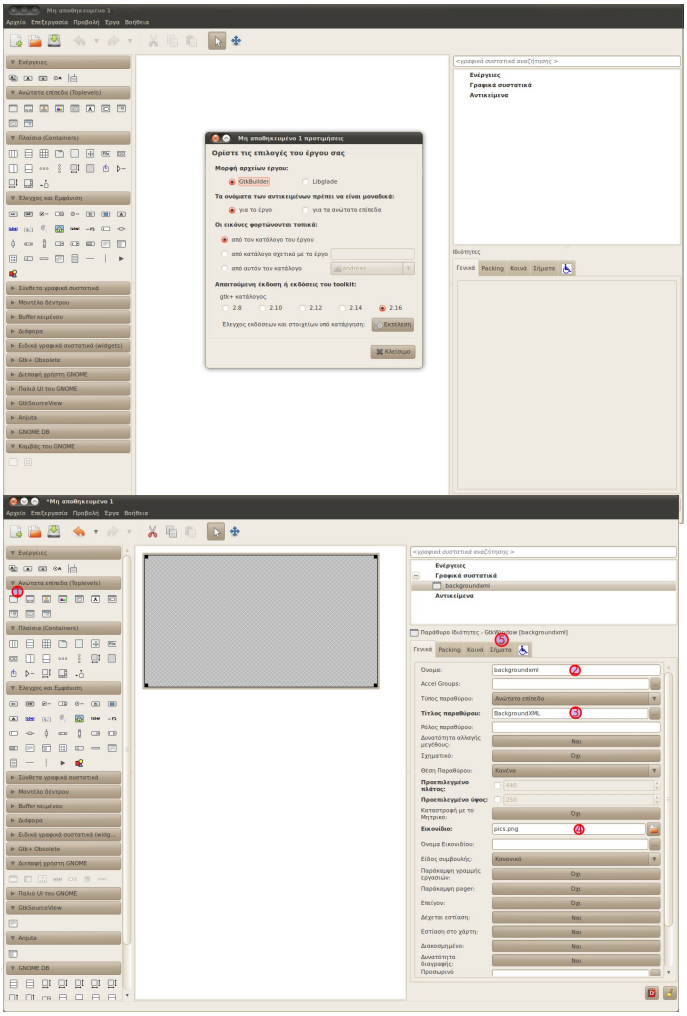
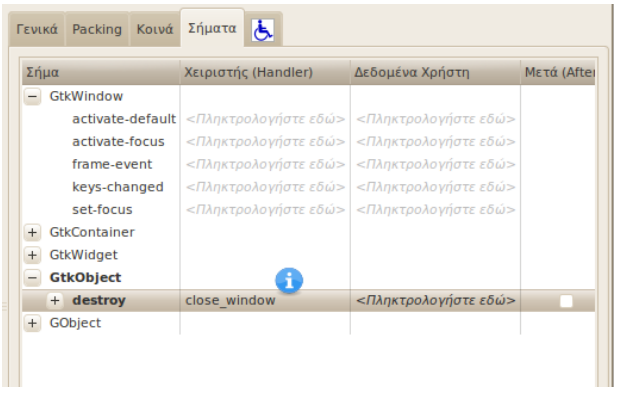
Το Glade είναι μια εφαρμογή για την δημιουργία του UI ενός προγράμματος σε GTK. Η μεθοδολογία που ακολουθείται είναι ο σχεδιασμός με την προσθήκη διάφορων αντικειμένων (παράθυρα, button, menu κλπ), η τακτοποίηση αυτών στο παράθυρο, η μεταβολή των ιδιοτήτων τους (χρώμα, μέγεθος, συμπεριφορά κλπ) και η σύνδεση των σημάτων που στέλνουν, σε απάντηση ενέργειας του χρήστη, με ρουτίνες του προγράμματος (πχ όταν πατήσει ο χρήστης το X κλεισίματος στο παράθυρο τι θα κάνει το πρόγραμμα).

Το Glade φτιάχνει ένα αρχείο XML όπου περιγράφει όλα τα παραπάνω και είναι δουλειά του προγράμματος να το διαβάσει, να δημιουργήσει τα αντικείμενα και να τα εμφανίσει. Υπάρχουν βιβλιοθήκες που κάνουν ακριβώς αυτό για όλες της σημαντικές γλώσσες C++, perl, python, ruby. Εδώ θα ασχοληθούμε με την python, μια πολύ διαδεδομένη γλώσσα με βιβλιοθήκες και προγράμματα για ότι μπορεί κανείς να φανταστεί. Θα γράψουμε ένα πρόγραμμα για την εύκολη δημιουργία του αρχείου XML εναλλαγής φόντων που σχολιάσαμε σε προηγούμενο άρθρο.

Όταν ξεκινάει το Glade παρουσιάζεται το εξής παράθυρο όπου ζητάει ορισμένα στοιχεία για να ξεκινήσει την δημιουργία του έργου. Στο “Μορφή έργου” διαλέγουμε το GtkBuilder γιατί το libglade είναι προς κατάργηση, και επιλέγουμε την μεγαλύτερη έκδοση toolkit που είναι διαθέσιμη. Το συγκεκριμένο Glade είναι το 3.6.7 από την διανομή Ubuntu Lucid και έχει την 2.16 σαν μεγαλύτερη. Στην Natty διανομή υπάρχει το Glade 3.8 και toolkit 2.24. Αφήνουμε τα υπόλοιπα προεπιλεγμένα χαρακτηριστικά και πατάμε “Κλείσιμο”.



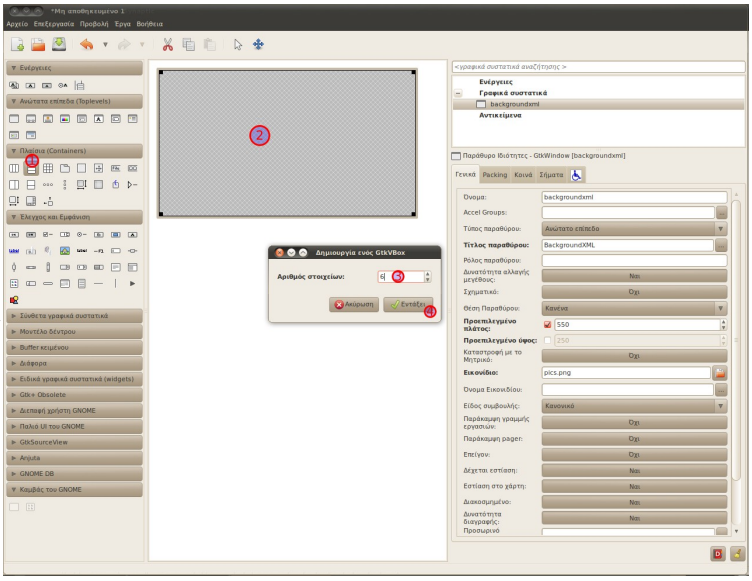
1. Για αρχή επιλέγουμε “Παράθυρο” το οποίο είναι το βασικό αντικείμενο το οποίο περιέχει αυτά που θέλουμε να εμφανίσουμε.
2. Αλλάζουμε το όνομα του σε “backgroundxml” για να υπάρχει μια ενιαία ονοματολογία.
3. Ο τίτλος του παραθύρου: BackgroundXML είναι αυτό που θα εμφανίζεται πάνω στην μπάρα όταν τρέχει το πρόγραμμα.
4. Βάζουμε ένα γραφικό το οποίο θα εμφανίζεται σαν γραφικό του προγράμματος στο task bar
5. Πατάμε το tab “Σήματα” για να δώσουμε το όνομα της ρουτίνας που θα καλείται όταν ο χρήστης πατήσει το X κλεισίματος παραθύρου.



Πατώντας το X το Gtk στέλνει το σήμα destroy για την επικείμενη καταστροφή του παραθύρου.
Στο destroy του GtkObject δίνουμε σαν handler το close_window. Μην ανησυχείτε για αυτό ακόμα, θα γράψουμε την ρουτίνα αυτή μετά στο πρόγραμμα μας. Μπορείτε να της δώσετε ότι όνομα θέλετε αρκεί να χρησιμοποιήσετε το ίδιο και στο πρόγραμμα και στον handler.
Στην συνέχεια θα γεμίσουμε το παράθυρο μας με τα

αντικείμενα που χρειαζόμαστε για το πρόγραμμα μας. Όταν ξεκινάμε μια γραφική εφαρμογή καλό θα είναι να έχουμε αποφασίσει τι αντικείμενα θέλουμε και να έχουμε σχεδιάσει το πως θα θέλαμε αυτά να εμφανιστούν στο παράθυρο. Στο Glade κάθε αντικείμενο καταλαμβάνει όλο τον χώρο μέσα στον container που βρίσκεται. Έτσι αν σε άδειο παράθυρο προσθέσουμε για παράδειγμα ένα label, αυτό θα καταλάβει όλο τον χώρο και δεν θα μπορούμε να προσθέσουμε τίποτα άλλο. Γι' αυτό και τα δύο πιο χρήσιμα container είναι το Vbox και το Hbox, κάθετος και οριζόντιος πίνακας αντίστοιχα.

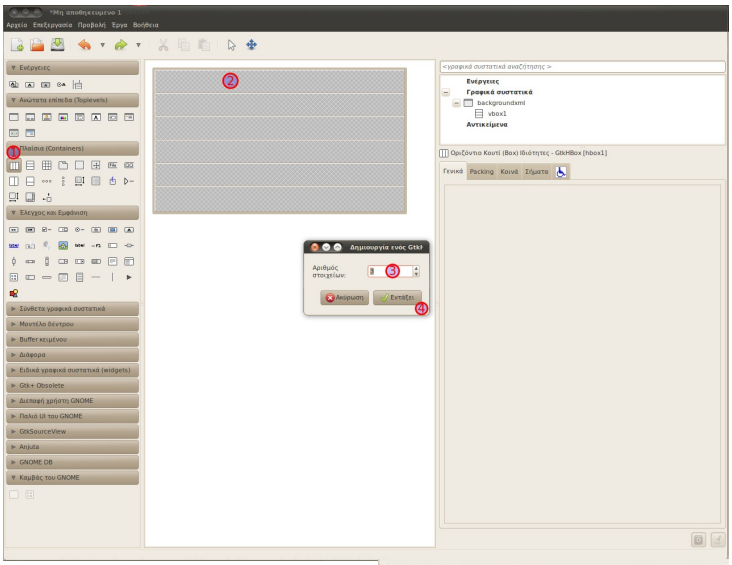
1. Ξεκινάμε με ένα Vbox. Κλικ στο εικονίδιο του και
2. Κλικ στο παράθυρο
3. Επιλέγουμε πόσες γραμμές θα έχει ο πίνακας, 6 εν προκειμένω, και
4. Κλικ στο “Εντάξει”



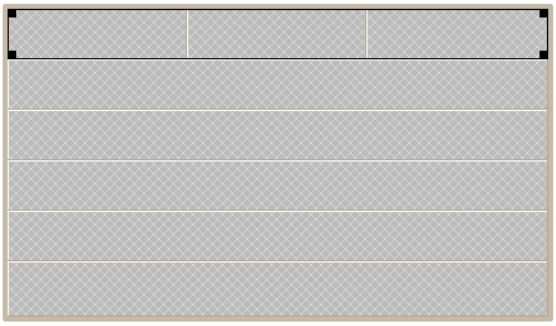
Έχουμε τώρα το παράθυρο μας χωρισμένο σε 6 τμήματα. Σε καθένα από αυτά μπορούμε να προσθέσουμε ότι θέλουμε και αυτό θα καταλάβει μόνο το αντίστοιχο τμήμα.

Ξεκινάμε με το πάνω τμήμα να βάλουμε έναν οριζόντιο πίνακα τριών στηλών.

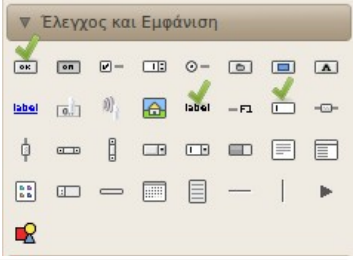
1. Κλικ στο Hbox
2. Κλικ στο πάνω τμήμα
3. Δίνουμε 3 στήλες
4. Πατάμε “Εντάξει”



Το παράθυρο μας έχει τώρα την παρακάτω μορφή.

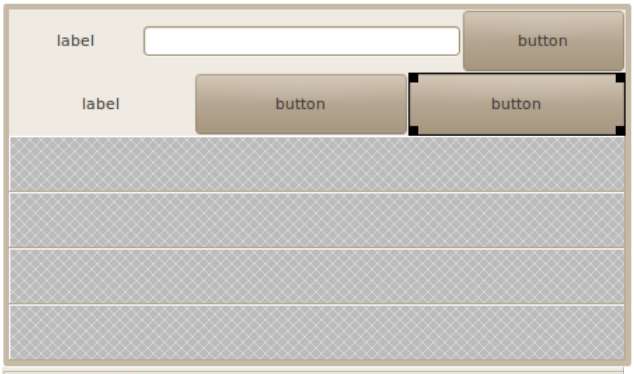


Στην συνέχεια επιλέγουμε ένα label για την πρώτη στήλη, ένα πεδίο εισόδου για την δεύτερη και ένα button για την τρίτη.

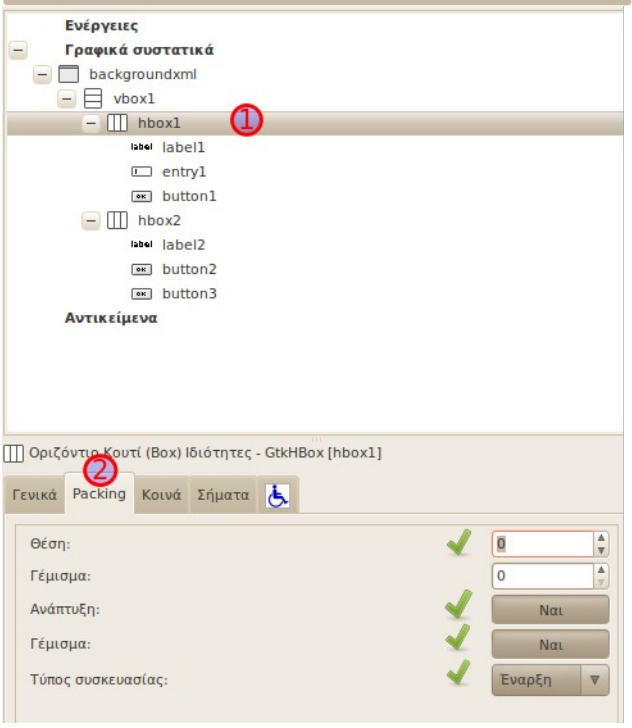


Η πρώτη γραμμή μας πρέπει τώρα να έχει αυτή την μορφή

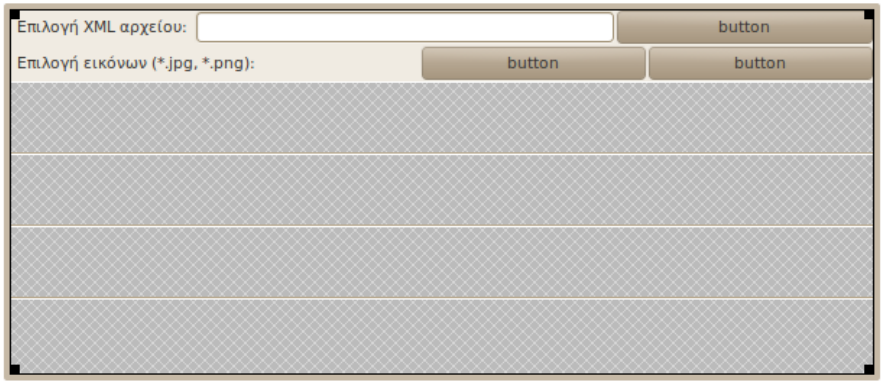
Στην δεύτερη γραμμή προσθέτουμε έναν οριζόντιο πίνακα 3 τμημάτων. Στο πρώτο τμήμα βάζουμε ένα label, στο δεύτερο και τρίτο τμήμα από ένα button. Τώρα το παράθυρο μας έχει την εξής μορφή:



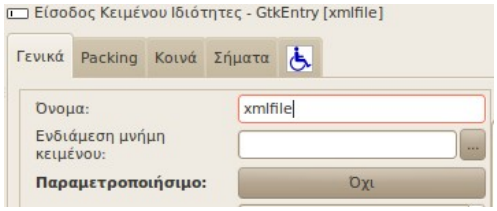
Πριν συνεχίσουμε λίγα λόγια για το πως διαχειρίζεται την τοποθέτηση και το μέγεθος των αντικείμενων το GTK. Κάνουμε κλικ στο hbox1 και μετά στο tab “packing”. Η σειρά που θα εμφανιστεί κάθε αντικείμενο μέσα στο container του, καθορίζεται από την τιμή “Θέση”. Για το hbox1 έχει τιμή 0 γιατί το βάλαμε πρώτο στο vbox1 και γι' αυτό φαίνεται στην πρώτη γραμμή. Αν το αλλάξουμε και το κάνουμε 1 ή 2 ή 3 θα πάει στην αντίστοιχη γραμμή του vbox1. Το ίδιο ισχύει και για τα αντικείμενα του hbox1, τα label1, entry1, button1. Έχουν κι αυτά αντίστοιχα τιμές 0,1,2. Τα παρακάτω πεδία “Ανάπτυξη” και “Γέμισμα” που έχουν τιμή “Ναι” καθορίζουν το πως θα συμπεριφερθεί το αντικείμενο όταν ο χρήστης αλλάξει μέγεθος στο παράθυρο. Το “Ανάπτυξη” καθορίζει αν θα μεγαλώσει όσο μπορεί για να καλύψει τον έξτρα χώρο και το “Γέμισμα” αν θα γεμίσει ο χώρος με το αντικείμενο ή θα παραμείνει κενός. Ο “τύπος συσκευασίας”έχει τιμή “Έναρξη”, σημαίνει ότι η “Θέση”μετράει από την αρχή του container. Αν το αλλάξετε σε “τέλος” το hbox1 θα πάει στην τελευταία γραμμή και θα παραμείνει σε αυτήν ακόμα και αν εμείς προσθαφαιρέσουμε γραμμές στο vbox1. Το πεδίο “Γέμισμα” που δέχεται αριθμητική τιμή είναι πόσα pixel γύρω γύρω από το αντικείμενο θα υπάρχουν σαν περιθώριο του.



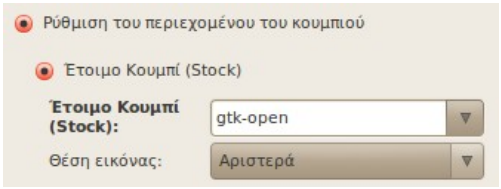
Αν κοιτάξουμε το “Packing” tab στα αντικείμενα μας θα δούμε ότι όλα έχουν “Ανάπτυξη” και “Γέμισμα” “Ναι”. Βάση των παραπάνω φανταζόμαστε ότι αν ο χρήστης μεγαλώσει το παράθυρο τα buttons θα γίνονται όλο και πιο χοντρά και μακριά το ίδιο και τα labels. Το label2 δεν είναι στοιχισμένο κάτω από το label1 και όλο θα έρχεται πιο δεξιά αφού θα μεγαλώνει ο χώρος του. Μόνο το entry1 θέλουμε να μεγαλώνει για να δείχνει ολόκληρο κάποιο μεγάλο όνομα.
Πάμε λοιπόν να τα διαμορφώσουμε όπως θέλουμε.
Για αρχή κάνουμε κλικ στο παράθυρο μας και επιλέγουμε το tab “Κοινά” και βάζουμε τιμή 550 στην “Αίτηση πλάτους”. Αυτό θα μεγαλώσει όλο το παράθυρο μας και μπορεί να χαθεί τμήμα του πίσω από την δεξιά στήλη με τα αντικείμενα γι΄ αυτό ρυθμίζουμε ανάλογα το πλάτος της.
Στα hbox1 και hbox2 βάζουμε “Όχι” σε “Ανάπτυξη” και “Γέμισμα”. Στο label1 βάζουμε “Όχι” στην “Ανάπτυξη” και 5 στο “Γέμισμα”. Στο tab “Γενικά” βάζουμε 0 στην “Στοίχιση Χ” και σαν “Ετικέτα” βάζουμε: “Επιλογή XML αρχείου:”
Στο label2: “Ανάπτυξη = Ναι”, “Γέμισμα = 5”, “Στοίχιση Χ = 0” και “Ετικέτα” = “Επιλογή εικόνων (*.jpg, *.png):”
Θα πρέπει τώρα το παράθυρο να έχει την εξής μορφή:



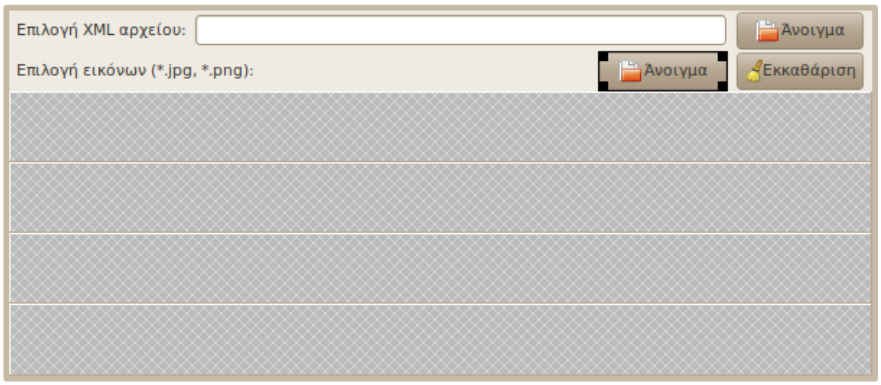
Στην συνέχεια αλλάζουμε το όνομα του entry1 σε xmlfile και αλλάζουμε το πεδίο “Παραμετροποιήσιμο” σε “Όχι” για να μην μπορεί ο χρήστης να γράψει με το χέρι, θα παίρνει τιμή από παράθυρο επιλογής αρχείων.



Τα κουμπιά θέλουμε να είναι ομοιόμορφα για αυτό σε όλα στο tab “Κοινά” βάζουμε “Αίτηση πλάτους” 90 και στο tab “Packing” κάνουμε την “Ανάπτυξη” “Όχι”. Το button1 το μετονομάζουμε σε xmlbutton στο tab “Γενικά” και επιλέγουμε “Ετοιμο κουμπί (stock)” το gtk-open.



Επιλέγουμε gtk-open και στο button2 και το μετονομάζουμε σε “picbutton”. Το button3 το ονομάζουμε “clearbutton” και σαν περιεχόμενο διαλέγουμε το έτοιμο gtk-clear.
Μετά από αυτά πρέπει να έχουμε την εξής μορφή:



Να σώσουμε τώρα το αρχείο μας και να φτιάξουμε ένα πρόγραμμα σε python για να το εμφανίσουμε και να τεστάρουμε τις μεταβολές μεγεθών.
Σώζουμε το αρχείο σε έναν φάκελο της αρεσκείας μας με το όνομα backgroundxml.glade.
Σε έναν κειμενογράφο γράφουμε το εξής πρόγραμμα:

```
import gtk

class BackgroundXML(object):

    def close_window(self, widget, data=None):
        gtk.main_quit()

    def __init__(self):
        builder = gtk.Builder()
        builder.add_from_file("backgroundxml.glade")
        builder.connect_signals(self)
        self.backgroundxml = builder.get_object("backgroundxml")

    def main(self):
        self.backgroundxml.show()
        gtk.main()

if __name__ == "__main__":
    app = BackgroundXML()
    app.main()
```

Το σώζουμε στον ίδιο φάκελο με το αρχείο glade, με όνομα backgroundxml.py
Το τρέχουμε από την γραμμή εντολών με: python backgroundxml.py
Το αποτέλεσμα είναι:

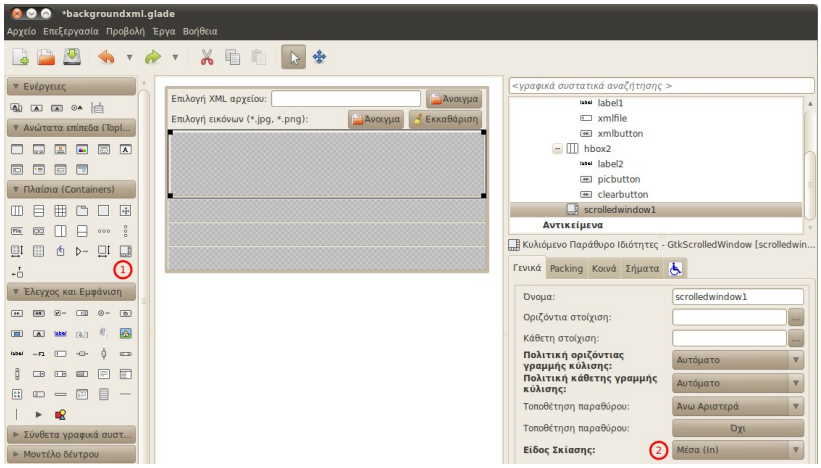
Όπως βλέπουμε οι κενές γραμμές του vbox1 δεν εμφανίζονται και αν το μεγαλώσουμε τα στοιχεία μας παραμένουν ίδια σε μέγεθος και θέση, έκτος από το πεδίο εισαγωγής, και ο υπόλοιπος χώρος παραμένει κενός.

Με λίγες γραμμές κώδικα έχουμε έτοιμο το γραφικό μας περιβάλλον.
Λίγα λόγια για το πρόγραμμα.
Κάνουμε import το απαραίτητο gtk. Η κλάση BackgroundXML είναι η βασική του προγράμματος, η οποία δημιουργεί το αντικείμενο app και στην συνέχεια καλούμε την main της κλάσης.
Για την δημιουργία του app η python χρησιμοποιεί την __init__ μέθοδο της κλάσης στην οποία φορτώνουμε το αρχείο glade με τον builder και συσχετίζουμε τα σήματα που έχουμε ορίσει με τις αντίστοιχες μεθόδους (builder.connect_signals(self)). Την μεταβλητή self.backgroundxml την συσχετίζουμε με το αντικείμενο backgroundxml το οποίο είναι το βασικό μας παράθυρο.
Οπότε όταν καλούμε self.backgroundxml.show() στην main εμφανίζεται το παράθυρο μας και με την gtk.main() ο έλεγχος περνάει στο παράθυρο μέχρι αυτό να κλείσει. Το παράθυρο κλείνει όταν ο χρήστης πατήσει το X (μιας και δεν έχουμε βάλει button τέλους) και στέλνεται το σήμα destroy το οποίο έχουμε συνδέσει με την close_window η οποία καλεί την gtk.main_quit(). Ο έλεγχος επιστρέφει στο πρόγραμμα και αυτό, επειδή δεν έχει άλλη εντολή μετά την app.main(), τερματίζει.

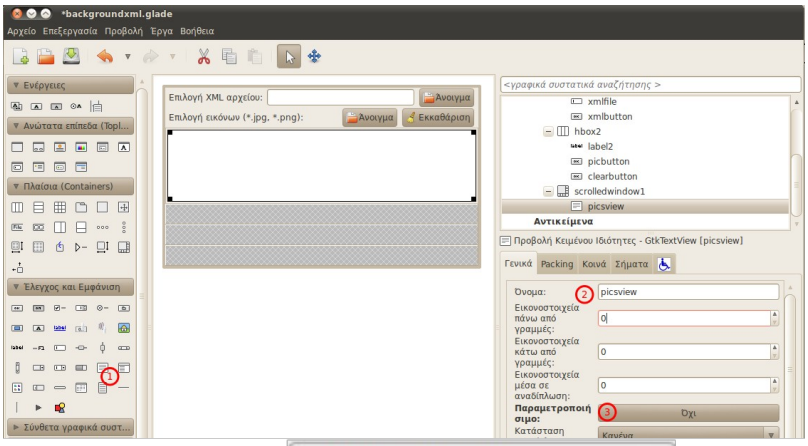
Αυτός είναι ο βασικός κορμός ενός gtk προγράμματος. Το τελικό πρόγραμμα θα έχει πιο πολλά σήματα, μεθόδους και μεταβλητές αλλά πήραμε τώρα μια ιδέα για το πως γίνεται η διαδικασία. Μπορείτε να πειραματιστείτε με τα αντικείμενα που έχουμε φτιάξει ήδη, αποκτώντας έτσι περισσότερη εμπειρία με το Glade και τις δυνατότητες του. Στο επόμενο άρθρο θα τελειώσουμε τον σχεδιασμό του παραθύρου μας και θα γνωρίσουμε νέα αντικείμενα.

Σε αυτό το άρθρο θα συνεχίσουμε την γνωριμία μας με το Glade και τον τρόπο δημιουργίας ενός UI στο GTK. Θα ολοκληρώσουμε την κατασκευή του βασικού μας παράθυρου προσθέτοντας ένα πεδίο το οποίο θα μας δείχνει τις εικόνες που έχει διαλέξει ο χρήστης για την εναλλαγή του φόντου, δύο πεδία στα οποία θα δίνει τις τιμές για τον χρόνο εμφάνισης και εναλλαγής κάθε εικόνας, και στο τέλος δύο κουμπιά εφαρμογής και κλεισίματος.

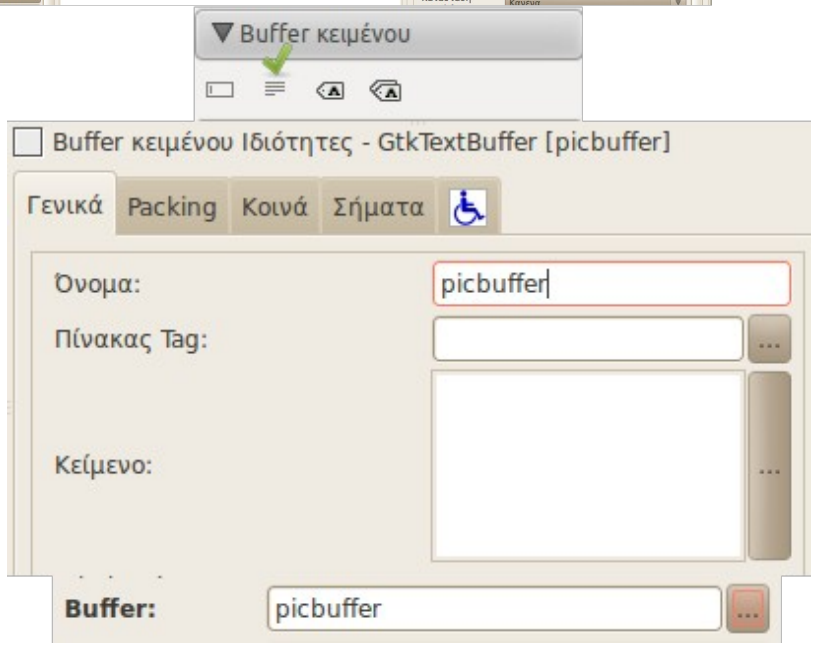
Ξεκινώντας από εκεί που είχαμε μείνει στο προηγούμενο άρθρο, προσθέτουμε ένα scrolled window στην τρίτη γραμμή και αλλάζουμε το “Είδος σκίασης” σε “Μέσα”. Αυτό είναι καθαρά αισθητικό και μπορείτε να διαλέξετε ότι αρέσει σ' εσάς. Το scrolled window είναι ο περιέκτης μέσα στον οποίο βάζουμε οποιοδήποτε αντικείμενο θέλουμε να κάνει κύλιση, αλλιώς το αντικείμενο από μόνο του δεν κάνει.



Στην συνέχεια προσθέτουμε μέσα στο κυλιόμενο παράθυρο ένα αντικείμενο “Προβολής κειμένου” (textview), αλλάζουμε το όνομα του σε picsview και κάνουμε την ιδιότητα του “Παραμετροποιήσιμο” “Όχι”. Εδώ θα εμφανίζουμε μια λίστα με τα ονόματα των εικόνων που έχει διαλέξει ο χρήστης προς προβολή.



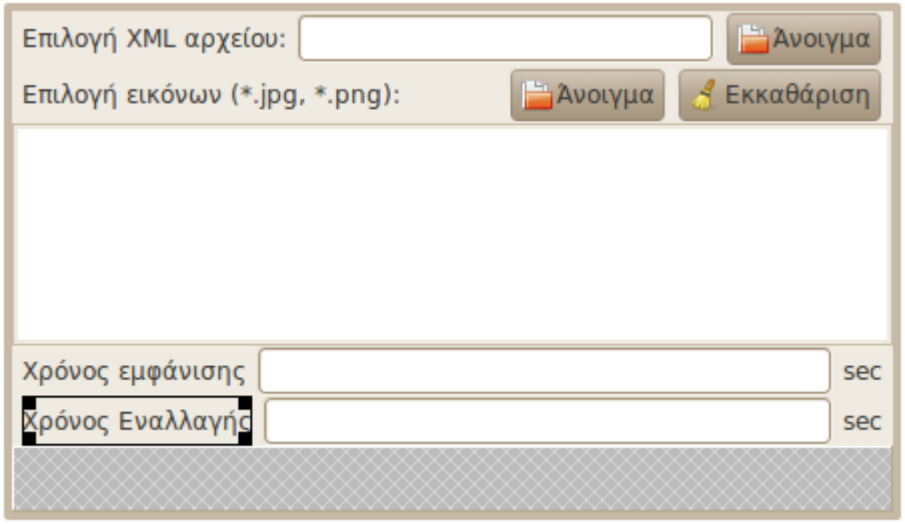
Για να μπορέσουμε να γράψουμε στο textview αντικείμενο, μέσα από το πρόγραμμα μας, θα πρέπει να χρησιμοποιήσουμε ένα text buffer. Διαλέγουμε λοιπόν το αντίστοιχο εικονίδιο και το ονομάζουμε picbuffer. Στην συνέχεια στο tab “Γενικά” του picsview, στο πεδίο “Buffer” διαλέγουμε αυτό που μόλις δημιουργήσαμε.



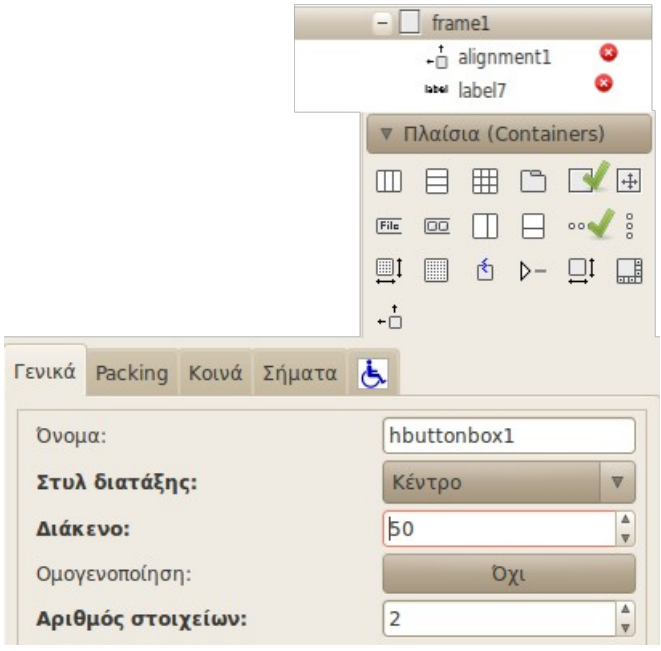
Στην τέταρτη γραμμή βάλτε ένα HBox 3 θέσεων. Διαλέγουμε “Όχι” για το “Γέμισμα” και το “Ανάπτυξη” του. Στη μεσαία θέση βάζουμε ένα πεδίο εισόδου και αριστερά και δεξιά από ένα label. Στην “Ετικέτα” του αριστερού βάζουμε το λεκτικό: “Χρόνος εμφάνισης” και στο δεξί “sec”. Και των δύο τα “Γέμισμα” και “Ανάπτυξη” τα κάνουμε “Όχι”. Και στα δύο δίνουμε αριθμητικό “Γέμισμα” ίσο με 5. Αυτό για να έχουν ένα περιθώριο από τα όρια του παραθύρου. Το πεδίο εισόδου το μετονομάζουμε “duration”.

Τα ίδια ακριβώς κάνουμε και για την 5 γραμμή. Οι μόνες διαφορές είναι το λεκτικό στο αριστερό label: “Χρόνος εναλλαγής” και το όνομα του πεδίου: change

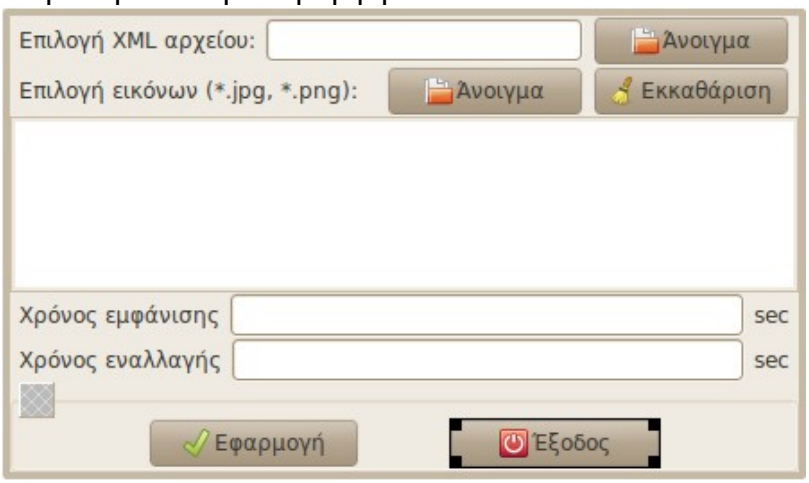
Τώρα το παράθυρο μας έχει την εξής μορφή:



Στην τελευταία γραμμή θα βάλουμε πρώτα ένα frame. Το frame δημιουργεί δύο δικά του αντικείμενα, τα οποία και δεν τα χρειαζόμαστε, ένα alignment και ένα label. Σβήστε τα και τα δύο. Επιλέξτε σαν “Σκίαση πλαισίου” το “Μέσα”. Στα “Γέμισμα” και “Ανάπτυξη” επιλέγουμε κι εδώ “Όχι”. Αυτό το προσθέσαμε καθαρά για αισθητικούς λόγους, για να περικλείονται τα παρακάτω κουμπιά σε ένα σκιασμένο πλαίσιο και να ξεχωρίζουν.
Στην συνέχεια προσθέστε μέσα στο frame ένα οριζόντιο button box. Ορίστε δύο τον αριθμό των στοιχείων και αλλάξτε το “Στυλ διάταξης” σε “Κεντρο” και στο “Διάκενο” βάλτε τιμή 50.

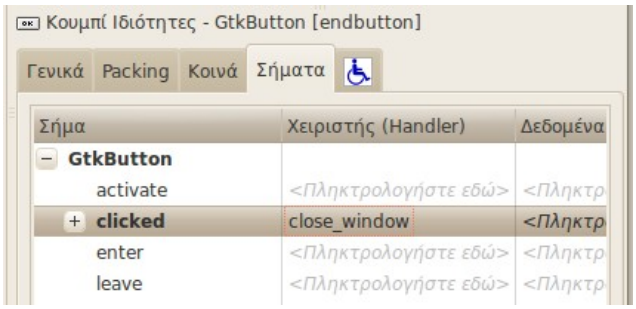


Στις δύο θέσεις που έχουν δημιουργηθεί προσθέτουμε δύο κουμπιά. Διαλέγουμε για το πρώτο από το stock το κουμπί gtk-apply και το ονομάζουμε okbutton και για το άλλο επιλέγουμε το gtk-quit και το ονομάζουμε endbutton.
Το παράθυρό μας έχει τώρα την τελική του μορφή:



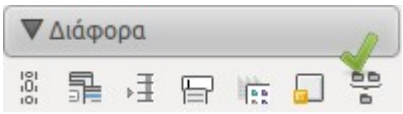
Θα ήθελα να αναφέρω ότι η εμφάνιση του παραθύρου μπορεί να διαφέρει από σύστημα σε σύστημα. Για παράδειγμα, αφού δεν έχουμε ασχοληθεί καθόλου με της γραμματοσειρές του παραθύρου, το GTK δείχνει αυτές που έχει σαν προεπιλογή για όλο το Gnome περιβάλλον. Οπότε είναι αναλόγως τι έχει επιλέξει ο καθένας για το σύστημά του. Αλλά και το θέμα, τα χρώματα ακόμα και τα εικονίδια μπορεί να διαφέρουν. Η λειτουργία και η συμπεριφορά όμως των αντικειμένων παραμένουν ίδιες.

Στην συνέχεια θα ορίσουμε τις μεθόδους που θα εκτελούνται όταν πατήσουμε το αντίστοιχο κουμπί. Στο tab “Σήματα” του κουμπιού endbutton επιλέγουμε το σήμα clicked και γράφουμε σαν handler την close_window που έχουμε ήδη χρησιμοποιήσει στο σήμα destroy του παραθύρου.



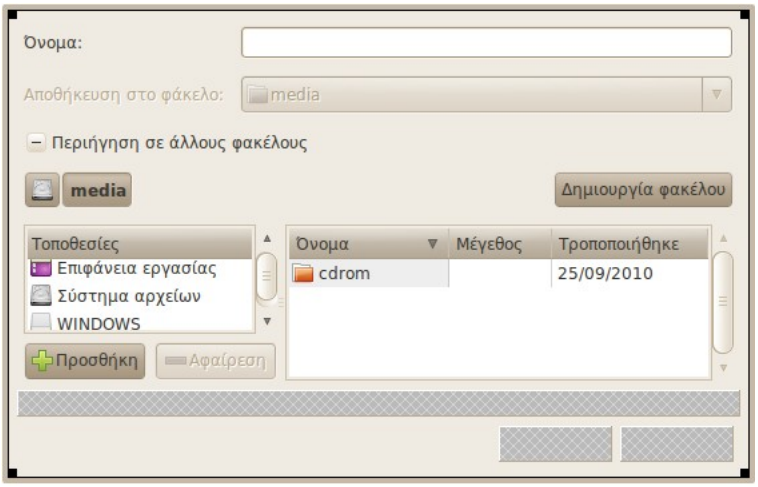
Στο clicked του okbutton δίνουμε clicked_ok, στο picbutton picbutton_click, στο clearbutton clearbutton_click και στο xmlbutton xmlbutton_click.

Όταν πατηθεί το κουμπί xmlbutton θέλουμε να ανοίγει ένα παράθυρο επιλογής αρχείων για να διαλέξει ο χρήστης που θα σώσει και με τι όνομα το XML αρχείο. Αναλόγως, όταν πατηθεί το picbutton θέλουμε να μπορεί ο χρήστης να επιλέξει τις εικόνες που θέλει να εμφανίζονται στο φόντο του. Το clearbutton θα καθαρίζει το picsview από όλα τα αρχεία.

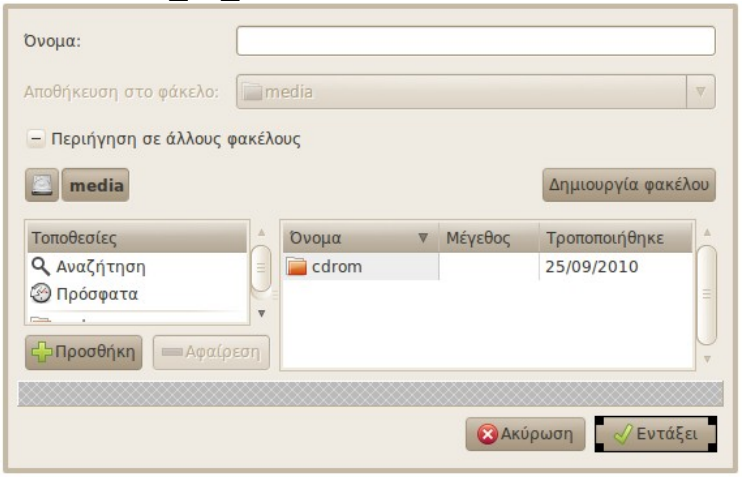


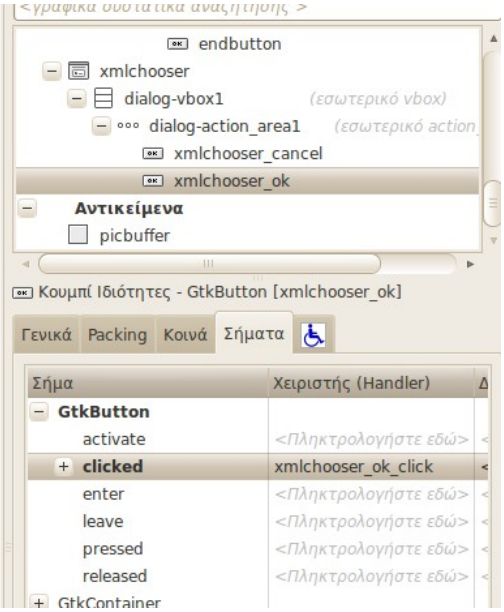
Χρειάζεται να δημιουργήσουμε δύο ακόμα αντικείμενα, δύο φίλτρα, όπου θα χρησιμοποιηθούν στα παράθυρα επιλογής αρχείων. Για το πρώτο θέλουμε να μας δείχνει μόνο τα .xml αρχεία και για το δεύτερο μόνο τα .jpg και .png. Από τα “Διάφορα” επιλέγουμε δύο φορές “Φιλτρο αρχείων” και ονομάζουμε το πρώτο xmlfilter και το δεύτερο picfilter.

Από τα “Ανώτατα επίπεδα” κάνουμε κλικ στον Διάλογο Επιλογής Αρχείου. Αυτό θα δημιουργήσει ένα καινούργιο παράθυρο με το όνομα filechooser1. Αλλάζουμε το όνομα του σε xmlchooser. Βάζουμε στον τίτλο παραθύρου: “Επιλέξτε αρχείο XML προς εγγραφή”. Αλλάζουμε στο τέλος την “Ενέργεια” του να γίνει “Αποθήκευση”. Για “Φιλτρο” επιλέγουμε το xmlfilter που δημιουργήσαμε προηγουμένως.



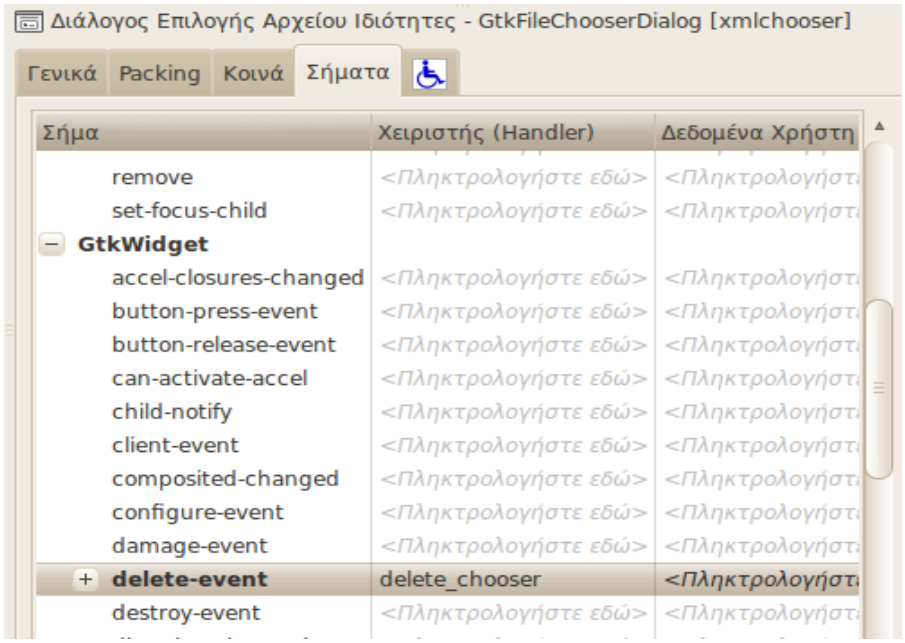
Στην συνέχεια προσθέτουμε δύο κουμπιά στις αντίστοιχες ελεύθερες θέσεις του. Ένα gtk-cancel (Ακύρωση) και ένα gtk-ok (Εντάξει). Τους δίνουμε αντίστοιχα τα ονόματα xmlchooser_cancel και xmlchooser_ok. Στα clicked σήματα τους δίνουμε σαν handler αντίστοιχα xmlchooser_cancel_clicked και xmlchooser_ok_clicked.



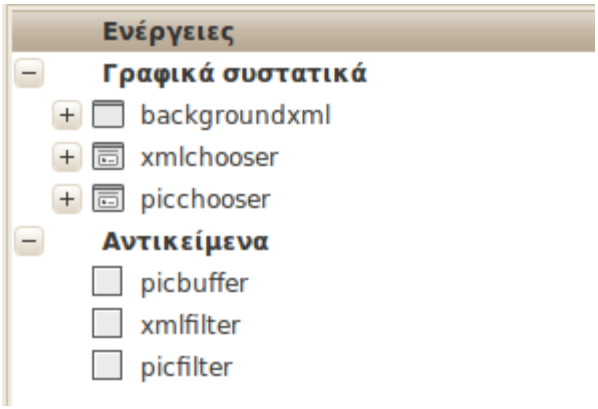


Έχουμε τώρα έτοιμο το πρώτο βοηθητικό παράθυρο επιλογής XML αρχείου. Θα ανοίγει όταν ο χρήστης πατήσει το πρώτο κουμπί “Ανοιγμα”, θα επιλέγει αρχείο προς εγγραφή και αναλόγως αν έχει πατήσει Εντάξει ή Ακύρωση το πρόγραμμα θα καταχωρεί ή όχι αυτήν την επιλογή και θα κλείνει το παράθυρο.

Το παράθυρο μπορεί να το κλείσει όμως ο χρήστης πατώντας και το X που υπάρχει σε μία από τις πάνω γωνίες του. Πατώντας το, το GTK καταστρέφει το παράθυρο και αν ο χρήστης ξαναπατήσει “Άνοιγμα” τότε εμφανίζεται λάθος γιατί το πρόγραμμα προσπαθεί να δείξει ένα αντικείμενο που δεν υπάρχει. Για να αποφευχθεί αυτή η περίπτωση δίνουμε στο σήμα “delete-event” του xmlchooser έναν handler για μια μέθοδο του προγράμματός μας, η οποία το μόνο που θα κάνει είναι να επιστρέφει True. Ας την ονομάζουμε delete_chooser. Γιατί στο delete-event και όχι στο destroy-event; Γιατί το delete προηγείται του destroy και επιστρέφοντας True δεν καλείται ποτέ η αντίστοιχη destroy συνάρτηση του GTK.



Με τον ίδιο τρόπο δημιουργούμε κι ένα δεύτερο παράθυρο επιλογής αρχείων. Τα χαρακτηριστικά του θα είναι: Όνομα: picchooser, Τίτλος: “Επιλέξτε εικόνες”, Επιλογή Πολλαπλών: Ναι, Ενέργεια: Άνοιγμα, Φίλτρο: picfilter. Προσθέτουμε τα ίδια κουμπιά και βάζουμε σαν handler τα picchooser_cancel_clicked και picchooser_ok_clicked αντίστοιχα. Θα δώσουμε και την ίδια μέθοδο, delete_chooser, στο delete_event του παραθύρου. Μπορούμε να χρησιμοποιήσουμε την ίδια μιας και το μόνο που κάνει είναι να επιστρέφει True. Τα τελικά toplevel αντικείμενα είναι αυτά:



Το γραφικό μας περιβάλλον είναι τώρα έτοιμο. Έχουμε την δυνατότητα να διαλέξουμε αρχείο προς εγγραφή, αρχεία προς ανάγνωση, ένα μέρος για να τα εμφανίσουμε όλα αυτά, δύο πεδία εισαγωγής των χρόνων, κουμπιά εφαρμογής και τέλους προγράμματος. Στο τρίτο μέρος θα παρουσιάσουμε το πρόγραμμα σε python το οποίο χειρίζεται αυτά τα αντικείμενα και δημιουργεί το αρχείο XML εναλλασσόμενων φόντων του Gnome.

Στα δύο προηγούμενα άρθρα είδαμε πως μπορούμε να φτιάξουμε το γραφικό περιβάλλον του προγράμματος μας χρησιμοποιώντας το Glade. Σε αυτό θα το παρουσιάσουμε και θα δούμε πως γίνεται η διασύνδεση μεταξύ τους. Η περιγραφή θα γίνει τμηματικά, ανά μέθοδο για να βοηθηθούν και όσοι δεν έχουν μεγάλη εμπειρία με την python.

Στην αρχή ενός αρχείου python καλό είναι να συμπεριλαμβάνουμε τα εξής:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

Η πρώτη γραμμή λέει στο shell του linux με τι πρόγραμμα θα προσπαθήσει να εκτελέσει αυτό το αρχείο, αν έχει δικαιώματα εκτέλεσης.
Η δεύτερη είναι μια οδηγία για τον python interpreter ότι το αρχείο είναι κωδικοποιημένο σε utf-8. Αυτό το χρειαζόμαστε αν έχουμε γράψει ελληνικά στο κώδικά μας (όπως σε κάποιο string), αλλιώς μας βγάζει μήνυμα λάθους.

Στην συνέχεια τα καθιερωμένα import. Εδώ θα χρειαστούμε τα:

```
import pygtk
pygtk.require('2.0')
import gtk
from datetime import datetime
```

Φορτώνουμε pygtk και gtk, οι βασικές μας βιβλιοθήκες και datetime για να έχουμε πρόσβαση στην ώρα και ημερομηνία του συστήματος.

```
class BackgroundXML(object):

    DIALOG_TYPE_ERROR = 0
    DIALOG_TYPE_INFO = 1

    CHOOSER_CANCEL = 0
    CHOOSER_OK = 1
```

Η δήλωση της κύριας κλάσης μας και μερικές σταθερές που θα χρησιμοποιήσουμε παρακάτω.

```
    def __init__(self):
        builder = gtk.Builder()
        builder.add_from_file("backgroundxml.glade")
        builder.connect_signals(self)
        self.backgroundxml = builder.get_object("backgroundxml")
        self.picchooser = builder.get_object("picchooser")
        self.picbuffer = builder.get_object("picbuffer")
        self.xmlchooser = builder.get_object("xmlchooser")
        self.xmlfile = builder.get_object("xmlfile")
        self.duration = builder.get_object("duration")
        self.change = builder.get_object("change")
        self.xmlfilter = builder.get_object("xmlfilter")
        self.xmlfilter.add_pattern('*.xml')
```

```
        self.picfilter = builder.get_object("picfilter")
        self.picfilter.add_pattern('*.jpg')
        self.picfilter.add_pattern('*.png')
        self.picfilter.add_pattern('*.JPG')
        self.picfilter.add_pattern('*.PNG')
```

Η μέθοδος __init__ η οποία φτιάχνει το κυρίως αντικείμενο μας από την κύρια κλάση μας. Ορίζουμε από ποιο αρχείο θα βρει ο builder τις οδηγίες κατασκευής των παραθύρων. Στην συνέχεια συνδέουμε τα σήματα με τις μεθόδους μας. Ορίζουμε μεταβλητές για το κυρίως παράθυρο και για τα παράθυρα επιλογής αρχείων, μεταβλητές για το buffer και τα πεδία εισόδου και στο τέλος ορίζουμε τα δύο φίλτρα αρχείων μας και ορίζουμε σε καθένα από αυτά τις τιμές που θέλουμε να φιλτράρουν.

```
    def close_window(self, widget, data=None):
        gtk.main_quit()

    def delete_chooser(self, widget, data=None):
        return True
```

Το GTK όταν καλεί μία μέθοδο ως αποτέλεσμα ενός σήματος του παραθύρου στέλνει και ένα ή δύο ορίσματα. Το πρώτο είναι το αντικείμενο (widget) από το οποίο προήλθε το σήμα και σε ορισμένες περιπτώσεις στέλνει και ορισμένα στοιχεία σχετικά με το αντικείμενο (data). Σε ένα drop-down για παράδειγμα όταν ο χρήστης επιλέξει μια τιμή από αυτό στέλνει το drop-down αντικείμενο και τον αριθμό της επιλογής ξεκινώντας από το 0 για την πρώτη κατά σειρά επιλογή του drop-down. Γι' αυτό και όλες οι μέθοδοί μας που έχουν να κάνουν με σήματα ορίζονται με αυτό τον τρόπο:

```
    def onoma_hanlder(self, widget, data=None):
```

Η close_window παραπάνω είναι υπεύθυνη για τον σωστό κλείσιμο του προγράμματος και η delete_chooser αποτρέπει την καταστροφή των παραθύρων επιλογής αρχείων.

```
    def xmlbutton_click(self,widget,data=None):
        response=self.xmlchooser.run()
        if response == self.CHOOSER_OK:
            self.xmlfile.set_text(self.xmlchooser.get_filename())
        self.xmlchooser.hide()
```

Η xmlbutton_click καλείται όταν ο χρήστης πατήσει το αντίστοιχο κουμπί επιλογής αρχείου XML. Το παράθυρο επιλογής εμφανίζεται καλώντας την μέθοδο run() του διαλόγου (το gtk.Filechooser αντικείμενο κληρονομεί από το gtk.Dialog). Ο έλεγχος πηγαίνει στο καινούργιο παράθυρο και επιστρέφει στο κυρίως όταν καταστραφεί ή καλέσει την μέθοδο response() η οποία επιστρέφει και τιμή. CHOOSER_OK είναι μία από τις δύο σταθερές μας που έχουμε ορίσει στην αρχή. Την επιστρέφει η responce όταν ο χρήστης πατήσει “Ένταξει”, αν διαλέξει “Άκυρο” επιστρέφει CHOOSER_CANCEL. Αν είναι OK βάζουμε στο αντίστοιχο πεδίο (xmlfile) αυτό που έχει διαλέξει. Αν έχει πατήσει το “Άκυρο” δεν κάνουμε τίποτα, και στις δύο περιπτώσεις κρύβουμε το παράθυρο. Μετά από αυτό ο έλεγχος έχει επιστρέψει στο κυρίως παράθυρο.

```
    def picbutton_click(self,widget,data=None):
        response = self.picchooser.run()
        if response == self.CHOOSER_OK:
            buffer1=self.picbuffer.get_text(self.picbuffer.get_start_iter(), self.picbuffer.get_end_iter())
            for fn in self.picchooser.get_filenames():
```

```
                buffer1=buffer1+fn+'\n'
            self.picbuffer.set_text(buffer1)
        self.picchooser.hide()
```

Η picbutton_click είναι ίδια με την προηγούμενη. Η μόνη της διαφορά είναι το ότι έχουμε να γεμίσουμε ένα buffer με πολλές γραμμές και όχι μόνο με μία τιμή. Γι' αυτό και βάζουμε στην μεταβλητή buffer1 ότι έχει η picbuffer μέχρι εκείνη την στιγμή και μετά προσθέτουμε κάθε αρχείο που έχει διαλέξει ο χρήστης με ένα “\n” στο τέλος. Όταν ολοκληρωθούν όλα τα αρχεία βάζουμε την buffer1 πίσω στην picbuffer προς εμφάνιση.

```
    def clearbutton_click(self,widget,data=None):
        self.picbuffer.set_text(u'')
```

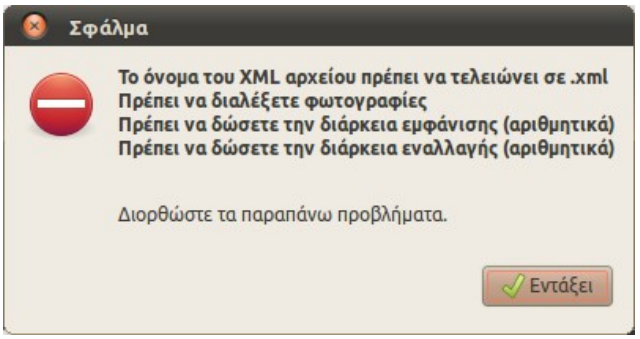
Η clearbutton_click καλείται όταν ο χρήστης πατήσει “Εκκαθάριση”. Διαγράφει ότι έχει η picbuffer βάζοντας ένα άδειο string.

```
    def picchooser_cancel_clicked(self, widget, data=None):
        self.picchooser.response(self.CHOOSER_CANCEL)

    def picchooser_ok_clicked(self, widget, data=None):
        self.picchooser.response(self.CHOOSER_OK)

    def xmlchooser_ok_clicked(self,widget,data=None):
        self.xmlchooser.response(self.CHOOSER_OK)

    def xmlchooser_cancel_clicked(self,widget,date=None):
        self.xmlchooser.response(self.CHOOSER_CANCEL)
```

Αυτές οι τέσσερις μέθοδοι είναι ταυτόσημες. Είναι υπεύθυνες για την επιστροφή της σωστής τιμής από τα δύο παράθυρα επιλογής αρχείων. Στο “Εντάξει” επιστρέφουν CHOOSER_OK και στο “Ακύρωση” CHOOSER_CANCEL. Χρησιμοποιούμε την response() μέθοδο για την επιστροφή η οποία κλείνει και το παράθυρο. Μπορούμε να τις αντικαταστήσουμε με μία η οποία να κάνει ακριβώς το ίδιο και για τα δύο παράθυρα;

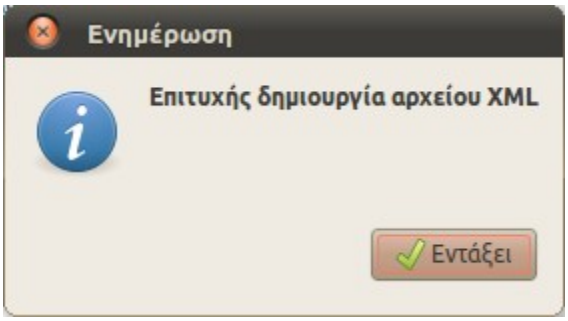
```
    def dialog(self, text, dialogtype):
        if dialogtype == self.DIALOG_TYPE_ERROR:
            dialog=gtk.MessageDialog(parent=None,flags=gtk.DIALOG_MODAL & \
                            gtk.DIALOG_DESTROY_WITH_PARENT, \
                    type=gtk.MESSAGE_ERROR, buttons=gtk.BUTTONS_OK)
            dialog.set_title(u"Σφάλμα")
            dialog.format_secondary_text(u"Διορθώστε τα παραπάνω προβλήματα.")
        elif dialogtype == self.DIALOG_TYPE_INFO:
            dialog=gtk.MessageDialog(parent=None,flags=gtk.DIALOG_MODAL & \
                            gtk.DIALOG_DESTROY_WITH_PARENT, \
                    type=gtk.MESSAGE_INFO, buttons=gtk.BUTTONS_OK)
            dialog.set_title(u"Ενημέρωση")
        dialog.set_markup(u"<b>"+text+"</b>")
        response=dialog.run()
        if response == gtk.RESPONSE_DELETE_EVENT or response == gtk.RESPONSE_OK:
            dialog.destroy()
```

Η dialog είναι μια δική μας βοηθητική μέθοδος η οποία δημιουργεί ένα παράθυρο διαλόγου. Παίρνει δύο ορίσματα: text και dialogtype. Το text είναι το κείμενο που θα εμφανίσει και dialogtype είναι το είδος του διαλόγου. Αν θέλουμε να εμφανίσουμε μήνυμα λάθους την καλούμε με DIALOG_TYPE_ERROR, αν θέλουμε να εμφανίσουμε κάποια άλλη πληροφορία την καλούμε με DIALOG_TYPE_INFO. Ο διάλογος δημιουργείται με την gtk.MessageDIalog η οποία παίρνει σαν ορίσματα τα parent, flags,type, buttons. Η parent χρησιμοποιείται για να δηλώσουμε κάποιο πατρικό παράθυρο, στην flags δηλώνουμε αν θέλουμε να είναι modal το παράθυρο μας και αν θα καταστραφεί μαζί με το πατρικό. (Modal είναι όταν το παράθυρο έχει αποκλειστικά τον έλεγχο και ο χρήστης δεν μπορεί να γυρίσει στο πατρικό αν πρώτα δεν κλείσει το modal παράθυρο).Η type παίρνει το είδος του διαλόγου και η buttons ποια κουμπιά θα εμφανιστούν με τον διάλογο. Όταν θέλουμε να εμφανίσουμε λάθος δημιουργούμε ένα gtk.MESSAGE_ERROR διάλογο με ένα μόνο κουμπί, το “Εντάξει”. Για πληροφορία το δημιουργούμε σαν gtl.MESSAGE_INFO με ένα μόνο κουμπί πάλι. Το μήνυμα το έχουμε συμπεριλάβει μέσα σε για να φαίνεται έντονο, και συμπληρωματικά αν είναι μήνυμα λάθους εμφανίζουμε και μια προτροπή, secondary_text. Εμφανίζουμε τον διάλογο με run() και περιμένουμε την απάντηση του χρήστη. Η απάντηση μπορεί να είναι είτε OK είτε DELETE (αν έχει πατήσει το X). Και στις δύο περιπτώσεις καταστρέφουμε το παράθυρο.




```
def clicked_ok(self, widget, data=None):
    picfiles = self.picbuffer.get_text(self.picbuffer.get_start_iter(), self.picbuffer.get_end_iter())
    filelist=picfiles.split("\n")
    filelist.remove('')
    errortxt=''
    now = datetime.now()
    if not self.xmlfile.get_text().endswith('.xml'):
        errortxt+=u'Το όνομα του XML αρχείου πρέπει να τελειώνει σε .xml\n'
    if not filelist :
        errortxt+=u'Πρέπει να διαλέξετε φωτογραφίες\n'
    if not self.duration.get_text().isdigit() :
        errortxt+=u'Πρέπει να δώσετε την διάρκεια εμφάνισης (αριθμητικά)\n'
    if not self.change.get_text().isdigit() :
        errortxt+=u'Πρέπει να δώσετε την διάρκεια εναλλαγής (αριθμητικά)\n'
    if errortxt:
        self.dialog(errortxt, self.DIALOG_TYPE_ERROR)
    else:
        with open(self.xmlfile.get_text(),'w') as f:
            try:
                f.write("<background>\n")
                f.write("  <starttime>\n")
                f.write("    <year>"+str(now.year)+"</year>\n")
                f.write("    <month>"+str(now.month)+"</month>\n")
                f.write("    <day>"+str(now.day)+"</day>\n")
                f.write("    <hour>"+str(now.hour)+"</hour>\n")
                f.write("    <minute>"+str(now.minute)+"</minute>\n")
                f.write("    <second>"+str(now.second)+"</second>\n")
```

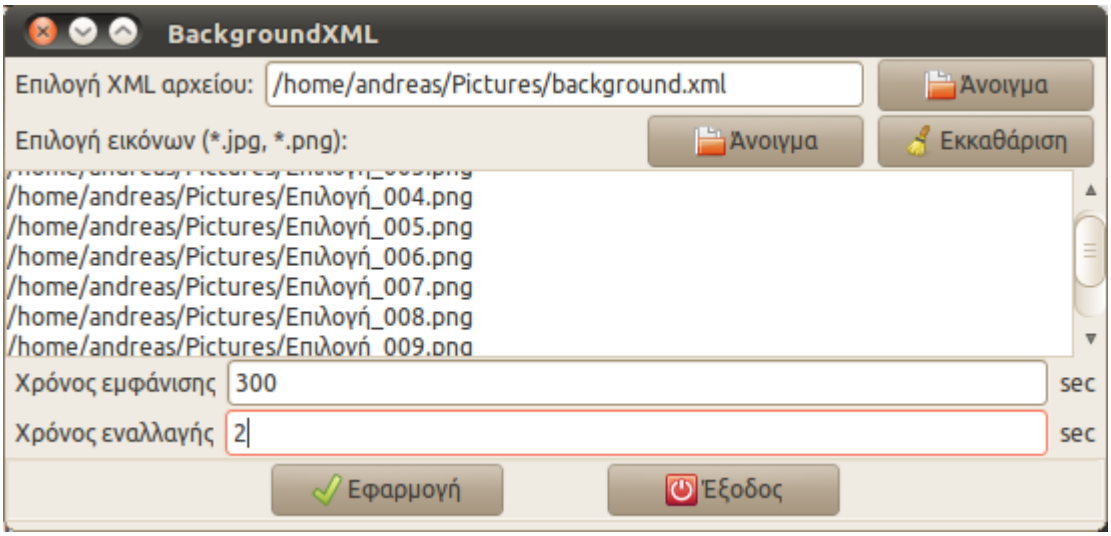
```
                f.write("  </starttime>\n")
                first=filelist[0]
                for ln in range(len(filelist)):
                    f.write("  <static>\n")
                    f.write("    <duration>"+self.duration.get_text()+"</duration>\n")
                    f.write("    <file>"+filelist[ln]+"</file>\n")
                    f.write("  </static>\n")
                    f.write("  <transition>\n")
                    f.write("    <duration>"+self.change.get_text()+"</duration>\n")
                    f.write("    <from>"+filelist[ln]+"</from>\n")
                    if ln == len(filelist)-1:
                        f.write("    <to>"+first+"</to>\n")
                    else:
                        f.write("    <to>"+filelist[ln+1]+"</to>\n")
                    f.write("  </transition>\n")
                    f.write("</background>\n")
                self.dialog(u"Επιτυχής δημιουργία αρχείου XML",self.DIALOG_TYPE_INFO)
            except:
                self.dialog(u"Πρόβλημα στην δημιουργία του αρχείου
XML",self.DIALOG_TYPE_ERROR)
```



Η clicked_ok είναι μέθοδος όπου δημιουργείται το αρχείο XML. Στην αρχή της φτιάχνουμε έναν πίνακα με τα περιεχόμενα του buffer και βάζουμε σε μια μεταβλητή την ώρα και ημερομηνία του συστήματος. Στην συνέχεια διενεργούμε ελέγχους για να επαληθεύσουμε τις εισόδους του χρήστη. Πρέπει να έχει δώσει αρχείο XML και αυτό να τελειώνει σε .xml, πρέπει να έχει διαλέξει εικόνες προς εμφάνιση και να έχει δώσει αριθμητικές τιμές στα πεδία με τους χρόνους. Για καθένα από αυτά, σε περίπτωση λάθους, προσθέτουμε και το αντίστοιχο λεκτικό σε μια μεταβλητή όπου και την εμφανίζουμε, αν έχει τιμή, με την dialog μέθοδό μας. Αν όλα είναι καλά ανοίγουμε το αρχείο XML προς εγγραφή και αρχίζουμε να γράφουμε σε αυτό τα δεδομένα όπως έχουμε περιγράψει εδώ. Με την επιτυχή δημιουργία του αρχείου εμφανίζουμε πληροφοριακό διάλογο ότι όλα τελείωσαν ομαλά, αλλιώς ένα μήνυμα λάθους.

```
    def main(self):
        self.backgroundxml.show()
        gtk.main()

if __name__ == "__main__":
    app = BackgroundXML()
    app.main()
```

Το πρόγραμμα τελειώνει με τον κλασικό τρόπο ενός pygtk προγράμματος, την main μέθοδο και τον τρόπο που την καλούμε στην έναρξη.



Σώζουμε το αρχείο σαν backgroundxml.py στον ίδιο φάκελο με το glade αρχείο και το τρέχουμε με python backgroundxml.py.

Εδώ ολοκληρώνεται μια πρώτη γνωριμία με το Glade, το GTK περιβάλλον και την python. Με λίγες γραμμές κώδικα σε python έχουμε έτοιμο το πρόγραμμα μας για την δημιουργία εναλλασσόμενων εικόνων στο φόντο του Gnome. Από εδώ και περά μπορεί ο καθένας να πειραματιστεί τόσο με το Glade και το γραφικό περιβάλλον όσο και με την python και την λογική του προγράμματος. Για παράδειγμα είναι χρήσιμο να προσθέσουμε έναν έλεγχο αν το αρχείο XML που έγραψε ο χρήστης υπάρχει ήδη ή όχι, να εμφανίζει ένα μήνυμα διαλόγου “Το αρχείο υπάρχει. Να το αντικαταστήσω;” και ανάλογα με την επιλογή του χρήστη να κάνει την αντικατάσταση ή όχι.

Σε επόμενο άρθρο θα ασχοληθούμε με την κατασκευή ενός προγράμματος σε gtk για την μαζική μετατροπή αρχείων text μεταξύ των διαφόρων τύπων κωδικοποίησης που υπάρχουν σήμερα, windows-1253, utf-8 κλπ