

Développer des applications win32 avec python et glade

Auteur: Nzeka Gilbert

E-mail de l'auteur: khaalel@gmail.com

Site web de l'auteur: www.nzeka-labs.com, <http://korriban-planet.blogspot.com/>

Document Version: 1.0 (article traduit de l'anglais)

Version Originale: <http://pygtk.org/articles/bitpodder/BitPodder.htm>

Index

- I] License**
- II] But de l'article**
- III] Ce que vous devez avoir avant de commencer...**
- IV] Qu'est-ce que PyGTK?**
- V] Commencement**
- VI] Construction de l'interface**
- VII] Ajout du code de l'application**
- VIII] Comment compiler BitPodder?**

I] Licence

Ce tutorial est couvert par la licence FDL.

Le programme développé dans ce tutorial est couvert par la licence GPL.

II] But de l'article

Le but de cet article est de créer un agrégateur de podcast à l'aide de PyGTK (Python + Glade + PyGTK). Ce logiciel, nommé BitPodder, est un agrégateur permettant d'obtenir des fichiers .torrent dont les adresses ont été publiées dans des flux RSS. En expliquant chaque étape de la création de BitPodder, j'espère aider certains développeurs windows ayant quelques difficultés avec Python et Glade.

III] Ce que vous devez avoir avant de commencer...

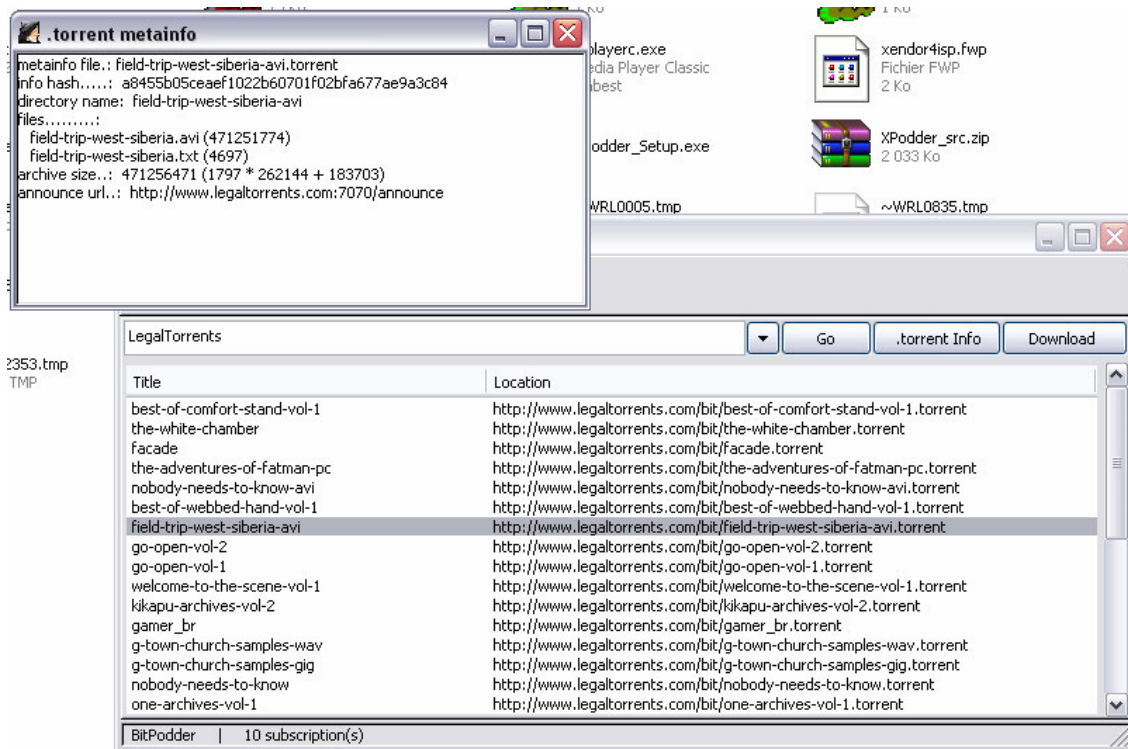
- python
- Gtk+/Win32 Development Environment (<http://gladewin32.sourceforge.net/>)
- Gtk+/Win32 Runtime Environment (<http://gladewin32.sourceforge.net/>)
- Pygtk et gtkmm pour Win32 (http://www.pcpm.ucl.ac.be/~gustin/win32_ports/)
- Tepache (<http://primates.ximian.com/~sandino/python-glade/>)
- Vous pouvez obtenir BitPodder_src.zip, l'archive ZIP contenant le code source de BitPodder à cette adresse: <http://sourceforge.net/projects/korriban>

IV] Qu'est-ce que PyGTK?

"PyGTK provides a **convenient wrapper** for the GTK+ library for use in Python programs, taking care of many of the boring details such as managing memory and type casting. When combined with PyORBit and gnome-python, it can be used to write full featured Gnome applications." (<http://www.pygtk.org/about.html>)

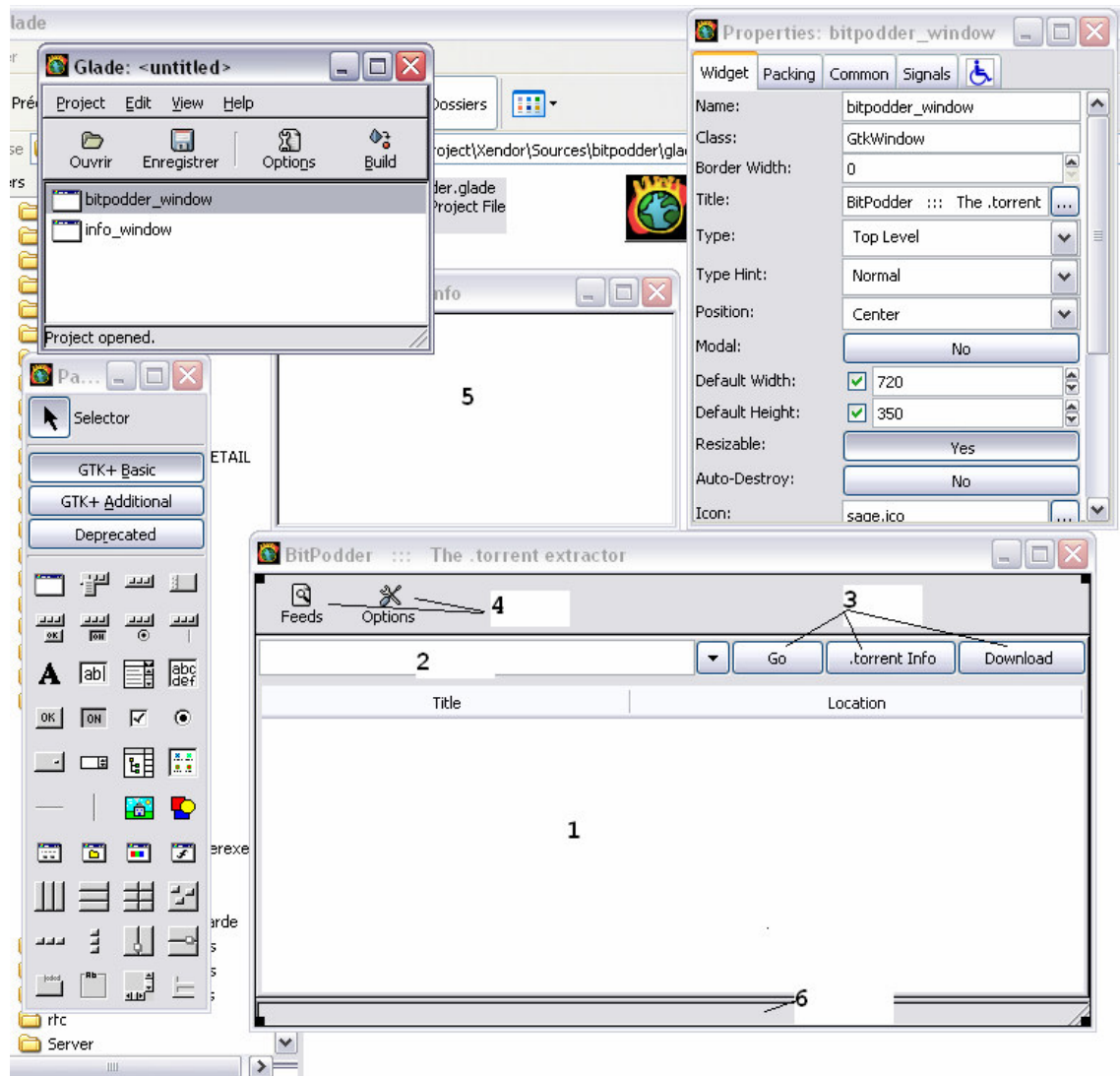
V] Commencement

Ceci est la copie d'écran de BitPodder :

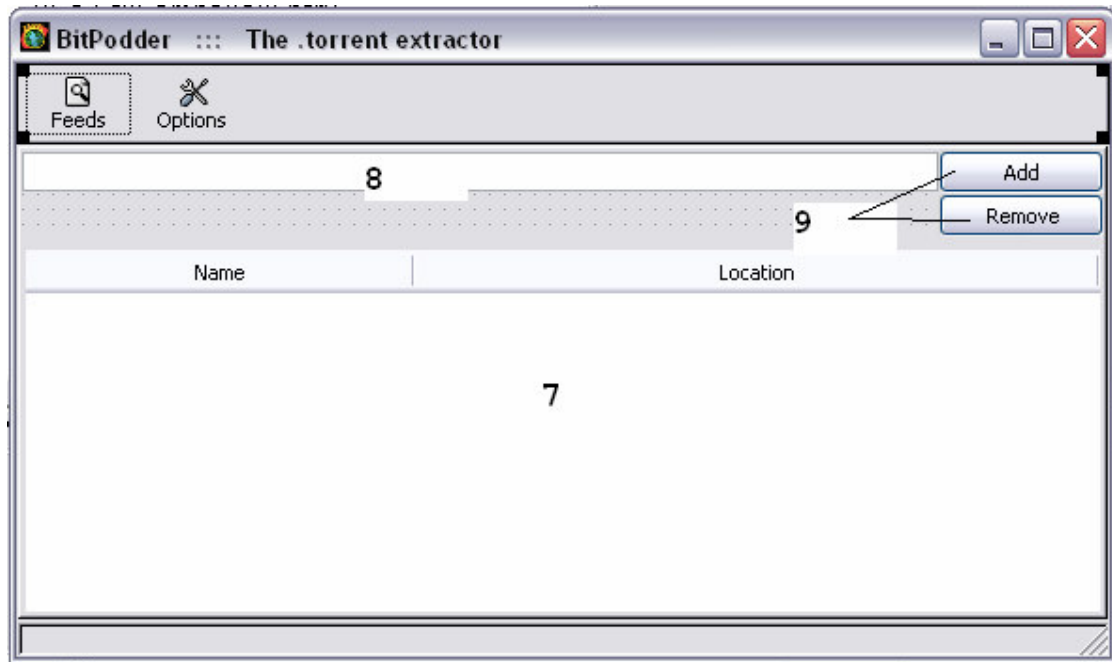


VI] Construction de l'interface

Nous allons commencer par créer l'interface utilisateur de BitPodder. Grâce à Glade, cela sera possible très aisément sans écrire une ligne de code. BitPodder possède beaucoup de widgets donc avant de continuer cet article, j'espère que vous avez déjà utilisé glade ou que vous connaissez quelques widgets et connaissez comment fonctionnent les signaux sous GTK+. Si toutes vos réponses sont: non, je vous conseille de lire des articles comme "Writing PyGTK applications in a visual way" (<http://primates.ximian.com/~sandino/python-glade/>) pour au moins avoir une idée du fonctionnement de glade.



- 1: un GtkCList
- 2: un GtkComboBoxEntry
- 3: un GtkButton
- 4: quelques GtkToolButton
- 5: un GtkTextView
- 6: un GtkStatusBar



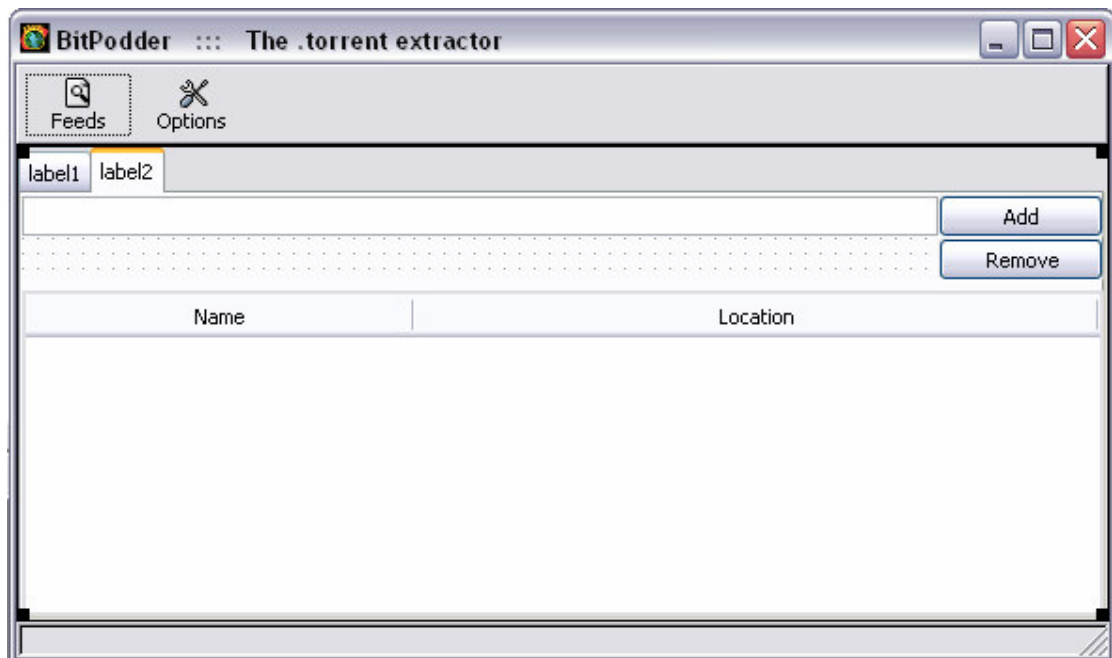
Maintenant les widgets de la page Option:

7: un GtkCList

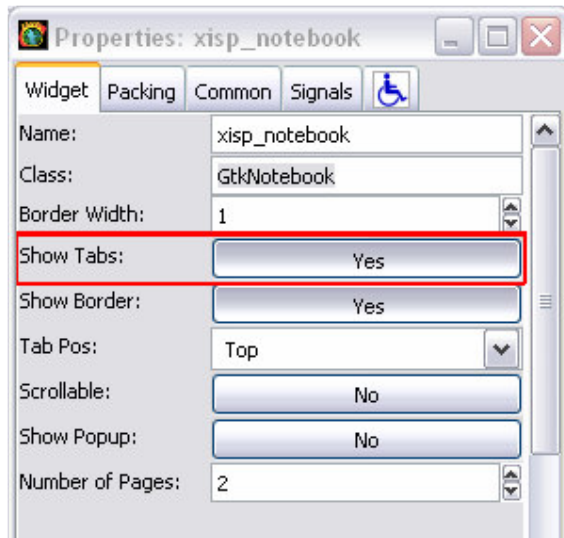
8: un GtkEntry

9: quelques GtkButton

J'ai dit "page" car pour switcher entre les pages Feeds et Options, j'ai utilisé un GtkNotebook puis j'ai caché les labels des pages du Notebook. Voici la vraie page avant d'avoir caché les labels :



Pour cacher les fameux labels, vous devez cliquer ici :



Je vais arrêter les explications sur la création de l'interface utilisateur sinon l'article serait trop long. Deplus, vous pouvez consulter le fichier glade (contenu dans l'archive) pour plus d'informations.

VII] Ajout du code de l'application

Nous allons utiliser un outil nommé tepache (<http://primates.ximian.com/~sandino/python-glade/tepache/>). Avec ce logiciel, il sera possible de générer le code en python correspondant à l'interface que l'on vient de créer. Tepache nous générera un script python possédant des classes claires qui nous permettrons de contrôler chacun des widgets.

Avant d'utiliser tepache, téléchargez son archive, dézippez-le puis installez-le en tapant "python setup.py install" (grâce à cmd.exe) dans le répertoire où l'archive a été dézippée.

Lancer un autre cmd.exe puis allez dans le répertoire où se trouve votre fichier glade. Copiez le script tepache dans ce même répertoire puis entrez dans cmd.exe : **python tepache xxx.glade** (xxx.glade est le nom de votre fichier glade, pour moi c'est **bitpodder.glade**).

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\khaalel>cd mes documents\project\xendor\sources\bitpodder\glade
C:\Documents and Settings\khaalel\Mes documents\Project\xendor\Sources\bitpodder\glade>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 0CE9-A850

Répertoire de C:\Documents and Settings\khaalel\Mes documents\Project\xendor\Sources\bitpodder\glade
09/09/2005  14:37    <REP>          -
09/09/2005  14:37    <REP>          -
08/09/2005  11:15             22 045 bitpodder.glade
08/09/2005  11:15             22 045 Copie de bitpodder.glade
01/08/2005  23:10             31 897 tepache
                3 fichier(s)          75 987 octets
                2 Rép(s)    8 782 319 616 octets libres

C:\Documents and Settings\khaalel\Mes documents\Project\xendor\Sources\bitpodder\glade>python tepache bitpodder.glade
written file bitpodder.py
C:\Documents and Settings\khaalel\Mes documents\Project\xendor\Sources\bitpodder\glade>
```

Si tout s'est bien passé, vous devriez avoir 3 nouveaux fichiers dans le répertoire : bitpodder.py, bitpodder.py.orig et SimpleGladApp.py.

bitpodder.py est le script python dans lequel on va écrire le code du programme. bitpodder.py.orig ne doit être en aucun cas modifié ou supprimé car si vous désirez mettre à jour bitpodder.py après une modification du fichier glade, tepache va consulter ce fichier pour ne pas supprimer le code que vous avez déjà saisi. SimpleGladApp.py contient le code python que votre programme va exécuter quand l'utilisateur fera des actions de base comme fermer le programme en cliquant sur la croix.

Dans un prochain article, j'expliquerai comment créer votre propre tepache ou votre SimpleGladApp.py pour customiser votre programme : par exemple, parfois quand un utilisateur quitte le programme, il peut être intéressant de créer un fichier qui contiendrait les dernières positions de la fenêtre, ou des informations relatives à l'utilisateur...

Maintenant, nous allons ouvrir bitpodder.py pour ajouter le code du programme.

Bitpodder.py devrait ressembler à cela :

```
#!/usr/bin/env python
# -*- coding: UTF8 -*-

# Python module bitpodder.py
# Autogenerated from bitpodder.glade
# Generated on Fri Sep 09 14:38:31 2005

# Warning: Do not modify any context comment such as #--
# They are required to keep user's code

import os

import gtk

from SimpleGladApp import SimpleGladApp
from SimpleGladApp import bindtextdomain

app_name = "bitpodder"
```

```

app_version = "0.0.1"

glade_dir = ""
locale_dir = ""

bindtextdomain(app_name, locale_dir)

class BitpodderWindow(SimpleGladeApp):

    def __init__(self, path="bitpodder.glade",
                 root="bitpodder_window",
                 domain=app_name, **kwargs):
        path = os.path.join(glade_dir, path)
        SimpleGladeApp.__init__(self, path, root, domain, **kwargs)

    #-- BitpodderWindow.new {
    def new(self):
        print "A new %s has been created" % self.__class__.__name__
    #-- BitpodderWindow.new }

    #-- BitpodderWindow custom methods {
    # Write your own methods here
    #-- BitpodderWindow custom methods }

    #-- BitpodderWindow.on_feed_toolbutton_clicked {
    def on_feed_toolbutton_clicked(self, widget, *args):
        print "on_feed_toolbutton_clicked called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_feed_toolbutton_clicked }

    #-- BitpodderWindow.on_options_toolbutton_clicked {
    def on_options_toolbutton_clicked(self, widget, *args):
        print "on_options_toolbutton_clicked called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_options_toolbutton_clicked }

    #-- BitpodderWindow.on_xisp_go_button_clicked {
    def on_xisp_go_button_clicked(self, widget, *args):
        print "on_xisp_go_button_clicked called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_xisp_go_button_clicked }

    #-- BitpodderWindow.on_show_button_clicked {
    def on_show_button_clicked(self, widget, *args):
        print "on_show_button_clicked called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_show_button_clicked }

    #-- BitpodderWindow.on_xisp_play_button_clicked {
    def on_xisp_play_button_clicked(self, widget, *args):
        print "on_xisp_play_button_clicked called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_xisp_play_button_clicked }

    #-- BitpodderWindow.on_xisp_results_clist_select_row {
    def on_xisp_results_clist_select_row(self, widget, *args):
        print "on_xisp_results_clist_select_row called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_xisp_results_clist_select_row }

    #-- BitpodderWindow.on_xisp_add_button_clicked {
    def on_xisp_add_button_clicked(self, widget, *args):
        print "on_xisp_add_button_clicked called with self.%s" % widget.get_name()
    #-- BitpodderWindow.on_xisp_add_button_clicked }

    #-- BitpodderWindow.on_xisp_remove_button_clicked {
    def on_xisp_remove_button_clicked(self, widget, *args):
        print "on_xisp_remove_button_clicked called with self.%s" % widget.get_name()

```

```

#-- BitpodderWindow.on_xisp_remove_button_clicked }

#-- BitpodderWindow.on_xisp_option_clist_select_row {
def on_xisp_option_clist_select_row(self, widget, *args):
    print "on_xisp_option_clist_select_row called with self.%s" % widget.get_name()
#-- BitpodderWindow.on_xisp_option_clist_select_row }

class InfoWindow(SimpleGladeApp):

    def __init__(self, path="bitpodder.glade",
                 root="info_window",
                 domain=app_name, **kwargs):
        path = os.path.join(glade_dir, path)
        SimpleGladeApp.__init__(self, path, root, domain, **kwargs)

#-- InfoWindow.new {
def new(self):
    print "A new %s has been created" % self.__class__.__name__
#-- InfoWindow.new }

#-- InfoWindow custom methods {
# Write your own methods here
#-- InfoWindow custom methods }

#-- main {

def main():
    bitpodder_window = BitpodderWindow()
    info_window = InfoWindow()

    bitpodder_window.run()

if __name__ == "__main__":
    main()

#-- main }

```

Mais nous voulons qu'il ressemble à cela (c'est le code source complet de bitpodder, ne vous inquiétez pas si cela ressemble à du charabia pour vous, je vais expliquer plus en détail chaque ligne) :

```

#!/usr/bin/env python
# -*- coding: UTF8 -*-

# Python module xpodder.py
# Autogenerated from xpodder.glade
# Generated on Tue Sep 06 00:21:46 2005

# Warning: Do not modify any context comment such as #--
# They are required to keep user's code

import os, re, urllib, urlparse, feedparser, codecs
from sys import *
from os.path import *
from sha import *
from bencode import *
import gtk

from SimpleGladeApp import SimpleGladeApp
from SimpleGladeApp import bindtextdomain

```

```

app_name = "bitpodder"
app_version = "0.1"

glade_dir = ""
locale_dir = ""

bindtextdomain(app_name, locale_dir)
XISP_Results_Row_Variable = []
XISP_Options_Row_Variable = []
XISP_Link = ""

class BitPodderWindow(SimpleGladeApp):

    def __init__(self, path="glade\\bitpodder.glade",
                 root="bitpodder_window",
                 domain=app_name, **kwargs):
        path = os.path.join(glade_dir, path)
        SimpleGladeApp.__init__(self, path, root, domain, **kwargs)

    #-- Xendor4ispWindow.new {
    def new(self):
        counter = 0
        self.xisp_results_clist.clear()
        self.xisp_option_clist.clear()
        fdw = open("bitpodder.fwp", 'r').readlines()
        for ligne in fdw:
            lesplit = ligne.split("?:?:")
            lien = lesplit[1].strip()
            rpln = self.xisp_option_clist.append([lesplit[0], lien])
            rpln = self.xisp_comboboxentry.append_text(lesplit[0])
            counter = counter + 1

            # http://www.pygtk.org/pygkktutorial/sec-statusbars.html
        global xisp_id, xispext_id
        stamess = " BitPodder | " + str(counter) + " subscription(s)"
        xispext_id = self.xisp_statusbar.get_context_id("XISP Statusbar")
        xisp_id = self.xisp_statusbar.push(xispext_id, stamess)
    #-- Xendor4ispWindow.new }

    #-- Xendor4ispWindow custom methods {
    # Write your own methods here
    #-- Xendor4ispWindow custom methods }

    #-- Xendor4ispWindow.on_feed_toolbutton_clicked {
    def on_feed_toolbutton_clicked(self, widget, *args):
        self.xisp_notebook.set_current_page(0)
    #-- Xendor4ispWindow.on_feed_toolbutton_clicked }

    #-- Xendor4ispWindow.on_options_toolbutton_clicked {
    def on_options_toolbutton_clicked(self, widget, *args):
        self.xisp_notebook.set_current_page(1)
    #-- Xendor4ispWindow.on_options_toolbutton_clicked }

    #-- Xendor4ispWindow.on_xisp_go_button_clicked {
    def on_xisp_go_button_clicked(self, widget, *args):
        self.xisp_results_clist.clear()
        xentry = self.xisp_comboboxentry.child
        xentry = self.xisp_comboboxentry.get_child()
        choosen = xentry.get_text()

        urrl = get_xisp_id(choosen)

        page = urllib.urlopen(urrl).read()

```

```

page = page.replace( '&quot;', '')
page = page.replace( '&amp;', '&')
page = page.replace( '&lt;', '<')
page = page.replace( '&gt;', '>')
page = page.replace( '&nbsp;', ' ')
# t = re.compile('.*<title>[a-zA-Z0-9_].*</title>')
p = re.compile('.*url=["\']([\^\"]*)["\'].*torrent')
files = p.findall(page)
# titless = t.findall(page)

cou = 0
for fff in files:
# na = titless[cou+1].strip("<title>")
# na = na.strip("</title>")
# try:
# metainfo_file = urllib.urlopen(fff)
# metainfo = bdecode(metainfo_file.read())
# info = metainfo['info']
# na = info['name']
# except ValueError:
# na = os.path.splitext(os.path.split(fff)[1])[0]
# if info.has_key('length'):
# na = info['name']
# else:
# na = info['name']
# na = os.path.splitext(os.path.split(fff)[1])[0]
na = os.path.splitext(os.path.split(fff)[1])[0]
scomplink = self.xisp_results_clist.append([na, fff])
cou = cou + 1

#-- Xendor4ispWindow.on_xisp_go_button_clicked }

#-- Xendor4ispWindow.on_xisp_play_button_clicked {
def on_xisp_play_button_clicked(self, widget, *args):
if len(XISP_Results_Row_Variable) != 0:
xrow = XISP_Results_Row_Variable[0]
scomplink = self.xisp_results_clist.get_text(xrow, 1)
title = os.path.split(scomplink)[1]

sharedfolder = 0
for x in os.listdir(os.getcwd()):
if x == "Torrents Files": sharedfolder = 1

fol = os.getcwd() + "\\\" + "Torrents Files"
if sharedfolder == 0:
err = os.mkdir(fol)

title = fol + "\\\" + str(title)
fdm = open(title, 'a')
fdm.close()
tor = urllib.urlretrieve(scomplink, title)
#-- Xendor4ispWindow.on_xisp_play_button_clicked }

#-- Xendor4ispWindow.on_xisp_results_clist_select_row {
def on_xisp_results_clist_select_row(self, widget, *args):
global XISP_Results_Row_Variable
XISP_Results_Row_Variable = self.xisp_results_clist.selection
#-- Xendor4ispWindow.on_xisp_results_clist_select_row }

#-- Xendor4ispWindow.on_show_button_clicked {
def on_show_button_clicked(self, widget, *args):
if len(XISP_Results_Row_Variable) != 0:
xrow = XISP_Results_Row_Variable[0]

```

```

        scomplink = self.xisp_results_clist.get_text(xrow, 1)
        global XISP_Link
        XISP_Link = scomplink
        info_window = InfoWindow()
    #-- Xendor4ispWindow.on_show_button_clicked }

    #-- Xendor4ispWindow.on_xisp_add_button_clicked {
    def on_xisp_add_button_clicked(self, widget, *args):
        xtring = self.xisp_add_entry.get_text()
        if xtring != "":
            lurl = xtring
            lxml = feedparser.parse(lurl.strip())
            title = lxml.channel.title
            fdw = open("bitpodder.fwp", 'a')
            if xtring != "":
                ch = title + "?:?:?" + xtring
                fdw.write(ch)
                fdw.write('\n')
            fdw.close()

            xtring = self.xisp_add_entry.set_text("")
            self.xisp_option_clist.clear()

            bitpodder_window = BitPodderWindow()
            self.bitpodder_window.destroy()

    #-- Xendor4ispWindow.on_xisp_add_button_clicked }

    #-- Xendor4ispWindow.on_xisp_remove_button_clicked {
    def on_xisp_remove_button_clicked(self, widget, *args):
        if len(XISP_Options_Row_Variable) != 0:
            xtrow = XISP_Options_Row_Variable[0]
            if xtrow != "":
                self.xisp_option_clist.remove(xtrow)

            fdw = open("bitpodder.fwp", 'w')
            for drow in range(self.xisp_option_clist.rows):
                ame = self.xisp_option_clist.get_text(drow, 0)
                d = self.xisp_option_clist.get_text(drow, 1)
                ch = ame + "?:?:?" + d
                fdw.write(ch)
                fdw.write('\n')
            fdw.close()

            bitpodder_window = BitPodderWindow()
            self.bitpodder_window.destroy()

    #-- Xendor4ispWindow.on_xisp_remove_button_clicked }

    #-- Xendor4ispWindow.on_xisp_option_clist_select_row {
    def on_xisp_option_clist_select_row(self, widget, *args):
        global XISP_Options_Row_Variable
        XISP_Options_Row_Variable = self.xisp_option_clist.selection
    #-- Xendor4ispWindow.on_xisp_option_clist_select_row }

class InfoWindow(SimpleGladeApp):

    def __init__(self, path="glade\\bitpodder.glade",
                 root="info_window",
                 domain=app_name, **kwargs):
        path = os.path.join(glade_dir, path)

```

```
SimpleGladeApp.__init__(self, path, root, domain, **kwargs)
```

```
!-- InfoWindow.new {
```

```
def new(self):
```

```
    global XISP_Link
```

```
    link = XISP_Link
```

```
    metainfo_file = urllib.urlopen(link)
```

```
    metainfo = bdecode(metainfo_file.read())
```

```
    info = metainfo['info']
```

```
    info_hash = sha(bencode(info))
```

```
    txt = "metainfo file.: %s" % basename(link) + "\n"
```

```
    txt = txt + "info hash.....: %s" % info_hash.hexdigest() + "\n"
```

```
    piece_length = info['piece length']
```

```
    if info.has_key('length'):
```

```
        # let's assume we just have a file
```

```
        txt = txt + "file name.....: %s" % info['name'] + "\n"
```

```
        file_length = info['length']
```

```
        name = 'file size.....: '
```

```
    else:
```

```
        # let's assume we have a directory structure
```

```
        txt = txt + "directory name: %s" % info['name'] + "\n"
```

```
        txt = txt + "files.....: " + "\n"
```

```
        file_length = 0
```

```
        for file in info['files']:
```

```
            path = ""
```

```
            for item in file['path']:
```

```
                if (path != ""):
```

```
                    path = path + "/"
```

```
                    path = path + item
```

```
                txt = txt + " %s (%d)" % (path, file['length']) + "\n"
```

```
                file_length += file['length']
```

```
                name = 'archive size..: '
```

```
    piece_number, last_piece_length = divmod(file_length, piece_length)
```

```
    txt = txt + "%s %i (%i * %i + %i)" \
```

```
        % (name, file_length, piece_number, piece_length, last_piece_length) + "\n"
```

```
    txt = txt + "announce url..: %s" % metainfo['announce'] + "\n"
```

```
    if metainfo.has_key('announce-list'):
```

```
        list = []
```

```
        for tier in metainfo['announce-list']:
```

```
            for tracker in tier:
```

```
                list+=[tracker,',']
```

```
            del list[-1]
```

```
            list+=[']
```

```
        del list[-1]
```

```
        liststring = ""
```

```
        for i in list:
```

```
            liststring+=i
```

```
        txt = txt + "announce-list.: %s" % liststring + "\n"
```

```
    if metainfo.has_key('httpseeds'):
```

```
        list = []
```

```
        for seed in metainfo['httpseeds']:
```

```
            list += [seed,']
```

```
        del list[-1]
```

```
        liststring = ""
```

```
        for i in list:
```

```
            liststring+=i
```

```
        txt = txt + "http seeds.....: %s" % liststring + "\n"
```

```
    if metainfo.has_key('comment'):
```

```
        txt = txt + "comment.....: %s" % metainfo['comment'] + "\n"
```

```

        buf = self.metainfo_textview.get_buffer()
        buf.set_text(txt)
    #-- InfoWindow.new }

    #-- InfoWindow custom methods {
    # Write your own methods here
    #-- InfoWindow custom methods }

def get_xisp_id(identifiant):
    fdpn = open("bitpodder.fwp").readlines()
    reqxisp = re.compile(identifiant)
    for pe in fdpn:
        if reqxisp.search(pe):
            PeersString = pe.split("?:?:")
    return PeersString[1]
#-- main {

def main():
    bitpodder_window = BitPodderWindow()

    bitpodder_window.run()

if __name__ == "__main__":
    main()

#-- main }

```

Comme vous pouvez le voir, Bitpodder est un petit programme de seulement 300 lignes : grâce au langage python !!!

Commençons l'analyse ligne par ligne de bitpodder.py.

En premier, il y a quelques inclusions de modules :

```

import os, re, urllib, urlparse, feedparser, codecs
from sys import *
from os.path import *
from sha import *
from bencode import *
import gtk

```

Feedparser peut être trouvé sur feedparser.org et bencode peut être trouvé dans le package de BitTorrent. Mais pour vous faciliter la vie, j'ai inclus tous ces modules dans BitPodder_src.zip.

Puis, il y a le code généré par tepache :

```

from SimpleGladeApp import SimpleGladeApp
from SimpleGladeApp import bindtextdomain

app_name = "bitpodder"
app_version = "0.1"

glade_dir = ""
locale_dir = ""

bindtextdomain(app_name, locale_dir)

```

Maintenant, il y a 3 variables globales qui seront utilisées dans l'application :

```
XISP_Results_Row_Variable = []
XISP_Options_Row_Variable = []
XISP_Link = ""
```

Ensuite, nous avons la classe de la fenêtre principale de BitPodder que j'ai nommé bitpodder_window dans glade. Comme vous pouvez le voir, j'ai modifié une ligne : "glade\\bitpodder.glade". Pourquoi ? Car j'ai décidé de mettre le fichier glade dans un autre répertoire : ".\glade\". Cette ligne contient 2 backslash car \b est une séquence d'échappement dans python donc il nous faut annuler le backslash précédant la lettre b.

```
class BitPodderWindow(SimpleGladeApp):

    def __init__(self, path="glade\\bitpodder.glade",
                 root="bitpodder_window",
                 domain=app_name, **kwargs):
        path = os.path.join(glade_dir, path)
        SimpleGladeApp.__init__(self, path, root, domain, **kwargs)

#-- Xendor4ispWindow.new {
```

Maintenant, nous allons remplir le code de la méthode new(). Que représente-t-elle? Le code de cette méthode sera exécuté avant que la fenêtre soit créée, grâce à cela, on pourra initialiser quelques widgets avant leur apparition.

Commençons par effacer le contenu des 2 CLists que nous utilisons dans BitPodder :

```
def new(self):
    counter = 0
    self.xisp_results_clist.clear()
    self.xisp_option_clist.clear()
```

Puis, nous allons lire le fichier qui contient les adresses des flux RSS. Une ligne de ce fichier ressemble à ceci : "BitTorrent @ AnimeSuki.com:?:?:http://www.animesuki.com/rss.php"

Explications : en premier, nous avons le nom du feed, puis il y a la chaîne ":?:?:", puis nous avons l'url du feed.

Donc, nous ouvrons le fichier et à chaque ligne, nous allons spliter la ligne en s'aidant de la chaîne ":?:?:". Ensuite, on ajoute le nom de chaque feed dans le comboboxentry puis le nom et l'url de chaque feed dans le CList de la page Options.

Attention !!! Glade a un petit bug... vous devez ajouter le code suivant dans le fichier glade à l'endroit où est défini le comboboxentry : "<property name="items" translatable="yes"></property>" sinon, vous risquez d'avoir un beau message d'erreur lors du lancement de l'application. La définition complète du comboboxentry doit ressembler à cela :

```
<child>
  <widget class="GtkComboBoxEntry" id="xisp_comboboxentry">
    <property name="visible">True</property>
    <property name="items" translatable="yes"></property>
    <property name="add_tearoffs">False</property>
    <property name="has_frame">True</property>
    <property name="focus_on_click">True</property>
  </widget>
  <packing>
    <property name="padding">0</property>
```

```
<property name="expand">True</property>
<property name="fill">True</property>
</packing>
</child>
```

```
fdw = open("bitpodder.fwp", 'r').readlines()
for ligne in fdw:
    lesplit = ligne.split("?:?:")
    lien = lesplit[1].strip()
    rpln = self.xisp_option_clist.append([lesplit[0], lien])
    rpln = self.xisp_comboboxentry.append_text(lesplit[0])
    counter = counter + 1
```

Maintenant nous pouvons ajouter un message dans la barre d'état :

```
# http://www.pygtk.org/pyggtkutorial/sec-statusbars.html
global xisp_id, xispext_id
stamess = " BitPodder | " + str(counter) + " subscription(s)"
xispext_id = self.xisp_statusbar.get_context_id("XISP Statusbar")
xisp_id = self.xisp_statusbar.push(xispext_id, stamess)
#-- Xendor4ispWindow.new }

#-- Xendor4ispWindow custom methods {
# Write your own methods here
#-- Xendor4ispWindow custom methods }
```

Maintenant, nous allons dire à l'application quoi faire lorsque l'utilisateur clique sur les bouton Feeds et Options. Comme je l'ai dit plus haut, j'ai utilisé un Notebook donc il nous faut juste ouvrir la bonne page : chaque page est identifiée par un numéro :

```
#-- Xendor4ispWindow.on_feed_toolbutton_clicked {
def on_feed_toolbutton_clicked(self, widget, *args):
    self.xisp_notebook.set_current_page(0)
#-- Xendor4ispWindow.on_feed_toolbutton_clicked }

#-- Xendor4ispWindow.on_options_toolbutton_clicked {
def on_options_toolbutton_clicked(self, widget, *args):
    self.xisp_notebook.set_current_page(1)
#-- Xendor4ispWindow.on_options_toolbutton_clicked }
```

Maintenant, nous allons dire à l'application quoi faire lorsque l'utilisateur clique sur le bouton Go.

Nous allons obtenir le nom du feed sélectionné par l'utilisateur dans le comboboxentry et mettre ce nom dans la variable *chosen* :

```
#-- Xendor4ispWindow.on_xisp_go_button_clicked {
def on_xisp_go_button_clicked(self, widget, *args):
    self.xisp_results_clist.clear()
    xentry = self.xisp_comboboxentry.child
    xentry = self.xisp_comboboxentry.get_child()
    chosen = xentry.get_text()
```

Après l'obtention du nom du feed, nous devons obtenir l'url associé à ce nom, c'est pour cela que j'ai créé une fonction, nommé `get_xisp_id()`, que je définirai plus tard.

Lorsque l'url nous est retournée par la fonction, il faut lire le code source de la page web, remplacer des caractères spéciaux et rechercher les chaînes semblables à cela : "url=http://blablabla.torrent" (vous devez savoir que les adresses des fichiers torrent sont mis dans la balise <enclosure> qui ressemble à : "<enclosure

```
url="http://www.anime-kraze.org/torrent/[Ani-Kraze]_Tsubasa_Chronicle_-_16_[2CE95BC4].avi.torrent" length="14008" type="application/x-bittorrent"/>").
```

Ensuite grâce à la fonction `os.path.splitext(xxx)[0]` nous allons pouvoir obtenir le nom du fichier torrent pour ajouter le tout dans le CList de la page Feed.

```
urrl = get_xisp_id(choosen)

page = urllib.urlopen(urrl).read()
page = page.replace( '&quot;', '' )
page = page.replace( '&amp;', '&' )
page = page.replace( '&lt;', '<' )
page = page.replace( '&gt;', '>' )
page = page.replace( '&nbsp;', ' ' )
# t = re.compile('.*<title>[a-zA-Z0-9_].*</title>')
p = re.compile('.*url=["\']([^\"]*)["\'].*torrent')
files = p.findall(page)
# titless = t.findall(page)

cou = 0
for fff in files:
# na = titless[cou+1].strip("<title>")
# na = na.strip("</title>")
# try:
#     metainfo_file = urllib.urlopen(fff)
#     metainfo = bdecode(metainfo_file.read())
#     info = metainfo['info']
#     na = info['name']
# except ValueError:
#     na = os.path.splitext(os.path.split(fff)[1])[0]
# if info.has_key('length'):
#     na = info['name']
# else:
#     na = info['name']
# na = os.path.splitext(os.path.split(fff)[1])[0]
na = os.path.splitext(os.path.split(fff)[1])[0]
scomplink = self.xisp_results_clist.append([na, fff])
cou = cou + 1

#-- Xendor4ispWindow.on_xisp_go_button_clicked }
```

Maintenant, nous allons dire à l'application quoi faire lorsque l'utilisateur sélectionne une ligne dans le CList. Quand une ligne (row) est sélectionnée, GTK+ envoie le numéro de la ligne sélectionnée. Nous allons utiliser une variable globale pour pouvoir utiliser le numéro de la ligne sélectionnée avec différentes fonctions :

```
#-- Xendor4ispWindow.on_xisp_results_clist_select_row {
def on_xisp_results_clist_select_row(self, widget, *args):
    global XISP_Results_Row_Variable
    XISP_Results_Row_Variable = self.xisp_results_clist.selection
#-- Xendor4ispWindow.on_xisp_results_clist_select_row }
```

Maintenant, nous allons dire à l'application quoi faire lorsque l'utilisateur clique sur le bouton Download.

Premièrement, nous allons vérifier que la variable globale associée au CList a une valeur, si pas de valeur, on ne fait rien. Si il y a une valeur, nous prenons le numéro de la ligne puis on affecte le contenu de la seconde colonne de la ligne (celle qui contient l'url du torrent) à une variable et pour finir, on télécharge le torrent dans le répertoire ".\Torrent Files\" (qui est créé si c'est le tout premier téléchargement de l'utilisateur) :

```

#-- Xendor4ispWindow.on_xisp_play_button_clicked {
def on_xisp_play_button_clicked(self, widget, *args):
    if len(XISP_Results_Row_Variable) != 0:
        xrow = XISP_Results_Row_Variable[0]
        scomplink = self.xisp_results_clist.get_text(xrow, 1)
        title = os.path.split(scomplink)[1]

        sharedfolder = 0
        for x in os.listdir(os.getcwd()):
            if x == "Torrents Files": sharedfolder = 1

        fol = os.getcwd() + "\\\" + "Torrents Files"
        if sharedfolder == 0:
            err = os.mkdir(fol)

        title = fol + "\\\" + str(title)
        fdm = open(title, 'a')
        fdm.close()
        tor = urllib.urlretrieve(scomplink, title)
#-- Xendor4ispWindow.on_xisp_play_button_clicked }

```

Maintenant, nous allons dire à l'application quoi faire lorsque l'utilisateur clique sur le bouton torrent Info.

Comme pour le bouton Download, on va essayer d'obtenir le numéro de la ligne sélectionnée puis l'url du torrent que nous mettrons dans une autre variable globale. Pour finir, nous allons ouvrir une autre fenêtre : celle qui affichera les informations voulues :

```

#-- Xendor4ispWindow.on_show_button_clicked {
def on_show_button_clicked(self, widget, *args):
    if len(XISP_Results_Row_Variable) != 0:
        xrow = XISP_Results_Row_Variable[0]
        scomplink = self.xisp_results_clist.get_text(xrow, 1)
        global XISP_Link
        XISP_Link = scomplink
        info_window = InfoWindow()
#-- Xendor4ispWindow.on_show_button_clicked }

```

Maintenant, nous allons écrire le code de la page Option.

Nous allons dire à l'application quoi faire lorsque l'utilisateur clique sur le bouton Add. Pour résumer le code, si l'utilisateur entre l'url du feed RSS, nous allons ouvrir le flux RSS pour obtenir son nom puis ajouter nom et url dans le fichier bitpodder.fwp. Pour finir, nous relançons le programme (en ouvrant une nouvelle fenêtre et détruisant la fenêtre courante) :

```

#-- Xendor4ispWindow.on_xisp_add_button_clicked {
def on_xisp_add_button_clicked(self, widget, *args):
    xstring = self.xisp_add_entry.get_text()
    if xstring != "":
        lurl = xstring
        lxml = feedparser.parse(lurl.strip())
        title = lxml.channel.title
        fdw = open("bitpodder.fwp", 'a')
        if xstring != "":
            ch = title + "?:?:?" + xstring
            fdw.write(ch)
            fdw.write('\n')
        fdw.close()

    xstring = self.xisp_add_entry.set_text("")
    self.xisp_option_clist.clear()

```

```

bitpodder_window = BitPodderWindow()
self.bitpodder_window.destroy()

#-- Xendor4ispWindow.on_xisp_add_button_clicked }

```

Maintenant, nous allons dire à l'application quoi faire lorsque l'utilisateur clique sur le bouton Remove.

Premièrement, nous allons vérifier qu'une ligne a été sélectionnée, si oui, nous supprimons cette ligne, détruisons le fichier bitpodder.fwp, créons un nouveau fichier bitpodder.fwp et stockons les lignes restant dans le CList. Pour finir, on relance le programme.

```

#-- Xendor4ispWindow.on_xisp_remove_button_clicked {
def on_xisp_remove_button_clicked(self, widget, *args):
    if len(XISP_Options_Row_Variable) != 0:
        xtrow = XISP_Options_Row_Variable[0]
        if xtrow != "":
            self.xisp_option_clist.remove(xtrow)

        fdw = open("bitpodder.fwp", 'w')
        for drow in range(self.xisp_option_clist.rows):
            ame = self.xisp_option_clist.get_text(drow, 0)
            d = self.xisp_option_clist.get_text(drow, 1)
            ch = ame + "?:?:" + d
            fdw.write(ch)
            fdw.write('\n')
        fdw.close()

        bitpodder_window = BitPodderWindow()
        self.bitpodder_window.destroy()

#-- Xendor4ispWindow.on_xisp_remove_button_clicked }

#-- Xendor4ispWindow.on_xisp_option_clist_select_row {
def on_xisp_option_clist_select_row(self, widget, *args):
    global XISP_Options_Row_Variable
    XISP_Options_Row_Variable = self.xisp_option_clist.selection
#-- Xendor4ispWindow.on_xisp_option_clist_select_row }

```

Maintenant, nous allons voir comment fonctionne la seconde fenêtre de BitPodder. Le but de cette fenêtre est d'afficher des informations sur le torrent sélectionné, si aucune information n'est affichée : le torrent n'est pas valide ou il y a eu une erreur de type timeoutsocket.

Quand la fenêtre est lancée, nous allons vérifier que la variable globale XISP_Link contient une url valide. Si oui, nous ouvrons le torrent sans le télécharger et nous recherchons des informations intéressantes.

Je n'expliquerai pas comment Bram Cohen a décidé d'écrire les fichiers .torrent; Si le fonctionnement de BitTorrent vous intéresse, je vous conseille de lire le code source de ce P2P (qui est écrit en Python) ou de lire la suite de mon code : c'est très facile à comprendre.

```

class InfoWindow(SimpleGladeApp):

    def __init__(self, path="glade\\bitpodder.glade",
                 root="info_window",
                 domain=app_name, **kwargs):
        path = os.path.join(glade_dir, path)

```

```
SimpleGladeApp.__init__(self, path, root, domain, **kwargs)
```

```
!-- InfoWindow.new {
```

```
def new(self):
```

```
    global XISP_Link
```

```
    link = XISP_Link
```

```
    metainfo_file = urllib.urlopen(link)
```

```
    metainfo = bdecode(metainfo_file.read())
```

```
    info = metainfo['info']
```

```
    info_hash = sha(bencode(info))
```

```
    txt = "metainfo file.: %s" % basename(link) + "\n"
```

```
    txt = txt + "info hash.....: %s" % info_hash.hexdigest() + "\n"
```

```
    piece_length = info['piece length']
```

```
    if info.has_key('length'):
```

```
        # let's assume we just have a file
```

```
        txt = txt + "file name.....: %s" % info['name'] + "\n"
```

```
        file_length = info['length']
```

```
        name = 'file size.....: '
```

```
    else:
```

```
        # let's assume we have a directory structure
```

```
        txt = txt + "directory name: %s" % info['name'] + "\n"
```

```
        txt = txt + "files.....: " + "\n"
```

```
        file_length = 0
```

```
        for file in info['files']:
```

```
            path = ""
```

```
            for item in file['path']:
```

```
                if (path != ""):
```

```
                    path = path + "/"
```

```
                    path = path + item
```

```
                txt = txt + " %s (%d)" % (path, file['length']) + "\n"
```

```
                file_length += file['length']
```

```
                name = 'archive size..: '
```

```
    piece_number, last_piece_length = divmod(file_length, piece_length)
```

```
    txt = txt + "%s %i (%i * %i + %i)" \
```

```
        % (name, file_length, piece_number, piece_length, last_piece_length) + "\n"
```

```
    txt = txt + "announce url..: %s" % metainfo['announce'] + "\n"
```

```
    if metainfo.has_key('announce-list'):
```

```
        list = []
```

```
        for tier in metainfo['announce-list']:
```

```
            for tracker in tier:
```

```
                list+=[tracker,',']
```

```
            del list[-1]
```

```
            list+=['|']
```

```
        del list[-1]
```

```
        liststring = ""
```

```
        for i in list:
```

```
            liststring+=i
```

```
        txt = txt + "announce-list.: %s" % liststring + "\n"
```

```
    if metainfo.has_key('httpseeds'):
```

```
        list = []
```

```
        for seed in metainfo['httpseeds']:
```

```
            list += [seed, '|']
```

```
        del list[-1]
```

```
        liststring = ""
```

```
        for i in list:
```

```
            liststring+=i
```

```
        txt = txt + "http seeds.....: %s" % liststring + "\n"
```

```
    if metainfo.has_key('comment'):
```

```
        txt = txt + "comment.....: %s" % metainfo['comment'] + "\n"
```

```

        buf = self.metainfo_textview.get_buffer()
        buf.set_text(txt)
    #-- InfoWindow.new }

    #-- InfoWindow custom methods {
    # Write your own methods here
    #-- InfoWindow custom methods }

```

Maintenant pour finir le programme, nous devons définir la fonction `get_xisp_id()` qui va retourner l'url associée au nom fournit en paramètre. Puis, il y a le point d'entrée du script python : la fonction `main` d'où l'on va lancer la fenêtre principale.

```

def get_xisp_id(identifiant):
    fdpn = open("bitpodder.fwp").readlines()
    reqxisp = re.compile(identifiant)
    for pe in fdpn:
        if reqxisp.search(pe):
            PeersString = pe.split("?:?:")
    return PeersString[1]

#-- main {

def main():
    bitpodder_window = BitPodderWindow()

    bitpodder_window.run()

if __name__ == "__main__":
    main()

#-- main }

```

Maintenant, essayer de lancer `bitpodder.py` après avoir créé un fichier `bitpodder.fwp` dans le même répertoire que `bitpodder.py`. BitPodder fonctionne !!!

VIII] Comment compiler BitPodder?

Pour compiler un script python sous windows, il faut créer un script `setup.py`. Ce script utilisera les packages `distutils` et `py2exe`. Avant d'écrire ce script, vous devez télécharger et installer `py2exe` (<http://sourceforge.net/projects/py2exe>).

Si l'installation s'est bien déroulée, il faut créer le script `setup.py`. Il y a une bon article expliquant comment créer de tel scripts à cette adresse : <http://www.python.org/doc/current/dist/setup-script.html>, mais cet article n'explique pas comment créer des `.exe` avec `py2exe`, je vais donc vous donner le script que j'ai utilisé pour compiler BitPodder.

```

#!/usr/bin/env python
# setup.py
from distutils.core import setup
import py2exe
import glob

opts = {
    "py2exe": {
        "includes": "pango,atk,gobject",
        "dll_excludes": [
            "iconv.dll","intl.dll","libatk-1.0-0.dll",

```

```

"libgdk_pixbuf-2.0-0.dll", "libgdk-win32-2.0-0.dll",
"libglib-2.0-0.dll", "libgmodule-2.0-0.dll",
"libgobject-2.0-0.dll", "libgthread-2.0-0.dll",
"libgtk-win32-2.0-0.dll", "libpango-1.0-0.dll",
"libpangowin32-1.0-0.dll"],
}
}
}

setup(
    name = "BitPodder",
    description = ".torrent podcaster from Nzeka Labs",
    version = "0.1",
    author="Nzeka Gilbert",
    author_email="khaalel@gmail.com",
    url="http://www.nzeka-labs.com",
    windows = [
        {"script": "bitpodder.py",
         "icon_resources": [(1, "sage.ico")]
        }
    ],
    options=opts,
    data_files=[("pixmaps", glob.glob("pixmaps/*.png")),
                ("glade", glob.glob("glade/*.*.")
    ],
)

```

Avant la compilation, n'oubliez pas de placer l'image sage.ico dans le répertoire où se trouve bitpodder.py où vous allez obtenir un beau message d'erreur. Si la compilation s'est achevée avec succès, vous devriez avoir 2 nouveaux répertoires : \dist\ et \build\. Vous pouvez supprimer le répertoire \build\, mais conservez le répertoire \dist\ car l'exécutable y est, ainsi que toutes les dlls nécessaires au bon fonctionnement du programme.

Ah oui, n'oubliez pas de mettre bitpodder.fwp dans le répertoire \dist\ avant de lancer l'exécutable.

Si vous voulez créer un installateur, je vous conseille d'utiliser 7-zip ou NSIS (des créateurs de Winamp). Ici est le script que j'ai utilisé avec zip2exe (du menu de NSIS) :

```

;Change this file to customize zip2exe generated installers with a modern interface

!include "MUI.nsh"

!insertmacro MUI_PAGE_DIRECTORY
!insertmacro MUI_PAGE_INSTFILES

!insertmacro MUI_LANGUAGE "English"

Section -post
SetOutPath $INSTDIR
CreateShortCut "$DESKTOP\BitPodder.lnk" "$INSTDIR\bitpodder.exe"
CreateDirectory "$SMPROGRAMS\Nzeka Labs"
CreateDirectory "$SMPROGRAMS\Nzeka Labs\BitPodder"
CreateShortCut "$SMPROGRAMS\Nzeka Labs\BitPodder\BitPodder.lnk" "$INSTDIR\bitpodder.exe"
""
WriteINIStr "$SMPROGRAMS\Nzeka Labs\Nzeka Labs.url" "InternetShortcut" "URL"
"http://www.nzeka-labs.com/"
WriteINIStr "$SMPROGRAMS\Nzeka Labs\Xendor Site.url" "InternetShortcut" "URL"
"http://korriban-planet.blogspot.com/"
SectionEnd

```

Pour comprendre le script, je vous conseille vivement de lire l'aide de NSIS.

Cet article est fini. Si vous l'avez trouvé intéressant ou utile, faites-le moi savoir par mail. Si par contre vous l'avez trouvé nul/ennuyant, contactez-moi aussi en m'envoyant vos suggestions pour que j'améliore l'article.

END