TALLINNA ÜLIKOOL
Informaatika Instituut

Mobiilirakenduste koostamine Pythoni abil

Bakalaureusetöö

Koostaja: Emil Azizov
õpinguraamatu nr. 060451
õpperühm LIF - 3

Juhendaja: lektor Jaagup Kippar

Autor: ..... „ .....„2010
Juhendaja: ..... „ .....„2010
Instituudi direktor: ..... „ .....„2010

Tallinn 2010

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Содержание

Koodid ja rakendused CD peal.

Вступление

Любой обладатель смартфона знает, что мобильная операционная система делает телефон похожим на старшего брата — настольный компьютер. Можно на свой вкус устанавливать разнообразные программы, которые кардинально влияют на удобство работы. Но мало кто знает, что на смартфоне, как и на компьютере, имеется возможность самостоятельно писать программы. Я расскажу о бесплатном языке программирования Python, который стараниями Nokia и Symbian адаптирован для смартфонов с интерфейсом S60. С помощью «змеиного» языка легко побеждать рутинные задачи: составлять отчеты, написать нужные вам конверторы, калькуляторы и многое другое. Так же в своей работе я покажу, как с помощью пару строчек можно написать первое приложение. По-этапно рассмотрим создание приложения для черчение графика. Представлю программу написанную мной, а так же попробую переделать это приложения для другого устройства, этим самым показав кросс-платформиность.

Что такое Питон

Python (питон) - интерпретируемый интерактивный объектно-ориентированный язык программирования высокого уровня. Python поддерживает классы, модули, обработку исключений, многонитевую обработку и др. Python относится к классу языков с динамической типизацией, предоставляет программисту автоматическую «сборку мусора» и удобные высокоуровневые структуры данных, такие как словари, списки, кортежи и др. Питон объединяет поразительную мощь с простым и ясным синтаксисом, продуманной модульностью и масштабируемостью. Python портируем и работает почти на всех известных платформах. Портом питона на мобильные телефоны S60 называют PyS60.

Несколько примеров приложений на питоне для S60

PyGSync

Рисунок 1 PyGSync

Программа на Питоне для синхронизации данных с календарем Google!

ImageDesigner

Рисунок 2 ImageDesigner

Многофункциональный фоторедактор. Умеет все, что только могут делать подобные программы - от рисования до преобразования изображений.

NiiMe

Рисунок 3 NiiMe

NiiMe это программное обеспечение на Python, который использует встроенный в смартфон акселерометр. Он может имитировать беспроводную мышь и колесико мыши, позволяет удаленно контролировать компьютер с телефона через Bluetooth.
Она также позволяет играть на барабанах просто встряхивая свой телефон.
http://www.niime.com/

SajiOS

Рисунок 4 SajiOS

SajiOS предназначен для людей, которым наскучил однотипный интерфейс.
Имитация windows на мобильном телефоне.
http://www.sajisoft.net.ms/

Это всего лишь несколько примеров того, что можно сделать с помощью питона. Если нужно быстро создать какой-нибудь, например, конвертер или любую другую программку, то быстрее и легче всего это сделать с помощью питона. Скрипты питона можно писать прямо на телефоне, используя всего лишь блокнот (Notepad) или же можно использовать редактор специальный для питона Ped[1], код которого также написан на питоне. Я сам пользовался Ped, когда создавал своё приложение. Найдя ошибку, можно легко исправит её с помощью редактора прямо в телефоне.

Для создания любого приложения нужна идея, в остальном поможет питон.

[1] Ped (Python Editor) - http://code.google.com/p/ped-s60

Установка необходимых приложений

Перед тем как начнём писать код, понадобится установить на компьютер и телефон следуящии программы:

- pyS60
- py2sis
- Symbian SDK
- Редактор для скриптов (я использую Notepad++)

Для начала установим pyS60на смартфон. Перед установкой необходимо знать версию операционной системы вашего смартфона, так как различные версии системы требуют различне версии Python. Для того чтобы узнать какой версии смартфон , нада зайти на http://www.forum.nokia.com/devices/matrix_s60_2ed_fp1_1.html и найти свой телефон в списке. У меня S60 2nd Edition, Feature Pack1. Заходим сюда http://sourceforge.net/project/showfiles.php?group_id=154155 и качаем нужную версию. Нада скачать 2 фаила: PythonForS60 и PythonScriptShell. Сначала устанавливаем в телефон PythonForS60 – это и есть сам интерпретатор Python, затем устанавливаем PythonScriptShell - среда для програмирования, для проверки скриптов. Это все что нужно инсталировать в смартфон.

Для того чтобы начать програмировать этого хватит.

Первое приложение на PyS60

Для примера напишем маленькую программу которая отправит введенный текст на введенный номер.

Открываем блокнот или любой другой редактор на ваше усмотрение.
Для начала нада импортиповать используемые библиотеки:

```
import appuifw # библиотека графического интерфейса
import messaging # библиоотека для отправки сообщений
```

Затем строка ввода текста *appuifw.query(label,type)*, где *label* – это метка в формате *unicode*, *type* – тип ввода информации: *'text',' number, 'date', 'time', 'code', 'query'* (для большой информации сморите доокуменацию pyS60, которую можна скачать там же где и сам pyS60):

```
sms_text=appuifw.query(u"Text to be send:","text")
sms_num=appuifw.query(u"Number:","number")
```

Проверка того, что пользватель дважды нажал ОК

```
if sms_num>0 & sms_text!=None:
```

Отправка SMS всего лишь одной функцией , думаю коментарии не нужны

```
    messaging.sms_send(sms_text,sms_num)
```

Диалоговое окно с сообщением «Message sent» *appuifw.note(text,type)*, где *text* – это текст в формате *unicode*, *type* – тип иконки: *'info','error','conf'*, по умолчанию *'info'* *(*для более подробной информации сморите доокуменацию pyS60)

```
    appuifw.note(u"Message sent","conf")
```

Сохраните этот файл в виде *.py .
И так код программы написан, теперь надо отослать код на телефон. Есть разница в версии телефона в зависимости от версии следуйте инструкциям.

Телефон 2nd ed :

1. Отправьте файл *.py на телефон по BLUETOOTH или с помощью PC suite (отощлется как сообщение)
2. Откройте полученое сообщение и инсталируйте его как «Python script»

Телефон 3rd ed:
Создайте папку 'Python' на диске e: (карта памяти) , откуда интерпретатор будет автоматически брать фаилы для запуска.

Перенесите скрипт *.py любым удобным вам способом в ту самую созданую папку 'Python' в карте памяти телефона.

Для того чтобя проверить скрипт на телефоне заходим в меню и находим там «Python», Затем Options→Run script выберите там свой скрипт и нажмите ОК (скрипт должен запустится).

Для телефонов 2nd ed не очень удобна каждый раз посылать файлы во входящие, а потом инсталировать занимает много времени для проверки программ в стадии создании. Я использую другой способ. Скрипты, которые мы инсталируем хранятся в "e:/system/apps/python/my". PC Suite Nokia Phone Browser не отображает папки "system", поетому, чтоб перенести туда скрипт нада создать на компьютере папку "system" внутри создать папку "apps" и так далее, пока не получится " system/apps/python/my" , туда помещаем скрипт и переносим папку system в телефон. И можна запускать скрипт.

О том как сделать программу с отдельной иконкой в меню написано пойже.

Как сделать программу GRAPH

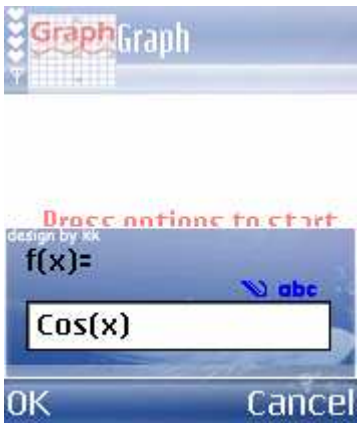
Описание программы:

Программа которая рисует график функции заданый пользователем ввиде f(x)=x. А также умеет посылать готовый график ввиде изобразения в формате .png. 2 способа отправки изображения :

1. по MMS
2. с помошью BLUETOOTH соединения.

Использование:

Рисунок 5
Нажать Options→Enter function



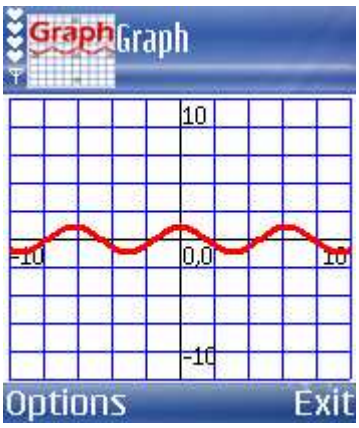


Рисунок 6 **Рисунок 7** **Рисунок 8**

Ввести функцию. Например cos(x)

Примечание: для функций используется синтаксис Python'а. Например для степени числа используется синтаксис x**y или pow(x,y).Пример: x^3 в синтаксисе Python'а будет выглядеть так x**3 или pow(x,3).

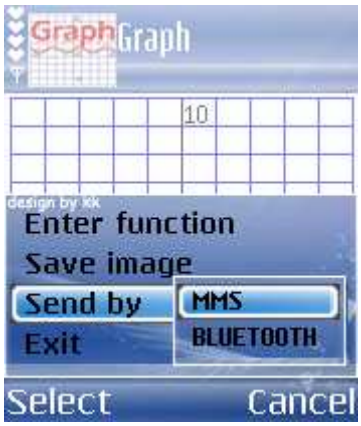


Рисунок 9

Options→Save image сохранит график ввиде рисунка в формате .png на карту памяти e:/graph.png .

Options→Send by→ MMS отправить по MMS. Для отправки требуется ввести номер получателя.

Options→Send by→BLUETOOTH отправить по BLUETOOTH соединению.

Произведется поиск устройств затем отошлется фаил на выбраное вами устройство.

Как сделать такое приложение

Jurgen Scheible в своей статье «Python for Series 60 tutorial» [2]преглагает 9 шагов для написания приложения:

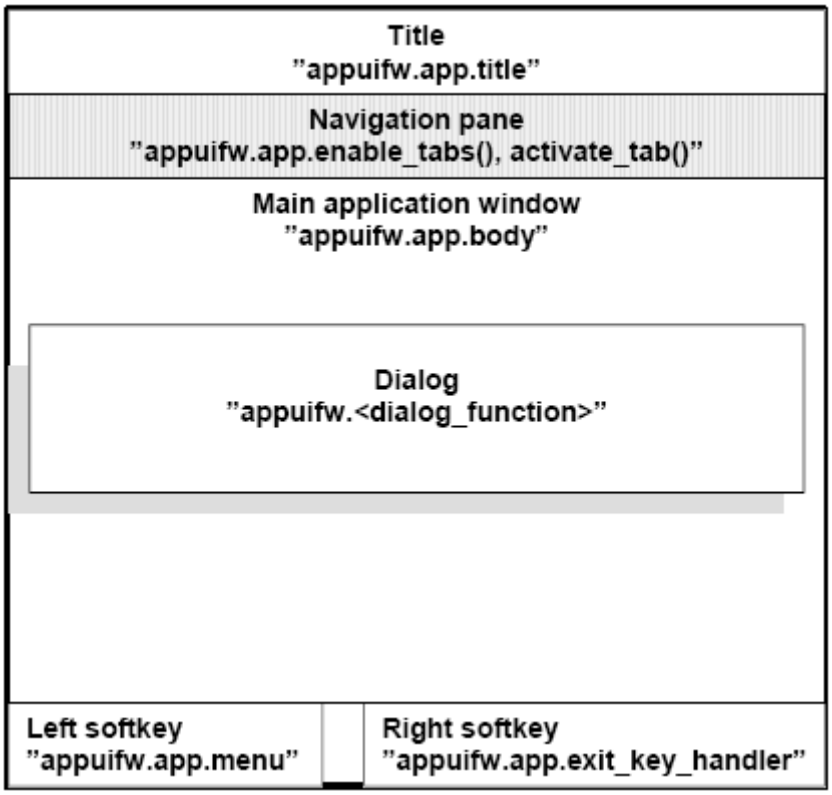


Рисунок 10 Тело приложения

1. импортировать необходимые модули (библиотеки)
2. установить размер окна (normal, large, full)
3. создать логику приложения
4. создать меню для приложения , если нада
5. установить процедуру выхода на кнопку exit
6. установить название приложения
7. позаботится об активных обьектах, если нужна
8. установить тело для приложения (text или canvas или listbox или None)
9. создать основной цикл если нужен

[2] Scheible, J. Python for Series 60 tutorial. 2010.
http://www.mobilenin.com/pys60/info_%20how_to_build_an_pys60_app.htm

Что за активные обьекты? И как ими пользоватся?

Импортируем e32 модуль

```
import e32
```

Создаем инстанцию активных обьектов

```
app_lock = e32.Ao_lock()
```

Запускает «замок» приложение будет ждать пока не вызовится lock.signal()

```
app_lock.wait()
```

Отключает замок

```
app_lock.signal()
```

Зачем он нужен?
Если например, вы создали приложение которое показывает список (listbox), и хотите чтобы пользователй выбрал что-то, и вы не использовали e32.Ao_lock(), то произойдет следуешее: список появится на экране и сразу же приложение закроется. Вот поэтому нужно использовать «замок».Для большой информации, смотрите документацию pyS60.

Вернемся к созданию приложения, так же Jurgen Scheible предлагает 2 скелета для написания программ, для программы Graph нам понадобится скелет без основного цикла.

```
# Copyright (c) 2006 Jurgen Scheible
# Application skeleton (no main loop)
```

1.импортируем необходимые модули

```
import appuifw
import e32
```

2.Устанавливаем размер экрана для приложения 'normal', 'large', 'full'

```
appuifw.app.screen='large'
```

3. Создаем логику приложения

```
# create your application logic ...
```

4. создаем меню для приложени

```
def item1():
    print "hello"

def subitem1():
    print "aha"

def subitem2():
    print "good"

appuifw.app.menu = [(u"item 1", item1),
                    (u"Submenu 1", ((u"sub item 1", subitem1),
                                    (u"sub item 2", subitem2)))]
```

5. установливаем процедуру выхода на кнопку exit

```
def exit_key_handler():
    app_lock.signal()
```

6. установливаем название приложения

```
appuifw.app.title = u"drawing"
```

7. Создаем инстанцию активных обьектов

```
app_lock = e32.Ao_lock()
```

8.установливаем тело для приложения (text или canvas или listbox или None)

```
appuifw.app.body = ...
```

установливаем процедуру выхода на кнопку exit

```
appuifw.app.exit_key_handler = exit_key_handler
```

Запускаем «замок» приложение будет ждать пока не вызовится lock.signal()

```
app_lock.wait()
```

Используя данный скелет начнём писать скрипт приложения.

Код приложения:

Импортируем все необходимые модули, модуль graphics для изобрзжения графика, mmsmodule для посылки MMScообшений(в 3rd ed. используется модуль messaging)

```
import appuifw, e32, graphics, mmsmodule, socket
from math import *
```

Для этого приложения установим размеры экрана приложения 'normal'

```
appuifw.app.screen='normal'
```

Пристуем к написанию логики программы

Глобальные переменные:

```
function = ""
filename='E:\\graph.png'
```

Устанавливаем шаг между точками на графике, чем меньше шаг тем чечше график, но занимает больше времени для прорисовки графика.

```
step =0.02
```

Переменные для разных цветов, для удобности использывания их в коде

```
BLUE=0x0000ff
RED=0xff0000
BLACK=0x000000
```

Функция getFunction вызовается из меню приложения, нужна для ввода пользавателем функции f(x)

```
def getFunction ():
```

Используем global для изменения глобальной переменной

```
    global function
```

Отображение запроса

```
    function = appuifw.query(u"f(x)=", "text")
```

Проверка правильности функции в случаи неисправности выводится ошибка и заново отображается запрос, вслучаи если пользователь нажмет 'cancel' переменная function примит значение None и запрос исчезнит. Неисправность проверяется запуском введенной функции eval(function.lower()) , переменная x вымышленое число. Но есть 'bug' , если пользователь введет например такую функцию 1/(x-0.00324945), что канешна маловероятно.

```
    #check if function  is right
    try:
        #check if user not pressed 'cancel'
        if function!=None:
            function=str(function) .lower()
            x=0.00324945
            eval(function)
            Plot()
    except:
        appuifw.note(u"Wrong function, try again","error")
        getFunction()
```

Функция Plot рисует сетку и график.

```
def Plot():
      #x from -10 to 10... set it to -10
      x=-10
      #clean screen
      img.clear()
```

Для того, чтобы нарисовать ровную сетку 10x10 понадабятся размеры кратные 10. Так, как я использую телефон 2 версии, то размеры экрана при 'normal' 176x144, поэтому будем использовать размеры 170x140 и каждая клетка будет иметь размер 17x14. Для того чтобы сетка отображалась по середине экрана добавляем поправку по оси x xIdent и поправку по оси y yIdent.

```
      #make the web in center of screen
      # w= 176, h=144
      w=170
      h=140
      xIdent=3
      yIdent=2
```

Переменные шагов по оси x и по оси y (через какое растояние рисовать следуюшую линию)/

```
      #Steps for drawing web
      Xstep=(w/10)
      Ystep=(h/10)
```

Координаты абциссы и ординаты

```
      Xzero=(w/2)
      Yzero=(h/2)
```

Рисуем сетку синим цветом(11 вертикальных и 11 горизонтальных линий).

```
      for i in range(11):
            img.line((xIdent+i*Xstep,yIdent,xIdent+i*Xstep,yIdent+h),
BLUE)
            img.line((xIdent,yIdent+i*Ystep,xIdent+w,yIdent+i*Ystep),
BLUE)
```

Рисуем абциссу и ординату чёрным цветом.

```
      img.line((xIdent,yIdent+Yzero,xIdent+w,yIdent+Yzero),BLACK)
      img.line((xIdent+Xzero,yIdent,xIdent+Xzero,yIdent+h),BLACK)
```

Рисуем метки координат так, как сетка 10x10, то делаем каждое деление равным 2, то получается максиму и минимумы сетки равны 10 и -10 соответственно.

```
      img.text((Xzero+5,Yzero+15),u"0,0")
```

```
img.text((4,Yzero+15),u"-10")
img.text((w-10,Yzero+15),u"10")
img.text((Xzero+5,15),u"10")
img.text((Xzero+5,h-5),u"-10")
```

Сетка нарисована переходим к черчению графика. Логика черчения простая: каждый раз увеличивая x на определённый шаг, подставляем в функцию введённую пользавателем и получаем y, затем рисуем точку на сетке.
Начиная от -10 до 10 с шагом step.

```
for i in range(20/step):
     x+=step
```

Подставляем x и получаем x. Могут появится ошибки, например при делении на 0, поэтому используем try , вслучаи обнаружении ошибки y присваивается 'NaN'. Почему именно 'NaN'? Проводя тесты обноружил что при извлечении из под корня отрицательного числа, ошибки не появляется, а y присваивается 'NaN'.

```
try: y= float(eval(function))
except: y='NaN'
if str(y)!='NaN':
```

У нас есть x и y, теперь нада перевести эти координаты на координаты экрана телефона. Нада использовать try, так как при проведении тестов обнаружил, что превышается размер числа float.
Рисуем точку толщеной в 3 пикселя.

```
try:
     fx = xIdent+Xzero+int(x*Xstep/2)
     fy = yIdent+Yzero-int(y*Ystep/2)
     img.point((fx,fy),RED,width=3)
except:pass
```

Отображаем полученный график на экране.

```
canvas.blit(img)
```

Программа Graph умеет посылать график в виде картинке по Bluetooth, что и делает следующая функция.

```
def BTsend():
```

Сохраняем сделаный ранее img (обьект класса graphics.Image) в фаил.

```
img.save(filename)
```

Ищем Bluetooth устройства, если Bluetooth был выключен ,то телефон спросит у вас включить его.

```
device=socket.bt_obex_discover()
```

Устройство выбрано, определяем адрес и канал. Передача фаила осушествляется с помощь протокола OBEX и профиля Object Push Profile (OPP, Базовый профиль для пересылки «объектов», таких как изображения, виртуальные визитные карточки и др. Передачу данных инициирует отправляющее устройство (клиент), а не приёмное (сервер)).

```
address=device[0]
channel=device[1][u'OBEX Object Push']
socket.bt_obex_send_file(address,channel,unicode(filename))
appuifw.note(u"Picture sent","info")
```

Функция для посылки изображения по MMS. Запрашивается номер телефона на который посылается MMS, если пользаватель нажмёт 'cancel', то цообщение не пошлётся.

def sendMMS():

```
nbr = appuifw.query(u"Sender to:", "number")
if nbr!=None:
     txt = u"" #mms text
     img.save(filename)
```

Данная строка и посылает сообщение, но есть разница в версиях телефона, на телефонах 3 версии нада использовать другой модуль для отправки сообщений, а именно модуль *messaging*.

```
     mmsmodule.mms_send(unicode(nbr),txt, unicode(filename))
     appuifw.note(u"Message sent", "conf")
```

Функция saveImage сохраняет экран приложения в фаил и оповешает пользователя об этом.

```
def saveImage():
    img.save(filename)
    appuifw.note(u'saved to '+unicode(filename))
```

Функция выхода, в данный момент выходом для програмы служит *app_lock.signal()*, который отключает «замок» *pp_lock.wait()*.

```
def exit():
    app_lock.signal()
```

Составляем меню вместе с подменю.

```
appuifw.app.menu = [(u"Enter function", getFunction), (u"Save
image", saveImage),\
                    (u"Send by", ((u"MMS", sendMMS),(u"BLUETOOTH",
BTsend))),\
                    (u"Exit",exit)]
```

Устанавливаем процедуру выхода на кнопку ' exit '.

```
appuifw.app.exit_key_handler = exit
```

Устанавливаем название приложения

```
appuifw.app.title = u"Graph"
```

Устанавливаем «замок» для приложения

```
app_lock = e32.Ao_lock()
```

Осталось создать тело приложения и им является *canvas*, на котором можно прорисовать изображение с помощью метода *canvas.blit(img)*.
Функцию *handle_redraw* вызовает *canvas* при случаи когда нада перерисовать *canvas* . Например, если пользователю кто-то позвонил, и затем он вернулся обратна в приложение, тогда *canvas* вызовит функцию *handle_redraw* для новоы прорисовки экрана приложения.

```
def handle_redraw(rect):
    canvas.blit(img)
canvas=appuifw.Canvas(handle_redraw)
```

Создаём переменные размера *canvas*

```
w,h = canvas.size
```

Создаём новый обьект *img* класса *graphics.Image*, с помощью которого рисуем график.

```
img=graphics.Image.new((w,h))
```

Рисуем текст, который пользователь увидит при старте приложения.

```
#set text at start of applet
img.text((20,h/2),u"Press options to start",RED,'title')
```

Устанавливаем тело приложения

```
appuifw.app.body=canvas
```

Устанавливаем «замок», который будет ждать сигнала *app_lock.signal()*, если вызовят функцию сигнала, то код закончится и приложение закроется.

```
app_lock.wait()
```

И так мы написали скрипт который работает через *PythonScriptShell*, теперь сделаем из неё приложение с собственной иконкой в меню.

Создание приложения с собственной иконкой в меню телефона.

До того как перейдём к установке программ, потребуется в скрипт приложения добавить одну строчку, для правильного закрытия приложения. В конец кода добовляем:

```
appuifw.app.set_exit()
```

Теперь нам понадобится установить следующие программы:

1. S60 Platform SDKs for Symbian, который можно скачать тут → http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html

   Symbian SDK, так же требует предустановленный Active Perl 5.6.1, который можно взять здесь→ http://www.activestate.com/activeperl/downloads/

2. Python plug-in for the SDK, плагин для поддержки питона в SDK, который можно скачать тут → http://www.mobilenin.com/pys60/resources/Python_for_2ndEd_SDK.exe

Последние версии SDK уже поддерживают питон, но они только для 3rd и 5th ed., если у вас 2nd ed., то требуется установить плагин.

Чтобы программа установилась на телефон со своей иконкой, нужен инсталятор, т.е .sis фаил. Самый легкий способ сделать со скрипта graph.py фаил graphInstaller.sis использовать инструмент py2sis, который устанавливается вместе с Python plug-in. py2sis – консольное приложение, которое запускается с командной строки. Что бы получить graphInstaller.sis в коммандной строке (START→Run→Cmd) нада ввести следуещее:

py2sis graph.py graphInstaller.sis --uid= 0x200240F0 --appname=Graph

Что есть что:
graph.py – сам скрипт, из которого делается приложение.
graphInstaller.sis – фаил, который получим в результате этой команды.

--appname=Graph – название приложение, которое будет отображатся при инсталяции, на телефоне.
--uid= 0x200240F0 – uid приложения, здесь я использовал свой собственный uid, узнать больше или зарегистрировать собственный uid можно здесь **www.symbiansigned.com** .

В результате, получаете фаил graphInstaller.sis, который можно установить на телефон, любым удобным способом, например через Nokia PC Suite. Минусом является, то что у программы нет иконки, и не возможно добавить фаилы в контейнер .sis. Для того чтобы сделать приложение вместе с иконкой, воспользуемся Symbian SDK.

Создаём иконки для приложения, нам понадобится 2 иконки, одна иконка отображается в «шапке» приложения, а вторая в меню телефона. Для каждой иконки нужно 2 .bmp фаила. Один 24-битный рисунок, а второй однобитная маска, которая служит для расспознования, прозрачных мест в рисунке. Иконка в загаловке приложения имеет следуюшие размеры: 44x44, а для меню телефона – 42x29.

44x44 для заголовка

icon1.bmp

icon1mask.bmp

42x29 для меню приложения

icon2.bmp

icon2mask.bmp

Теперь нужно обединить фаилы в контейнер .mbm, который будет использоватся для создания приложения. Для этого понадобится инструмент bmconv, который установился вместе с Symbian SDK. Как и py2sis, bmconv нада использовать в командной строке.

bmconv icon.mbm /c24icon1.bmp icon1mask.bmp /c24icon2.bmp icon2mask.bmp

Результатом является фаил *icon.mbm*, который понадобится пойже.

Нам понадобится .rss фаил, котрый использует *aiftool*, который создаст фаил .aif (информация для приложения, которая хранит в себе ссылки на иконки).
Открываем редактор и пишем следуюшее:

```
#include <aiftool.rh>

RESOURCE AIF_DATA
{
caption_list=
        {
        CAPTION { code=ELangEnglish; caption="Graph"; },
        CAPTION { code=ELangFrench; caption="Graph";}
        };
app_uid=0x200240F0;

num_icons=2;
}
```

Сохраняем фаил, как Graph.rss

Что есть что:
caption_list – список названий проложений на разных языках, из-за бага в *aiftool*, нада использовать минимум 2 языка.
app_uid - uid приложения, здесь я использовал свой собственный uid, узнать больше или зарегистрировать собственный uid можно здесь **www.symbiansigned.com** .
num_icons – количество иконок в приложении.

Создаём фаил .aif, используя инструмент aiftool и командную строку.

aiftool Graph icon.mbm

Примечание: Graph – это название фаила .rss
Получаем фаил *Graph.aif*.

Используя инструмент *py2sis* и аргумент *--leavtemp* получаем фаилы, которые используются для создания фаила .sis.

py2sis graph.py graphInstaller.sis --uid= 0x200240F0 --appname=Graph –leavetemp

В результате, получаете фаил graphInstaller.sis и папка temp , в которой хранятся фаилы сгенерированые с помощью *py2sis*. Делается это, для того чтобы можно было добавить туда фаил *Graph.aif* и сапоковать обратно в .sis. Так же понадобится переделать фаил .pkg, в нашем случаи это фаил *Graph.pkg*.

```
;
; Standalone Python for S60 app
;
;Languages
&EN
;
;
#{"Graph"},(0x200240F0),1,0,0
;
;Supports Series 60 v 2.0
;
(0x101F7960), 0, 0, 0, {"Series60ProductID"}
;
; Files to install:

"default.py"          -"!:\system\apps\Graph\default.py"
"Graph.app"           -"!:\system\apps\Graph\Graph.app"
"Graph.rsc"           -"!:\system\apps\Graph\Graph.rsc"
```

Добавляем в список фаилов также *Graph.aif*.

```
"Graph.aif"           -"!:\system\apps\Graph\Graph.aif"
```

Конечный фаил должен получится такой:

```
;
; Standalone Python for S60 app
;
;Languages
&EN
;
;
#{"Graph"},(0x0FFFFFFF),1,0,0
;
;Supports Series 60 v 2.0
;
(0x101F7960), 0, 0, 0, {"Series60ProductID"}
;
; Files to install:

"default.py"          -"!:\system\apps\Graph\default.py"
"Graph.app"           -"!:\system\apps\Graph\Graph.app"
"Graph.rsc"           -"!:\system\apps\Graph\Graph.rsc"
"Graph.aif"           -"!:\system\apps\Graph\Graph.aif"
```

Сохраните фаил, скопируйте *Graph.aif* в папку temp и в командной строке сначала перейдите в папку temp затем используя утилиту *makesis* создайте.sis фаил.

makesis Graph.pkg

В результате, получаете фаил Graph.sis, который можно установить на телефон.

Выше указаный метод создает .sis фаил, который можно использовать в телефоне, но в телефоне уже должен быть установлен питон, для использования этого приложения. Есть возможность поместить в своё приложение также интерпретатор питон, делается это добавлением следующей строки в фаиле .pkg :

```
@".\PythonForSeries60.SIS",(0x10201510)
```

И перенесите PythonForSeries60.SIS в папку temp и заново создайте .sis фаил. Но это повысит размер вашего .sis фаила на размер интерпретатора, а это не много не мало 580kb, без интерпретатора размер Graph.sis 9kb.

Приложение S60FlickrUp

За основную часть своей бакалаврской работы я взял написание приложения для мобильного телефона на базе S60v2. Идею написания данного приложения мне подал мой руководитель по этой работе Jaagup Kippar, его идея заключалась в том, что мало программ или вообще нет (в момент написания приложения) для отсылки фотографий или других изображений из мобильного телефона в социальную сеть. Основными сайтами для использования приложения были orkut.com и facebook.com. На orkut.com Я не нашёл способа для получения фотографий извне сайта. На facebook.com есть подробная API для поддержки просмотра сайта в разных приложениях, а также отсылка фотографий на сайт. Начав писать приложение, я столкнулся с проблемой. Авторизация пользователя в данном API осуществлялась с помощью браузера, что было не удобно или не возможно осуществить в телефоне. Поискав в Интернете другие аналогические сервисы, я наткнулся на flicr.com. Сайт для хранения и обмена изображениями. На данном сайте есть хороший API, изучив который, я начал писать приложения, и вот что получилось.

Эта программа написана для телефонов S60 2nd edition. А тестировалась на двух телефонах Nokia 6670 и Nokia 6630. Скриншоты были взяты именно с этих телефонов. На 6630 это приложение работало немного быстрей из-за более мощного процессора и большего объёма оперативной памяти.

Описание:

PyS60FlicrUp – это приложение для отправки снятых на телефоне фотографий на саит flickr.com в вашу учётную запись. Программа полностью написана на питоне.

Возможности:

- Снять и сразу отослать фотографию
- Отправлять фотографии уже имеющие в телефоне
- Задавать название и описание изображения
- Редактор изображений:
    - Повороты изображения
    - Отразить изображения (flip)
    - Изменение размера фотографии
- Запоминание Интернет настройки

Несколько скриншотов и объяснений:

Начальная страница

Рисунок 11 Начальная страница

То, что Пользователь видит при включении приложения

Стартовое меню

Рисунок 12 Стартовое меню

Стартовое меню: съемка фотографии, используя встроенную камеру; открыть рисунок имеющийся в памяти телефона; настройли и выходю

Настройки

Рисунок 13 Настройки

Пользователь может настроить: размер фотографии, использования точки доступа по умолчанию а так же пользователю нада 1 раз зарегистрироваться.

Регистрация

Рисунок 14 Регистрация

Для того чтобы зарегистрироватьсяб пользователю 1 раз надо зайти на указанную страничку и там ввести имя пользователя и паролью Это нужно для того чтобы программа знала на чью учётную запись посылать фотографии.

Просмотр файлов

Рисунок 15 Просмотр файлов

Возможность выбрать изображения для отправки

Режим съёмки

Рисунок 16 Режим съёмки

Снимать можно прямо из программы.

Просмотр снятой или открытой фотографии

Рисунок 17

Просмотр фотографии, а так же информация.

При необходимости изображение можно отредактировать или снять заново.

Рисунок 18

Редактор изображения

Рисунок 19 Редактор изображения

Простейший редактор изображения, который умеет Изменять размеры, поворачивать и отражать рисунок.

Название и описание фотографий

Рисунок 20
Название и описание фотографий

Перед отправкой пользователь может задать имя и описание файлу, но это не обязательно.

Тесты, Результаты и известные проблемы

Проблемы, связанные во время написания

Много времени заняло изучения flickr API[3], а так же проблемы с редактором некоторые из которых так и не удалось решить. Проблемы связаны с редактированием изображений с высоким разрешением. Во время редактирования этих фотографий телефону не хватает оперативной памяти. Проблему решить не удалось, программа выводит сообщение о не хватке оперативной памяти. Начальная версия приложения загружалась около 5 секунд, т.е. с момента нажатия на иконку приложения до появления меню в приложении. Путём неоднократного тестирования это удалось решить благодаря удалению не нужных библиотек и компиляции в байт код приложения. Время загрузки приложения уменьшилось благодаря компиляции используемых для приложения библиотек. Время загрузки сократилось с 5 секунд до 2 секунд.

[3] Flickr API , 2010 http://www.flickr.com/services/api/

Тестирование приложения на других версиях ОС symbian.

За проверку взят телефон N96 – смартфон S60 3rd edition с версией ОС symbian 9.3. Также этот телефон использует другую версию питона. Поэтому библиотеки а также не которые функции не совпадают. Телефон имеет больше оперативной памяти и дисплей с другим разрешением. Изменение кода постараюсь привести к минимуму, а так же изменение фонового рисунка для поддержки нового разрешения экрана.

Для начала в смартфон был установлен PyS60 версии 1.45. Путём изучения документации PyS60 и проверки библиотек используемых в приложении, пришёл к выводу, что ничего менять в коде не надо. Запуск скрипта в телефоне прошёл успешно, единственные изменения которые нужно будет сделать это:

- поменять размеры фонового рисунка
- переделать инсталятор (.sis файл) для поддержки S60 3rd edition.

Исправив картинки и создав нужный инсталлятор, следующее, что нужно было сделать – подписать готовый инсталлятор, так как в 3rd erdition используется система сертификатов для приложения. Так как на приложение никакого сертификата от symbian не получал, то можно использовать свой собственный сертификат для собственного телефона. Подробней о том, как получить сертификаты и как подписывать приложения вы можете найти тут → http://symbianv3.com/how-to-sign-sis-files-in-easy-3-steps/

Подписав и запустив программу, всё работало как нужно, единственной и серьёзной проблемой было то, что отснятые с помощью S60flickrUp вместо фотографии получался черный прямоугольник.

В поисках решения проблемы, я перечитал документацию, и не найдя проблему зашёл на сайт разработчиков питона[4], где нашёл что это известная проблема[5], решения этой проблемы пока нет. Это проблема касается только с N96, и проблема в питоне, который

[4] PyS60 maemo garage - https://garage.maemo.org/projects/pys60/
[5] Camera not working properly on N96
https://garage.maemo.org/tracker/index.php?func=detail&aid=3582&group_id=854&atid=3201

пока “не понимает” камеру N96. Плюсом использования N96 было то, что у этого телефона имеется WiFi ,с помощью чего, также удавалось посылать фотографии в Интернет.

Протестировав S60flickrUp на N96 и получив не желательные результаты было принято решение протестировать данное приложение на другом телефоне S60 3rd edition. Повторное тестирование было проведено с60 на Nokia E50, который, так же как и Nokia N96 имеет S60 3rd edition. Переделанная программа работала без проблем. Работала заметно быстрей, чем на телефонах s60 2nd edition, так как имеит более быстрый процессор и больший объём оперативной памяти.

Приложения, написанные PyS60, можно легко адаптировать под любую платформу мобильных телефонов S60. Но надо так же учитывать, то что у телефонов разные размеры оперативной памяти, разрешения экрана. Так же у более новых мобильных телефонов имеются другие возможности такие как, акселерометр, датчики света, а так же GPS приёмники. С помощью PyS60 можно использовать и такие не мало важные свойства телефона.

Заключение

В своей работе я показал, как быстро и легко можно написать маленькое приложение. На примере приложения Graph показал, как можно написать приложение, которое могут использовать и другие обладатели мобильных телефонов. А так же я предоставил ещё одно приложение S60flickrUP для показа возможностей питона на S60. Показал, что можно использовать данное приложение, при этом ничего не меняя в самом коду приложения, и на более новых мобильных телефонах, а именно на примере Nokia N96 и Nokia E50.

Питон самый лёгкий в обучении язык программирования, не зря у нас в университете именно с него начинают учить программированию. С созданием питона для телефонов S60, а это немного не мало 21% [6] мирового рынка мобильных телефонов, у пользователей появилась возможность писать приложения для своего мобильного телефона. Ведь до этого программирование для мобильных телефонов было на только на ограниченной JAVA и тяжелым в понимании, но мощным C++.

[6] Dickter A. April 27, 2010. AdMob Says Android Traffic Tops Apple's iPhone in U.S - http://www.newsfactor.com/story.xhtml?story_id=13100CVSEXM9

Ülevaade

Selles töös tutvustatakse Pythoni abil mobiilirakenduste lihtsat loomist. Pythoni porti S60 mobiiltelefoni jaoks nimetatakse PyS60. S60 telefonid on enamasti Nokia smartfonid nagu N-seeria ja E-seeria telefonid (Nokia N95, N70 jne). Selles töös näidatakse rakendusi, mis on tehtud Pythoni abil ja juhatatakse, kust saab neid alla laadida. Selgitatakse kuidas saab kiiresti väikest rakendust kirjutada. Isegi rakendust võib kirjutada otse mobiiltelefonis, kasutades Notepad'i või näiteks Ped'i (Python Editor), millest ka töös räägitakse. Selleks, et seda väikest rakendust kirjutada, on lihtsalt vaja telefoni peale paigutada PyS60. Õige versiooni valimiseks, on Teil vaja oma telefoni S60 versiooni teada, et kas on S60 2nd edition või S60 3rd edition.
Pikemat koodi kirjutame arvutis, kasutades Notepad'i või teist mugavat redaktorit ja salvestame seda .py failina.

Lihtsa koodi näide:

```
import appuifw
import messaging

sms_text=appuifw.query(u"Text to be send:","text")
sms_num=appuifw.query(u"Number:","number")

if sms_num>0 & sms_text!=None:
      messaging.sms_send(sms_text,sms_num)
      appuifw.note(u"Message sent","conf")
```

See rakendus oskab saada SMSi ette antud numbrile. Saatke see ".py" fail telefonile. S60 2nd edition telefonil avage see fail, siis Python soovitab seda koodi salvestada, aga S60 3rd Edition'i puhul lihtsalt pange oma koodi mälukaardile Python kausta, näiteks: ":\Python\fail.py" Avage Pythoni rakendus oma mobiiltelefonil ja leidke seal oma skripti, kust Te saate seda käivitada. Seda skripti/rakendust saab kasutada ainult Pythoni rakendusega abil. Kuidas teha rakendust oma ikooniga menüüs, selgitasin kirjutades oma rakendust GRAPH. See rakendus oskab graafikut joonistada kujul: f(x)=x ja saab need graafikuid pildina edastada MMSiga või Bluetooth'I abiga. Omal tööl kasutades Jurgen Scheible artiklit, näitasin, kuidas saab seda rakendust koostada. Ma kommenteerisin iga koodi rida, et seda oleks kergem arusaada. Pärast koodi kirjutamist on vaja Symbian SDK ja teised tööriistad allalaadida.

Kasutades neid tööristu saab rakendust oma ikooniga menüüs. Aga selleks, et seda saaks kasutada teine kasutaja, peab tal telefonis Python juba installeeritud olema. Teine probleem, millega ma kohtusin on sertifikaatide probleem S60 3rd Edition telefonidel. Selleks, et kasutaja saaks teie loodud rakendust kasutada, peate oma rakenduse allkirjastama. Kuidas oma rakendust allkirjastada võite leida veebilehel http://symbianv3.com/how-to-sign-sis-files-in-easy-3-steps/ Minu töö peaosaks oli Pythoni abil kirjutada rakendus, mille abil saaks tehtud pildi sotsiaalse tarkvara veebilehele panna. Sellise idée andis mulle minu töö juhendaja Jaagup Kippar. Oma sotsiaalse tarkvara teenusepakkujaks valisin Flickr'i. Ma tutvusin nende detailse API-ga ja hakkasin looma. Tuli välja S60FlickrUP rakendus, mis oskab fotod teha ja neid Flickrisse oma kasutaja albumisse panna. Minu rakenduses on ka olemas väike pildiredaktor järgmise funktsioonidega:

- Pööramine
- Peegeldumine (flip)
- Pildi suuruse muutmine

Selle rakenduse loomisel, läks palju aega Flickr API õppimisega ja rakenduste optimiseerimisega. Alguses lõpetatud S60FlickrUP oli väga aeglane, kuna laadimine toimus umbes 5 sekundit S60 2nd edition telefonis. Seda näidet sain lühendada 2 sekundiks, kustutades kasutamatuid mooduleid, koodi optimiseerimise ja kompileerimisega Pythoni 2.2.2 abil. Rakendus oli alguses kirjutatud ainult S60 2nd edition telefonile. Tegin uuringut ja proovisin seda rakendust Nokia N96 ja Nokia E50 telefonidel, mis on S60 3rd edition telefonid. Koodi muuta ei olnud vaja, sest moodulid olid samad, mis teises PyS60 versioonis. Selleks, et seda kasutada oli vaja oma logo suurust muuta ja teha uus installaatoreid 3rd edition jaoks.

Siinse bakalaureusetöö kaudu peaks iseõppijale valminud olema materjal, mille abil Pythonipõhiste mobiilirakenduste koostamist alustada. Ideid ja vihjeid leiab veidi keerulisemategi lahenduste jaoks.

Использованный материал

PyS60 download page. Retrieved Aprill 25, 2010, from
http://sourceforge.net/projects/pys60/

Scheible, J. Python for Series 60 tutorial. Retrieved April 21, 2010, from
http://www.mobilenin.com/pys60/menu.htm

Nokia. 2010. Python for S60. Retrieved April 21, 2010, from
http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Runtimes/Python_for_S60/

Nokia. Python for S60 wiki. Retrieved April 19, 2010, from
http://wiki.opensource.nokia.com/projects/PyS60

Nokia. 2010. Python Discussion Forum. Retrieved April 15, 2010, from
http://discussion.forum.nokia.com/forum/forumdisplay.php?s=&forumid=102

DZone, Inc. 2007.PyS60 snippets. Retrieved May 1, 2010, from
http://snippets.dzone.com/tag/pys60

Python Software Foundation. 2007-2010. Python. Retrieved April 21, 2010, from
http://www.python.org/

Flickr. 2010. Flickr API. Retrieved April 14, 2010, from
http://www.flickr.com/services/api/

Dickter A. April 27, 2010. AdMob Says Android Traffic Tops Apple's iPhone in U.S.
Retrieved April 28, 2010, from
http://www.newsfactor.com/story.xhtml?story_id=13100CVSEXM9

Lisa

Graphs.py

```
import appuifw, e32, graphics, mmsmodule, socket
from math import *
appuifw.app.screen='normal'
# create your application logic ...
function = ""
filename='E:\\graph.png'
step =0.02
BLUE=0x0000ff
RED=0xff0000
BLACK=0x000000

def getFunction():
      global function
      function = appuifw.query(u"f(x)=", "text")
      #check if function  is right
      try:
            #check if user not pressed 'cancel'
            if function!=None:
                  function=str(function).lower()
                  x=0.00324945
                  eval(function)
                  Plot()
      except:
            appuifw.note(u"Wrong function, try again","error")
            getFunction()
def Plot():
      #appuifw.note(u"plotting")
      #x from -10 to 10... set it to -10
      x=-10
      #clean screen
      img.clear()
      #make the web in center of screen
      # w= 144, h=176
      w=170
      h=140
      xIdent=3
      yIdent=2
      #Steps for drawing web
      Xstep=(w/10)
      Ystep=(h/10)
      #coordinates for abscissa and ordinate
      Xzero=(w/2)
      Yzero=(h/2)
      #drawing web
      #xIdent in every X coordinates and yIdent in every Y coordinates to
make web in center of screen
      for i in range(11):
            img.line((xIdent+i*Xstep,yIdent,xIdent+i*Xstep,yIdent+h),BLUE)
#vertical
            img.line((xIdent,yIdent+i*Ystep,xIdent+w,yIdent+i*Ystep),BLUE)
#horizontal
      #drawing abscissa and ordinate
      img.line((xIdent,yIdent+Yzero,xIdent+w,yIdent+Yzero),BLACK)
#horizontal
      img.line((xIdent+Xzero,yIdent,xIdent+Xzero,yIdent+h),BLACK)
#vertical
      #drawing labels :   0,0    -10    10
```

```
      img.text((Xzero+5,Yzero+15),u"0,0")
      img.text((4,Yzero+15),u"-10")
      img.text((w-10,Yzero+15),u"10")
      img.text((Xzero+5,15),u"10")
      img.text((Xzero+5,h-5),u"-10")
      #from -10 to 10 with known steps
      for i in range(20/step):
            x+=step
            #appuifw.note(u"try")
            try:
                  y = float(eval(function))
                  #appuifw.note(u"trying")
            except:
                  y='NaN'
                  #appuifw.note(u"except")
            #appuifw.note(u"if")
            if str(y)!='NaN':
                  #coordinates for dotes
                  #appuifw.note(u"try2")
                  try:
                        fx = xIdent+Xzero+int(x*Xstep/2)
                        fy = yIdent+Yzero-int(y*Ystep/2)
                        img.point((fx,fy),RED,width=3)
                  except:pass
                  #appuifw.note(u"cycle")
      canvas.blit(img)

def BTsend():
    img.save(filename)
    device=socket.bt_obex_discover()
    address=device[0]
    channel=device[1][u'OBEX Object Push']
    socket.bt_obex_send_file(address,channel,unicode(filename))
    appuifw.note(u"Picture sent","info")
def sendMMS():
      nbr = appuifw.query(u"Sender to:", "number")
      if nbr!=None:
            txt = u"" #mms text
            img.save(filename)
            appuifw.note(u"WTF")
            mmsmodule.mms_send(unicode(nbr),txt, unicode(filename))
            appuifw.note(u"Message sent", "conf")

#4 menu
def screenshot():
    img.save(filename)
    appuifw.note(u'saved to '+unicode(filename))
def exit():
    app_lock.signal()
      appuifw.app.exit()

appuifw.app.menu = [(u"Enter function", getFunction), (u"Save image",
screenshot),\
                    (u"Send by", ((u"MMS", sendMMS),(u"BLUETOOTH",
BTsend))),\
                    (u"Exit",exit)]
#5
appuifw.app.exit_key_handler = exit
#6
appuifw.app.title = u"Graph"
#7
app_lock = e32.Ao_lock()
#8
def handle_redraw(rect):
```

```
    canvas.blit(img)
canvas=appuifw.Canvas(handle_redraw)
w,h = canvas.size
img=graphics.Image.new((w,h))
#set text at start of applet
img.text((20,h/2),u"Press options to start",RED,'title')
appuifw.app.body=canvas
app_lock.wait()
```

S60flickrUP.py

```
import e32,camera, os, appuifw, key_codes,graphics,socket
api_key="134dc8ef8f94d18896c3cfecb633248a"
secret="bac14aec215957a6"
full_token="" #needed for authentication
logo =
graphics.Image.open(os.path.split(appuifw.app.full_name())[0]+"\\logo.png")
title,description,tags,type="","","","" #photo descriptions
img_filename='d:\\lastCaptured.jpg' #captured image path
#default setings
img_size_str="640x480"
img_size=(640,480)
apid=None #access point id
apname="None" #access point name
#temporary settings
temp_img_size_str,temp_img_size,temp_apid,temp_apname=None,None,None,None
old_sets_list=['','','']
#picture edit
pic_edited=None
img_path=""
img_name=""
#navigation functions
def notLoggedIn():
      #user will see it if he run it first time
      global screen
      screen=logo
      appuifw.app.title=u"S60flickrUp"
      appuifw.app.menu=[(u"Register",
registr),(u"About",about),(u'Exit',quit)]
      appuifw.app.exit_key_handler=quit
      appuifw.app.body = canvas
def menu():
      #main menu logged in user see it
      appuifw.app.exit_key_handler=quit
      camera.stop_finder()
      canvas.bind(key_codes.EKeySelect, take_picture)# to take care for bug
      canvas.bind(key_codes.EKeySelect, None)
      global screen
      screen=logo
      appuifw.app.body=canvas
      appuifw.app.title=u"S60flickrUp"
      appuifw.app.menu=[(u"Take a picture",cam),(u"Browse
photo",browser),(u"Settings",settings),(u"About",about),(u"Exit",quit)]
def registr(back="notLoggedIn"):#where to back
      global screen
      appuifw.app.menu=[(u"Enter code",getMini_token)]
      if back=="settings":
            appuifw.app.menu.append((u"Back",settings))
            appuifw.app.exit_key_handler=settings
      else:
            appuifw.app.menu.append((u"Back",notLoggedIn))
            appuifw.app.exit_key_handle= notLoggedIn
      appuifw.app.body=canvas
      #registr_image = graphics.Image.open("e:\\registr.png") #registration
picture
      registr_image =
graphics.Image.open(os.path.split(appuifw.app.full_name())[0]+"\\registr.pn
g")#registration picture
      screen=registr_image
      del registr_image
      canvas.blit(screen)
```

```
def setDescription():
      appuifw.app.title = u"Select picture options"
      appuifw.app.menu=[(u"Save and send",sendfoto),(u"Back",menu)]
      appuifw.app.exit_key_handler=menu
      #setting description list to listbox
      desc_listbox.set_list(desc)
      # set description listbox
      appuifw.app.body=desc_listbox
def imageViewer():
      appuifw.app.title = u"Please wait.."
      appuifw.app.menu=[(u"Send",setDescription),(u"Edit",editor),(u"New",(
(u"Take Photo",cam),(u"Browse photo",browser))),(u"Back",menu)]
      try:
            pic=graphics.Image.open(img_path)
      except:
            appuifw.note(u"Wrong image format, or file is too
large","error")
            browser()
      appuifw.app.exit_key_handler=menu
      appuifw.app.body=canvas
      pic_img=graphics.Image.new((w,h))
      # show image, check for picture sizes and put it to center of screen
      if pic.size[0]>pic.size[1]:
            pic_img.blit(pic, \
            target=(0, 0, w, float(pic.size[1])/float(pic.size[0]) * w),
scale = 1)
      else:
            indent=(w/2)-(((h-12)*float(pic.size[0])/float(pic.size[1]))/2)
            pic_img.blit(pic, \
            target=(indent,0, \
            (h-12)*float(pic.size[0])/float(pic.size[1])+indent, h-12 ),
scale = 1)
      appuifw.app.title=unicode(img_name)
      pic_img.rectangle((0,0.75 * w,w,h),fill=0xf0ffff)
      import time
      # show picture description: image size ,  size in KB and date
      text=unicode(str(pic.size[0])+"x"+str(pic.size[1])+" , "+\
            str(long(os.stat(img_path).st_size)/1024)+" KB , "+\

      time.strftime("%d.%m.%Y",time.localtime(os.stat(img_path).st_mtime)))
      pic_img.text((0,h-2),text)
      global screen,pic_edited
      pic_edited=pic #for edititing picture
      del pic
      screen=pic_img #set canvas screen to image viewer
      canvas.blit(screen)
def cam():
      appuifw.app.exit_key_handler = menu
      appuifw.app.title=u"Camera"
      appuifw.app.menu=[(u"Shoot",take_picture),(u"Back",menu)]
      appuifw.app.body=canvas
      global screen
      bg=graphics.Image.new((w,h))
      bg.clear()
      screen=bg
      canvas.blit(screen)
      camera.start_finder(finder_cb)
      canvas.bind(key_codes.EKeySelect, take_picture)
def editor():
      appuifw.app.title = u"Please wait.."
      global pic_edited
      pic_img=graphics.Image.new((w,h))
      pic_img.clear()
      if pic_edited.size[0]>pic_edited.size[1]:
```

```
            pic_img.blit(pic_edited, \
            target=(0, 12, w,
(float(pic_edited.size[1])/float(pic_edited.size[0]) * w)+12), scale = 1)
      else:
            indent=(w/2)-
((h*float(pic_edited.size[0])/float(pic_edited.size[1]))/2)
            pic_img.blit(pic_edited, \
            target=(indent,12, \
            h*float(pic_edited.size[0])/float(pic_edited.size[1])+indent, h
), scale = 1)
      text=unicode(str(pic_edited.size[0])+"x"+str(pic_edited.size[1])+"
Editing: "+img_name)
      pic_img.text((1,10),text)
      appuifw.app.title = u"Editor"
      appuifw.app.menu=[(u"Resize",resize), \
      (u"Rotate",((u"Left",rotate_left),(u"Right",rotate_right))), \
      (u"Flip",((u"Horizontal",flip_h),(u"Vertical",flip_v))), \
      (u"Save",editor_save),(u"Cancel",imageViewer)]
      appuifw.app.exit_key_handler=imageViewer
      global screen
      screen=pic_img
      canvas.blit(screen)

def settings():
      global
old_sets_list,temp_img_size_str,temp_img_size,temp_apid,temp_apname
      for x in range(len(sets_list)):
            old_sets_list[x]=sets_list[x]
      temp_img_size_str,temp_img_size,temp_apid,temp_apname=img_size_str,im
g_size,apid,apname
      appuifw.app.title = u"Settings"
      appuifw.app.menu=[(u"Edit",handler_settings),(u"Save and
exit",savesettings),(u"Back",cancel_settings)]
      appuifw.app.exit_key_handler=cancel_settings
      settings_listbox.set_list(sets_list)
      appuifw.app.body=settings_listbox
def browser():
      appuifw.app.title = u"Choose image to open"
      img_dirs = [(u'', 'C:\\Nokia\\Images'),(u'', 'E:\\Images')] #watch
folders
      def is_jpg(x): #function for filter .   check fo jpg files
            ext=os.path.splitext(x)[1].lower()
            return ext == '.jpg'
      img_list = []
      for nickname,path in img_dirs:
            if os.path.exists(path):
                  img_list += map(lambda x:
(nickname+x,path+'\\'+x),map(lambda x: unicode(x,'utf-8'),\
                  filter(is_jpg, os.listdir(path))))
      index = appuifw.selection_list(map(lambda x: unicode(x[0]),
img_list))
      if index >= 0:
            global img_path,img_name
            img_path=img_list[index][1].encode('utf-8')
            img_name=img_list[index][0].encode('utf-8')
            img_name=img_name[:-4] #without extension
            imageViewer()
      else:appuifw.app.title=u"S60flickrUp"
def help():
      #instructions.. coming soon
      appuifw.note(u"HELP!")
def about():
      appuifw.note(u"S60flickrUp by Emil Azizov©")
#write read settings
```

```
def write_settings():
    # write your settings:
    # define     CONFIG_DI the directory where you want to store your
settings
      #CONFIG_DIR='c:/flickrconf'
      CONFIG_DIR=os.path.split(appuifw.app.full_name())[0]
      CONFIG_FILE=os.path.join(CONFIG_DIR,'conf.txt')
    # make sure the settings file exists (created here when running the
script for the first time).
      if not os.path.isdir(CONFIG_DIR):
            os.makedirs(CONFIG_DIR)
            CONFIG_FILE=os.path.join(CONFIG_DIR,'conf.txt')
      config={} # create an empty dictionary (to store your values attached
to variables)
      config['full_token']= full_token
      config['img_size']=img_size
      config['apname']=apname
      config['apid']=apid
      config['img_size_str']=img_size_str
      f=open(CONFIG_FILE,'wt') # open the your "settings" file in "write
mode" where to store the dictionary
      f.write(repr(config)) #write settings to file
      f.close()
def read_settings():
      # read your settings:
      global full_token,apid,apname,img_size,img_size_str
    # define the settings file where to read from
      CONFIG_DIR=os.path.split(appuifw.app.full_name())[0]
      CONFIG_FILE=os.path.join(CONFIG_DIR,'conf.txt')
      try:
        # open the settings file in "read mode"
            f=open(CONFIG_FILE,'rt')
            try:
            # read the file
                  content = f.read()
                  config=eval(content)
                  f.close()
            # get the value that is attached to the variable
                  full_token=config.get('full_token','')
                  img_size=config.get('img_size','')
                  apname=config.get('apname','')
                  apid=config.get('apid','')
                  img_size_str=config.get('img_size_str','')
                  if full_token!=None:return True #if full token exists
then authentication exists
            except:
                  print 'can not read file'
                  return False
      except:
            print 'can not open file'
            return False
#using connection
def registration(mini_token):
      import md5,httplib
      global full_token
      #signature needed to be hashed
      signature = secret + "api_key" + api_key +
"methodflickr.auth.getFullTokenmini_token" + mini_token
      api_sig=md5.new(signature).hexdigest() # hash signature
      #making connection with flickr
      conn = httplib.HTTPConnection("www.flickr.com")
      conn.request("GET",
"/services/rest/?method=flickr.auth.getFullToken&api_key=" + api_key +
"&mini_token=" + mini_token + "&api_sig=" + api_sig)
```

```
	#get response
	resp = conn.getresponse()
	respXML =resp.read()
	print respXML
	# parse response for answer
	token_resp=miniparser(respXML,"token")
	conn.close()
	# check if login success
	if token_resp == "" :
		appuifw.note(u"LOGIN FAILED","error")
	else: # if success write settings
		appuifw.note(u"LOGIN SUCCESS","conf")
		full_token=token_resp
		write_settings()
		menu() # goto to menu

def sendfoto():
	import md5
	#sending photo
	auth_token=full_token
	#signature needed to be hashed
	signature = secret + "api_key" + api_key + "auth_token" + auth_token
	#open file (photo) to be send
	try:
		f=open(img_filename,'rb')
	except:
		appuifw.note(u"READING FAILED")
	rawfile = f.read()
	f.close()
	#fields for post_multipart
	fields=[['api_key',api_key],['auth_token',auth_token]]
	# check for photo descriptions in alphabetic sort
	if description!="":
		signature+='description'+description
		fields.append(['description',description])
	if type!="":
		signature+=type+'1'
		fields.append([type,'1'])
	if tags!="":
		signature+='tags'+tags
		fields.append(['tags',tags])
	if title!="":
		signature+='title'+title
		fields.append(['title',title])
	#hash signature
	api_sig=md5.new(signature).hexdigest()
	#append fields for post_multipart
	fields.append(['api_sig',api_sig])
	#send and get response from flickr
	resp_xml=post_multipart("api.flickr.com",'/services/upload/',fields,[
img_filename,rawfile])
	print "resp_xml="+resp_xml
	#parse response to get photoid
	resp=miniparser(resp_xml,'photoid')
	print "resp="+resp
	#if photoid != " " then success
	if resp=="":
		appuifw.note(u"SEND FAILED","error")
		#get error code from response
		code=atrparser(resp_xml,'code')
		#print code
		#note error
		if code=="3":
			appuifw.note(u"error: General upload failure","error")
```

```
            if code=="98":
                  appuifw.note(u"error: Login failed","error")
            if code=="105":
                  appuifw.note(u"error: Service currently
unavailable","error")
      else:
            appuifw.note(u"SEND SUCCESS","conf")
def check_login():
      import md5,httplib
      #check for authentication status
      signature = secret + "api_key" + api_key + "auth_token"+
full_token+"methodflickr.auth.checkToken"
      api_sig=md5.new(signature).hexdigest()
      conn = httplib.HTTPConnection("www.flickr.com")
      conn.request("GET",
"/services/rest/?method=flickr.auth.checkToken&api_key=" + api_key +
"&auth_token=" + full_token + "&api_sig=" + api_sig)
      resp = conn.getresponse()
      respXML =resp.read()
      username=atrparser(respXML,"username")
      if username=="":
            registr()
      else:
            appuifw.note(unicode("Hey "+username))
            menu()

#handlers(callback functions)
def handler_desc():
      #handler for description listbox
      global title,description,tags,type
      #change title
      if desc_listbox.current()==0:
            title=appuifw.query(u"Title:","text")
            if title==None: title=" Click to add title "

      desc[desc_listbox.current()]=(desc[desc_listbox.current()][0],unicode
(title))
            if title==" Click to add title ": title=""
            else: title=str(title)
      #change description
      if desc_listbox.current()==1:
            description=appuifw.query(u"Description:","text")
            if description==None: description=" Click to add description "

      desc[desc_listbox.current()]=(desc[desc_listbox.current()][0],unicode
(description))
            if description==" Click to add description ":description=""
            else:description=str(description)
      #change tags
      if desc_listbox.current()==2:
            tags=appuifw.query(u"Tags:","text")
            if tags==None: tags=" Click to add tags "

      desc[desc_listbox.current()]=(desc[desc_listbox.current()][0],unicode
(tags))
            if tags==" Click to add tags ": tags=""
            else:tags=str(tags)
      #change photo type: public,friends,family
      if desc_listbox.current()==3:
            type_list = [u"Public", u"Friends", u"Family"]
            type_names=["is_public","is_friends","is_family"]
            type_selected= appuifw.popup_menu(type_list, u"Select type:")
            if type_selected==None : type_selected=0
            type=str(type_names[type_selected])
```

```
      desc[desc_listbox.current()]=(desc[desc_listbox.current()][0],unicode
(type_list[type_selected]))
      #set shanged list to listbox and focus on current row
      desc_listbox.set_list(desc,desc_listbox.current())
def handle_screen(rect):
      # handler for canvas
      # is called whenever a part of the Canvas has been obscured by
something
    canvas.blit(screen)
def handler_settings():
      # handler for settings listbox
      global
sets_list,temp_img_size_str,temp_img_size,temp_apid,temp_apname
      # change captured photo size
      if settings_listbox.current()==0:
            # get available image sizes
            img_sizes=camera.image_sizes()
            #change size format from (w,h) to wxh
            for x in range(len(img_sizes)):

      img_sizes[x]=unicode(str(img_sizes[x][0])+"x"+str(img_sizes[x][1]))
            img_selected= appuifw.popup_menu(img_sizes, u"Select size:")
            if img_selected==None:img_selected=1
            #img size in format wxh
            img_size_str=str(img_sizes[img_selected])
            #img size tuple (w,h)

      img_size=(int(img_size_str.split('x')[0]),int(img_size_str.split('x')
[1]))

      sets_list[settings_listbox.current()]=(sets_list[settings_listbox.cur
rent()][0],unicode(img_size_str))
            settings_listbox.set_list(sets_list,settings_listbox.current())
            #temporaly saving settings
            temp_img_size_str=img_size_str
            temp_img_size=img_size
      if settings_listbox.current()==1:
            apid = socket.select_access_point()
            if apid!=None:
                  for x in range(len(socket.access_points())):
                        if socket.access_points()[x].get('iapid','')==apid:

      apname=socket.access_points()[x].get('name','')
            else:
                  apname="None"

      sets_list[settings_listbox.current()]=(sets_list[settings_listbox.cur
rent()][0],unicode(apname))
            settings_listbox.set_list(sets_list,settings_listbox.current())
            #temporaly saving settings
            temp_apid=apid
            temp_apname=apname
      if settings_listbox.current()==2:registr("settings")

def finder_cb(im):
      #handler for camera
    canvas.blit(im)
#functions for editor
def flip_h():
      #flip image horisontally
      appuifw.app.title = u"Please wait.."
      global pic_edited
      try:pic_edited=pic_edited.transpose(graphics.FLIP_LEFT_RIGHT)
```

```
      except:appuifw.note(u"Flip error","error")
      editor()
def flip_v():
      #flip image vertically
      appuifw.app.title = u"Please wait.."
      global pic_edited
      try:pic_edited=pic_edited.transpose(graphics.FLIP_TOP_BOTTOM)
      except:appuifw.note(u"Flip error","error")
      editor()
def rotate_left():
      #rotate image to left
      appuifw.app.title = u"Please wait.."
      global pic_edited
      try:pic_edited=pic_edited.transpose(graphics.ROTATE_90)
      except:appuifw.note(u"Rotate error","error")
      editor()
def rotate_right():
      #rotate image to right
      appuifw.app.title = u"Please wait.."
      global pic_edited
      try:pic_edited=pic_edited.transpose(graphics.ROTATE_270)
      except:appuifw.note(u"Rotate error","error")
      editor()
def editor_save():
      #save edited image
      appuifw.app.title = u"Saving.."
      global img_path,img_name
      pic_edited.save(img_filename)
      #set image path and image name for image viewer
      img_path=img_filename
      img_name="Edited_"+img_name
      imageViewer()
def resize():
      # resize picture to known sizes: 640x480, 320x240 and 160x120
      global pic_edited
      def custom():
            global pic_edited
            appuifw.app.title=u"Please enter new size"
            newx,newy = appuifw.multi_query(u"Enter width", u"Enter hight")
#Asks for the new size
            valid_sizes=False
            try:
                  if int(newx)> 1280 or int(newy) > 1280:
                        appuifw.note(u"Most be integer, not bigger than
1280","info")
                        custom()
                  else:
                        valid_sizes=True
            except:
                  appuifw.note(u"Most be integer, not bigger than
1280","info")
                  custom()
            if valid_sizes:
                  try:
                        appuifw.app.title = u"Resizing.."
                        pic_edited=pic_edited.resize((int(newx),int(newy)))
                        editor()
                  except:
                        appuifw.note(u"Resize error, not enough
memory","error")
                        editor()

      img_sizes=camera.image_sizes()[1:]
      for x in range(len(img_sizes)):
```

```
      img_sizes[x]=unicode(str(img_sizes[x][0])+"x"+str(img_sizes[x][1]))
      img_sizes.append(u"Custom")
      img_selected= appuifw.popup_menu(img_sizes, u"Select size:")
      if img_selected==(len(img_sizes)-1):
            custom()
      else:
            if img_selected>=0:
                  try:
                        appuifw.app.title = u"Resizing.."

      pic_edited=pic_edited.resize(camera.image_sizes()[img_selected+1])
                  except:
                        appuifw.note(u"Resize error, not enough
memory","error")
                  editor()

def cancel_settings():
      # cancel settings and set old settings list to listbox
      global sets_list
      for x in range(len(sets_list)):
            sets_list[x]=old_sets_list[x]
      menu()
def savesettings():
      #save settings and write them to file
      global apid,apname,img_size_str,img_size
      img_size_str,img_size,apid,apname=temp_img_size_str,temp_img_size,tem
p_apid,temp_apname
      write_settings()
      menu()
def getMini_token():
      #input for minitoken
      code=str(appuifw.query(u"Code:","number"))
      if code!="None":
            if len(code)==9:
                  mini_token=code[:3]+"-"+code[3:6]+"-"+code[6:] #to
correct mini_token
                  registration(mini_token)
            else:
                  appuifw.note(u"Wrong code. Try again", "error")
                  getMini_token()
      else:
            registr()

def take_picture():
      # take picture from camera
      canvas.bind(key_codes.EKeySelect, None) # deactivate camera button
      camera.stop_finder() #stop camera preview
      try:
            pic = camera.take_photo(size = img_size) # take picture
            pic.save(img_filename)
            global img_path,img_name
            img_path=img_filename
            img_name="from_Camera"
            imageViewer()
      except:appuifw.note(u"Capturing error","error")

#quit function
def quit():
      camera.stop_finder()
      app_lock.signal() #quit to python shell
# parsers and multipart generator
def atrparser(xstr,atr):
      #parsing atributes
```

```
    atrname=atr+'="'
    if(xstr.find(atrname)>-1):
        for x in range(len(xstr)):
            if xstr[x+xstr.find(atrname)+len(atrname)]=='"':
                return xstr[(xstr.find(atrname) +
len(atrname)):x+xstr.find(atrname)+len(atrname)]
    else:
        return ""
def miniparser(xstr,param):
      #parsing for parametrs
      startTag = "<" + param + ">"
      endTag = "</" + param + ">"
      if(xstr.find(startTag)>-1) & (xstr.find(endTag)>-1):
            if(xstr.find(startTag) + len(startTag) < xstr.find(endTag)):
                  return xstr[(xstr.find(startTag) +
len(startTag)):xstr.find(endTag)]
            else:
                  return ""
      else:
            return ""
def post_multipart(host, selector, fields, file):
      import httplib
      #Post fields and files to an http host as multipart/form-data.
      #fields is a sequence of (name, value) elements for regular form
fields.
      #files is a sequence of (filename, value) elements for data to be
uploaded as files.
      #Return response
      content_type, body = encode_multipart_formdata(fields, file)
      h = httplib.HTTPConnection(host)
      h.putrequest('POST', selector)
      h.putheader('content-type', content_type)
      h.putheader('content-length', str(len(body)))
      h.endheaders()
      h.send(body)
      response = h.getresponse()
      answer = response.read()
      return answer
def encode_multipart_formdata(fields, file):
      #fields is a sequence of (name, value) elements for regular form
fields.
      #files is a sequence of (filename, value) elements for data to be
uploaded as files.
      BOUNDARY = '----------ThIs_Is_tHe_bouNdaRY_---$---'
      CRLF = '\r\n'
      L = []
      for (key, value) in fields:
            L.append('--' + BOUNDARY)
            L.append('Content-Disposition: form-data; name="%s"' % key)
            L.append('')
            L.append(value)
      L.append('--' + BOUNDARY)
      L.append('Content-Disposition: form-data; name="photo";
filename="%s"' % (file[0]))
      L.append("Content-Type: image/jpeg")
      L.append('')
      L.append(file[1])
      L.append('--' + BOUNDARY + '--')
      L.append('')
      body = CRLF.join(L)
      content_type = 'multipart/form-data; boundary=%s' % BOUNDARY
      #print body
      return content_type, body
#canvas image
```

```
screen=logo
#creating canvas
canvas = appuifw.Canvas(redraw_callback=handle_screen)
w,h = canvas.size
canvas.blit(screen)
#listbox for settings and descriptions
read_settings() #get settings for listbox
sets_list = [(u'Captured image size:',unicode(img_size_str)),\
      (u'Default access point:',unicode(apname)),(u'Registration',u'If
having problems with authentication')]
desc = [(u'Title:',u' Click to add title '),(u'Description:',u' Click to
add description '),\
      (u'Tags:',u' Click to add tags '),(u'Type',u'Public')]
settings_listbox=appuifw.Listbox([(u"",u"")],handler_settings)
desc_listbox=appuifw.Listbox([(u"",u"")],handler_desc)
#app properties
appuifw.app.body = canvas
appuifw.app.title = u"S60flickrUp"
appuifw.app.exit_key_handler = quit
#reading settings
if read_settings():menu()
else:notLoggedIn()
#making default connection
if apid!=None:apo = socket.access_point(apid)
else: apo=None
socket.set_default_access_point(apo)
#app lock
app_lock = e32.Ao_lock()
app_lock.wait()
#appuifw.note(unicode(end_time-start_time))
```