Pardus Notification Manager: A Google Summer of Code 2008 Project

Documentation

Mehmet Ozan Kabak [1]
e-mail: wanderer2@gmail.com

August 2008

[1]I would like to thank my mentor Gökmen Göksel for his continuous support over the summer.

Contents

Chapter 1

Pardus Notification Manager Documentation

1.1 Introduction to *Pardus Notification Manager*

Pardus Notification Manager project aims to develop a software suite that includes the necessary programs that make up a user interface framework for displaying notifications and feeding the user response back to the event origin (source of the notification). If found successful, *Pardus Notification Manager* will be used as the main notification framework of the Pardus GNU/Linux distribution. The entire software is developed using Python programming language and is written from scratch. The software suite consists of four main modules, which are:

1. D-Bus Listener Module (*notman*)
2. Notification Displayer (GUI) Module (*notdisplayer*)
3. Notification Requesting Client Module (*notclient*)
4. Configuration Module (*config*)

Each of these modules will be explained in the following subsections.

1.1.1 D-Bus Listener Module

Simply put, this module is a program that connects to the session bus and exports a well-defined interface to it. When this module receives a notification display request, it tries to display the requested notification right away by forwarding the request to the notification displayer module (see section 1.1.2). If it is not possible to fullfill the request right away, the request is put in a notification queue. This queue is checked periodically and any pending requests are processed in a FIFO manner. The D-Bus Listener Module has a short lifespan (default is 15 seconds) and it shuts itself down if no notification request arrives in its lifespan. If a notification arrives when the

program is not running, the session bus executes the program automatically. This approach uses the system resources more conservatively than traditional daemon-based approaches. This module also conveys user's response to a notification back to its sender. The response can be delivered both synchronously and asynchronously.

1.1.2 Notification Displayer (GUI) Module

This module is a Qt thread which displays notifications in small windows. These windows are fully skinnable. Any UI file containing some basic elements (closing button, notification text area etc.) can be used to skin the notification window. Details on the skinning procedure is given in section 1.4. Apart from the skin, geometrical properties of the notification windows are also variable and can be set by the user by using the configuration tool (more on this in section 1.1.4). This module uses pure Qt calls (PyQt4.4 in particular) and does not require PyKDE. The notification windows can include buttons by which the user can respond to the notification. The user's response is forwarded back to the D-Bus Listener module.

1.1.3 Notification Requesting Client Module

Unlike the other modules, this module is not used by the notification manager itself. This module will be imported by the program that wants to display a notification to the user. This module presents a simple object to the programmer. By calling appropriate methods of the presented object, the programmer can display notifications without dealing with the D-Bus itself. If the programmer uses a main loop (GLib and Qt main loops are supported), he/she can receive notification responses both synchronously and asynchronously. This module wraps implementation details of the response mechanism and presents a simple interface to the programmer.

1.1.4 Configuration Module

This module is a stand alone GUI program by which the user can configure the notification manager. The program edits the configuration file (/.notify/config.xml). The program forces the user to enter valid data by doing XML Schema (XSD) checks. The configuration file is read by the D-Bus Listener Module and the users preferences are reflected to the program behavior.

1.2 Installing *Pardus Notification Manager*

1.2.1 Dependencies

Pardus Notification Manager mostly makes use of standard python libraries. Two non-standard libraries that are required are the following:

1. lxml: XML processing library
2. dbus: D-Bus python bindings

1.2.2 Installing Instructions

The suite is packaged by the python distutils module. To setup and install the module, one can use the following commands:

```
user@host:/path/to/notifier$ ./setup.py build
user@host:/path/to/notifier$ ./setup.py install
```

Similarly, to uninstall the module issue the following command:

```
user@host:/path/to/notifier$ ./setup.py uninstall$
```

All `.py` files are copied to the system-default directory. Other required data (icons, sample config file etc.) are copied to /usr/share/pnm. When the notification manager module runs for the first time, it creates a user-specific config file by copying the sample config file to `~/.notify`. In subsequent runs, configuration variables are read from this file.

1.3 How to send notifications from your own program

One needs to import the *notclient* module (introduced in section 1.1.3) in order to send notifications. This is done by a statement similar to:

```
from pnm.notclient import *
```

After importing the module, a *Notifier* object needs to be created. Constructor of the *Notifier* object takes one optional parameter which specifies the type of the main loop used by your program. You don't have to supply this argument, but asynchronous callbacks will not work without this argument. In summary, the *Notifier* object can be constructed by one of these expressions:

```
no_async_support = Notifier()
my_program_uses_qt = Notifier("qt")
glib_rules = Notifier("glib")
```

This *Notifier* object provides the following methods:

1. `Echo(message_string)`: A diagnostic method, which asks the *notman* module to echo the message given as argument.
2. `SendExitSignal()`: Forces the *notman* module to shutdown.

3. `SendNotification(ntf_obj, [rply_callback], [error_callback])`: This method is the one that does the heavylifting. It takes a *Notification* object as its main argument and sends it to the *notman* module to be displayed. In blocking (synchronous) mode, this method returns the notification object as modified by the *notman* module. This modified version contains the (possible) user response. `rply_callback` and `error_callback` are optional arguments for non-blocking (asynchronous) mode. These are ordinary Python callables with one argument which is the notification instance as modified by the *notman* module. They are called when the *notman* module displays the notification succesfully.

Notification objects are created by supplying three optional arguments to its constructor. If these arguments are not supplied, default values are used in their place. The following expression summarizes the usage:

```
my_notification = Notification([title], [text], [icon_path])
```

Buttons can be added to the notification window by the `AddButton()` method of the *Notification* object. This method takes two arguments which are the button text and the button icon path. Note that the icon path is optional. Each button is identified with an index. The first button you add has the index 0, the second has 1 etc. The following can be given as an example:

```
my_notification.AddButton(button_text, [button_icon_path])
```

Notification objects have the field `isReceived` which is initially `False`. When the *notman* module displays the notification successfully, it returns the same object with the field `isReceived` set to `True`. The index of the chosen button is written in the field `chosen_button`.

The following demonstrates how to create a *Notification* object and sending it:

```
nt = Notifier()
this_notification = Notification("hi guys", "whats crackin?")
this_notification.SetNotificationIcon("/path/to/kewl/icon")
this_notification.AddButton("i'm doin good")
this_notification.AddButton("not much dude")
result = nt.SendNotification(this_notification)
if result.isReceived == True:
        # Do something with result.chosen_button
```

1.4 Skinning *Pardus Notification Manager*

1.4.1 How to apply a skin file

Pardus Notification Manager can use `.ui` files as skins. To choose a skin, use the configuration module (*config*) and choose the file as in the relevant dialog box. The *config* module will try to

create the corresponding `.py` and `.pyc` files and copy them to `~/.notify/`. If no error occurs after saving the config file, the *notman* module will use the skin file the next time it runs.

1.4.2 How to create skin files

You can use Qt Designer to create skin files for *Pardus Notification Manager*. You can add extra components to the notification window or remove components you find unnecessary. The *notman* module only assumes the existence of five mandatory components:

1. `exit_button`: A Qt button which will serve as the closing button of the notification window.
2. `notification_picture`: A Qt label which is used for displaying notification icons.
3. `notification_title`: A Qt label which is used for displaying notification titles.
4. `notification_text`: A Qt label which is used for displaying notification text.
5. `gridlayout2`: A Qt grid layout which contains responding buttons.

As long as you provide these mandatory components, your `.ui` file should work fine.

Chapter 2

Configuring *Pardus Notification Manager*

2.1 Configuration Variables

Configuration module (*config*) allows the user to change a number of configuration variables. Although most are self explanatory, I will explain all below for the sake of completeness:

2.1.1 Notification Window Geometry

You can set the height and the width of notification windows (in percent) by entering appropriate values into relevant textboxes.

2.1.2 Notification Manager Options

There is only one variable here: the notification manager lifetime. The *notman* module runs for this long (in seconds) before shutting down.

2.1.3 Notification Window Stacking Direction

You can choose up or down as the stacking direction.

2.1.4 Skin

You can enter (or browse for) the path of a compliant `.ui` file to be used as the notification window skin.

2.1.5 Notification Window Pop-up Position

There are three options here. You can choose to have notification windows pop-up from the lower right of the screen, the upper right of the screen or a manual place. If you choose manual, a dummy window appears. You can use this dummy window to assign a manual location to your notification windows.

2.1.6 Animation Parameters

You can set three animation options here. *Total animation time* refers to the time interval in which notification windows move to take appropriate places. *Time quanta* is the frame time: Notification windows move one step at each *time quanta*. *Notification window lifetime* refers to the amount of time for which the notification windows stay on screen. All these values are in milliseconds.