match. Activating any of the menu/toolbar items will execute the callbacks attached to the action.

This provides a simple way of disabling or enabling an action, as described in the previous section. By changing the sensitive state of the action, all the related menu/toolbar items will be disabled to match.

As with menu items and toolbar items, *EggMenu* supports multiple types of actions. At the moment the following types are included in the library:

`EggAction`	roughly equivalent to the classic `GtkMenuItem`.
`EggToggleAction`	equivalent to a `GtkCheckMenuItem`. Has an "active" state specifying whether the action has been checked or not.

`EggRadioAction` Similar to `GtkRadioMenuItem`. A number of `EggRadioAction` actions can be linked together so that only one may be active at any one time.

An application can easily implement new types of actions. For instance, a word processor might want to implement a font selection action that displays as a drop down list on the toolbar, and a menu item that pops up a font selector when on a menu.

Action Groups

Most actions in an application should only be available to the user in certain situations. Some possible ways to categorise actions include:

- Global actions that should be available in every context, such as "Quit", "New" and "Open".
- Actions that act on a particular document in a multiple document application. For instance, "Save".
- Actions that are only valid in particular editing modes. For instance, a word processor may have a set of commands that can only be performed when editing a table.

EggMenu uses `EggActionGroup` objects to hold related actions. The action group acts like a dictionary mapping action names to the action objects themselves. The API for using an action group is as follows:

```
EggActionGroup *egg_action_group_new           (const gchar *name);

const gchar    *egg_action_group_get_name      (EggActionGroup *action_group);
EggAction      *egg_action_group_get_action    (EggActionGroup *action_group,
                                                 const gchar *action_name);
GList          *egg_action_group_list_actions  (EggActionGroup *action_group);
void            egg_action_group_add_action    (EggActionGroup *action_group,
                                                 EggAction *action);
void            egg_action_group_remove_action (EggActionGroup *action_group,
                                                 EggAction *action);
```

Action groups are used to add and remove sets of user actions from the user interface when used with the menu merging system.

UI Merging

The concept of actions in a toolkit is quite useful in its own right. When combined with a menu and toolbar merging system, we get a very powerful API.

Menu merging is handled by the `EggMenuMerge` object. This class holds a tree of nodes that represent the menus and toolbars of the application. XML files can be loaded and unloaded to add and remove items from the user interface.

UI XML Files

The XML files used by *EggMenu* use a subset of the Bonobo UI format. An example UI file is shown below:

```
<?xml version="1.0"?>
<Root>
  <menu>
    <submenu name="FileMenu" verb="StockFileMenuAction">
      <menuitem name="Open" />
    </submenu>
    <submenu name="HelpMenu">
      <menuitem name="About" />
    </submenu>
  </menu>
</Root>
```

When the above XML file is loaded by a `EggMenuMerge` object, the resulting tree will look a lot like the DOM representation of the tree. Each node in the tree is assigned a name according to the following rules:

1. If the element has a name attribute, it will be used as the node's name.
2. otherwise, the element name (Root, menu, submenu, etc) will be used.

Menu items, menus and toolbar items also have an action associated with them, which is given in the verb attribute. If no verb attribute is given, then the node's name is used instead. The action is used to work out what label and icon the menu item should have, along with handling callbacks when the menu item is selected.

After loading the XML file into the `EggMenuMerge` object, an idle function is queued to actually build the menus. This is where the actual widgets get created. The action names are looked up in one or more `EggActionGroup`'s associated with the `EggMenuMerge` object.

Merging Multiple UI Files

When multiple XML files are loaded, more than one file may reference a particular node name. In this case, `EggMenuMerge` makes note of this (to aid in demerging part of the UI), and the top most action will be used.

To get a better feeling for how merging works, consider the following two trees:

Figure 1. file1.ui

```
Root
    menu
        submenu: FileMenu
            menuitem: Open
        placeholder: TestPlaceholder
        submenu: HelpMenu
            menuitem: About
    dockitem: toolbar1
        toolitem: NewButton
```

Figure 2. file2.ui

```
Root
    menu
        submenu: FileMenu
            separator
            menuitem: Quit
        placeholder: TestPlaceholder
            submenu: EditMenu
                menuitem: Cut
    dockitem: toolbar1
        toolitem: OpenButton
```

When `file2.ui` is merged on top of `file1.ui`, we get the following node tree:

Figure 3. Merged UI

```
Root
    menu
        submenu: FileMenu
            menuitem: Open
            separator
            menuitem: Quit
        placeholder: TestPlaceholder
            submenu: EditMenu
                menuitem: Cut
        submenu: HelpMenu
            menuitem: About
    dockitem: toolbar1
        toolitem: NewButton
        toolitem: OpenButton
```

The important things to note here are:

- New nodes are appended to the end of their parent node. This will often result in different merged menus if you perform the merge in a different order. In practise, this isn't much of a problem, since most apps have a base menu layout and then want to merge something on top.
- Placeholders are another type of container. They are used in cases where simply appending menu items does not give the desired layout. For the purposes of menu merging, placeholders are treated as submenus, except that the child nodes are displayed inside the placeholder's parent.

Demerging UI Files

Demerging is a simpler process than merging. While merging, each node in the `EggMenuMerge` object is tagged with the UI files that it was referenced by. The demerge process goes something like this:

1. All nodes in the tree are iterated over, and if they are tagged by the target UI file, then that tag is removed.
2. If a node is no longer tagged by any UI files, or the action name it uses changes, then the node is marked as dirty.

An idle is queued to do the final cleanup. For any node that is no longer referenced by any UI file, the corresponding widget is destroyed, and the node is deleted. For nodes that have changed their action, the widget is updated to reflect the change.

An idle function is used to reduce the number of changes when the application removes one UI file and adds another one to the user interface.

The Future

There is still work to be done on *EggMenu*. Some developers are experimenting with the code in larger applications (such as Mr Project), which will helped highlight portions of the API that need work.

Another feature that has been requested but not yet implemented is the ability to dynamically add a menu item to the user interface without having to generate an XML string (which is what Bonobo requires you to do). The proposed method of handling this is:

1. Call an API to create a merge "tag". This tag would be similar to the ones created when loading a UI file, and could be used to demerge the created menu items later.
2. Call an function that creates a named node using a particular action, referenced by the previously created tag.

This provides a convenient API for creating dynamic entries, while allowing them to be handled the same as any other UI file (for the purposes of merging, demerging, etc).

EggMenu will also need to go through the GEP standardisation process as part of consideration for inclusion in GTK 2.4.

Related Links

Source code	http://cvs.gnome.org/lxr/source/libegg/libegg/menu/
GEP List	http://developer.gnome.org/gep/list.html