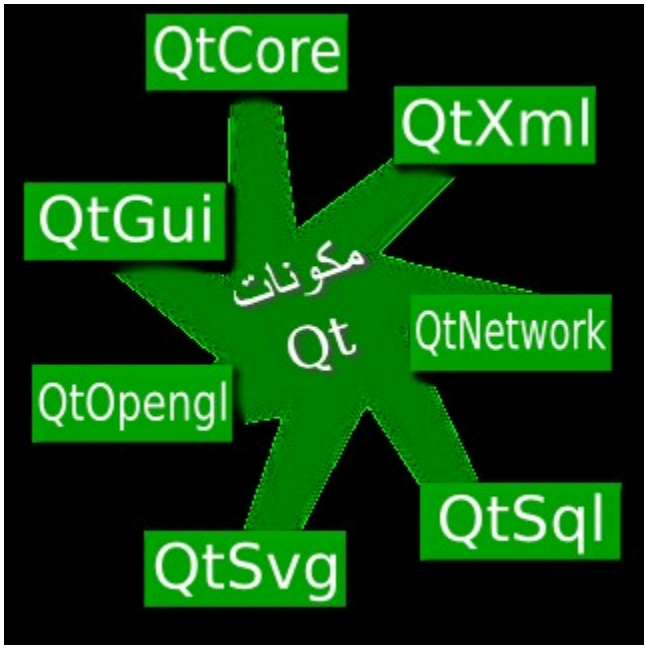
Ruby/Qt tutorial

كدا انهينا RubyGnome و wxRuby نستكمل رحلتنا مع Ruby/Qt

Qt هى اداة تطوير تطبيقات toolkit كانت مملوكة لشركة trolltech حتى استحوذت عليها Nokia
من مميزاتها انها عابرة للنظم Cross-Platform
تنقسم Qt للعديد من الوحدات

--اعجوبة pyqt4



الوحدة QtCore تحتوي الوظائف المركزية غير الرسومية. تستخدم هذه الوحدة في التعامل مع الوقت والملفات والأدلة (المجلدات) وأنواع البيانات المختلفة والسيالات streams وروابط URL وأنواع الملفات MIME والعمليات processes and threads.

الوحدة QtGui فتحتوي على المكونات الرسومية وما يتعلق يها من صنوف. وتشتمل على سبيل المثال على الأزرار والنوافذ وأشرطة الأدوات والحالة والزلاقات والصور النقصية والألوان والخطوط ...إلخ.

الوحدة QtSVG بها الصنوف اللازمة لعرض محتويات ملفات SVG. وهي لغة وصفية للرسومات المتجهية ثنائية الأبعاد والتطبيقات الرسومية ب XML

الوحدة QtOpenGL تستخدم لتوليد رسومات ثلاثة الأبعاد أو ثنائية باستخدام مكتبة OpenGL. مما يمكننا من دمج Qt مع OpenGL دون عناء.

الوحدة QtNetwork تحتوي على الصنوف اللازمة لبرمجة الشبكات. هذه الصنوف تسمح بكتابة برامج خوادم severs أو مخدومات clients لبروتوكولات TCP/IP و UDP. مما يجعل برمجية الشبكات أسهل وقابل للنقل لمنصات أخرى.

الصنوف في الوحدة QtXml للعمل على ملفات xml. هذه الوحدة تقدم تطبيق ل SAX و DOM.

الوحدة QtSql تقدم صنوف التعامل مع قواعد البيانات.

--اخر الإقتباس

البرنامج الأول

عرضلنا ويدجت عليها title مكتوب عليه Hello, QtRuby! وبمساحة 300 عرض و 200 ارتفاع

1- ضيف ال Qt4 لساحة العمل

```
require "Qt4"
```

2- انشئ application object ليعبر عن التطبيق (لاحظ هو application واحد فقط الذى يتم انشائه)عن طريق

```
Qt::Application.new
```

كالتالى

```
app=Qt::Application.new(ARGV)
```

بيتم تجهيزه بال ARGV اللى هتتباصى للبرنامج

3- انشئ الويدجت بإستخدام ال Widget class

```
w=Qt::Widget.new
```

4- حدد ال title بإستخدام الميثود setWindowTitle الموجودة بال w object

```
w.setWindowTitle "Hello, QtRuby!"
```

5- اعمل اعادة تحديد للمساحة بإستخدام الميثود resize الموجودة بال w object

```
w.resize(300, 200) #width, height
```

6- اظهر ال widget

```
w.show
```

7- نفذ ال application بإستخدام ال exec method الموجود بال app object

```
app.exec
```

رائع جدا بس مش هو دا اللى عايزينه احنا عايزين نكتب ب إسلوب *افضل*

انشئ class بإسم MyWidget فى my_widget.rb على فرض ان ال entry point للبرنامج هتكون دائما main.rb

```
class MyWidget < Qt::Widget
  def initialize
    super()
    init_comp
  end
```

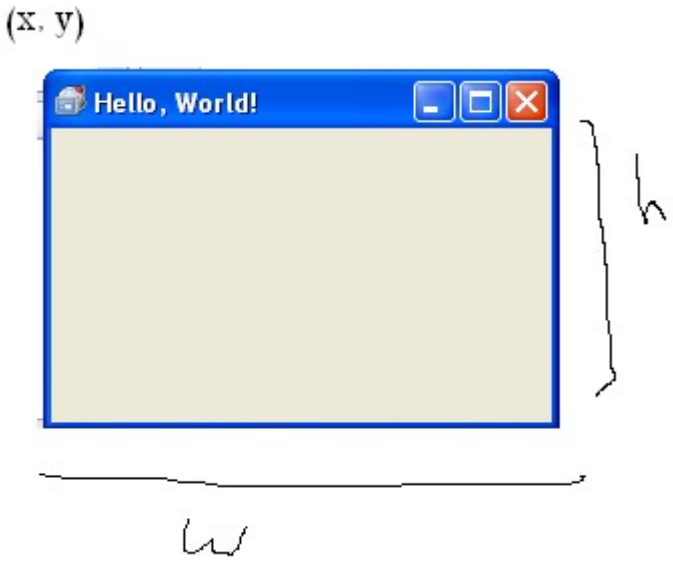
end

ال init_comp دى هتكون الميثود المسئولة عن تجهيز ال components على الويدجت او الفورم الخ..
وفى المثال هنا معرفة كالتالى

```
def init_comp

  setWindowTitle('Hello, World!')
  #resize(200, 300) #w, h
  setGeometry(300, 300, 250, 150) #x, y, w, h
  setToolTip("Simple Widget")
  setWindowIcon(Qt::Icon.new('icons/apps/preferences-desktop-remote-desktop')) #Set Icon..

end
```

فى حاجات جديدة وهما setGeometry و setWindowIcon و setToolTip

.setGeometry(x, y, w, h)



.setToolTip

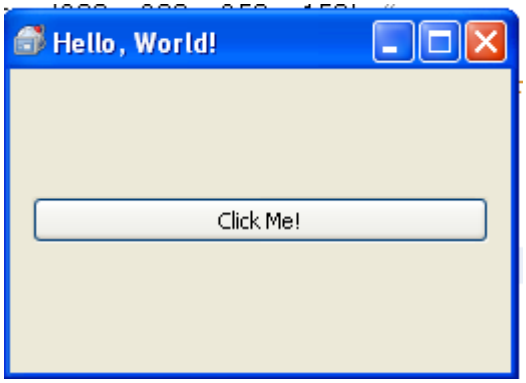
لعرض tooltip

تقدر تحط فيها markup ايضا

.setWindowIcon

وبتاخد Icon object لتحديد ايكون على الويدجت
سهل مش كدا؟؟

عايزين ننشئ حاجة مشابهه لكدا



تمام ويدجت وعليه button مكتوب عليه Click Me!
اوكى تمام هنحتاج نتكلم الأول عن الإستراتيجى بتاعتنا
اولا الويدجت بأبسط صورة عبارة عن مستطيل كدا

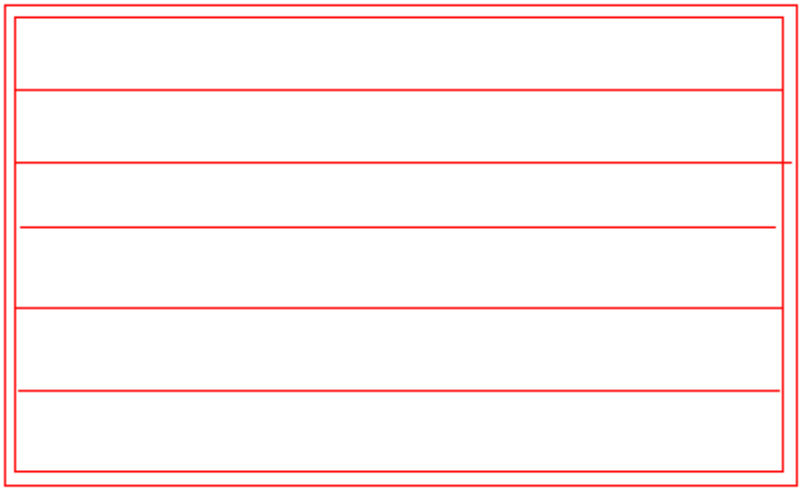
محتاجين نضيف ليه حاجة اسمها Layout لنقدر نضيف اى ويدجتس اخرى جواه (اسلوب مشهور جدا)
ال Layout دا عبارة عن صندوق داخلى جوا الويدجت الأصلى

ممكن يشمل Layouts اخرى وممكن يشمل فى نفس الوقت كنترولز (راجع جزئية ال RubyGnome بخصوص ال Hbox, Vbox)
المهم فى عندنا اكتر من نوع لل Layout فى افقى وفى رأسى.. الأفقى بيتم فيه اضافة الويدجتس/اللاي اوتس بصورة افقيه والرأسى بيتم إضافتهم بصورة رأسية..

مثلا اللى داخل الويدجت دا الأفقى

وعند إضافة كنترولز بيتم التحجيم وتجهيز مكان جديد للإضافة مثلا

اما الرأسى فبيتم الإضافة رأسيا كما فى الصورة التالية مثلا



انا ديما بعتبر ان ال Layout الرئيسي هو رأسى وبعمل جواه Layouts داخلية بس ال Main بيكون Vbox
تعالى نجرب مثال على الإضافة الرأسية

```
  def init_comp
    setWindowTitle('Hello, World!')
    #resize(200, 300) #w, h
    setGeometry(300, 300, 250, 150) #x, y, w, h
    setToolTip("Simple Widget")
    setWindowIcon(Qt::Icon.new('icons/apps/preferences-desktop-remote-desktop')) #Set Icon..
    vlay=Qt::VBoxLayout.new

#add 5 buttons vertically!
(1..5).each { |i|
    vlay.addWidget(Qt::PushButton.new(i.to_s))

  }
    setLayout(vlay)

  end
```

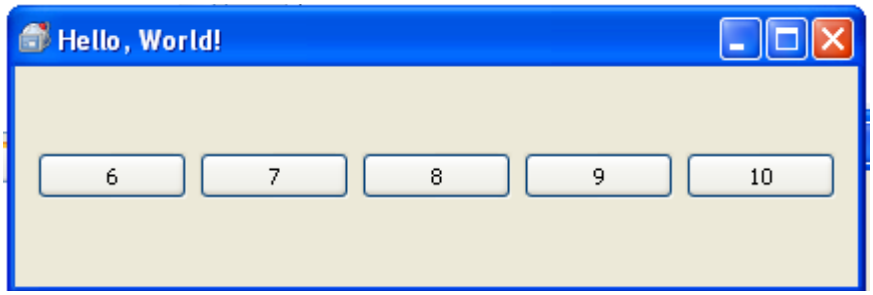
لإنشاء Layout رأسى إستخدام VboxLayout class
لإنشاء push button استخدم PushButton class وباصى للمشيد ال text
لإضافة ويدجت معين ل layout استخدم addWidget

لتحديد ال Layout الأساسى استخدم setLayout

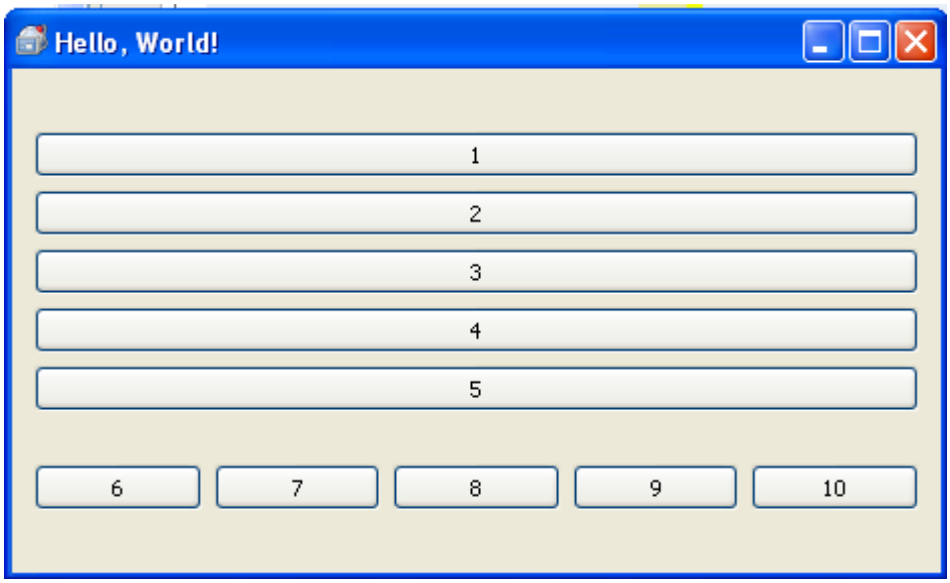
طب لو افقى ؟ تقدر تطبق نفس السابق كالتالى

```
vlay=Qt::VBoxLayout.new

hlay=Qt::HBoxLayout.new {|lay|
  (6..10).each{|i|
    lay.addWidget(Qt::PushButton.new(i.to_s))
  }
  vlay.addLayout(lay)
```

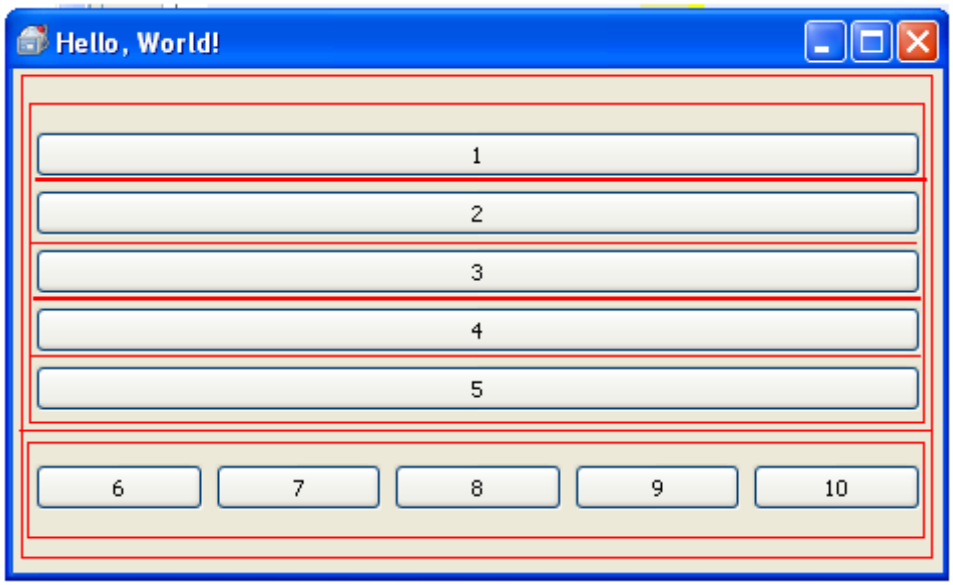
لإضافة Layout ل layout معين استخدم الميثود addLayout



طب لو عايزين نضيف الإتنين بنفس الوقت ال 5 الرأسى وال 5 الأفقى ؟



التصميم




```
def init_comp
   setWindowTitle('Hello, World!')
   #resize(200, 300) #w, h
   setGeometry(300, 300, 250, 150) #x, y, w, h
   setToolTip("Simple Widget")
   setWindowIcon(Qt::Icon.new('icons/apps/preferences-desktop-remote-desktop')) #Set Icon..

   mainlayout=Qt::VBoxLayout.new
   vlay=Qt::VBoxLayout.new {|lay|

     (1..5).each { |i|
        lay.addWidget(Qt::PushButton.new(i.to_s))

     }
     mainlayout.addLayout(lay)
   }

   hlay=Qt::HBoxLayout.new {|lay|
     (6..10).each{|i|
       lay.addWidget(Qt::PushButton.new(i.to_s))
     }
     mainlayout.addLayout(lay)
   }

   setLayout(mainlayout)

  end
```

مثال Hello World

اولا لكدا لازم نتكلم عن تكنيك ال SIGNAL/SLOT

ال signal هى إشارة بتتبعت من وإلى
ال slot هى ال هاندلر زى مابنقول او الميثود اللى هيتم تنفيذها حين وصول الإشارة من اوبجكت معين
مثال ال button ليه كذا سيجنال ممكن يرسلها (بناء على التفاعل مع المستخدم او تغير فى الحالة) زى clicked صح؟هيتم ارسالها إلى مين؟
الى ال ويدجت اللى فيه ال button تمام لو مش اتحددت slot يتم تنفيذها حين الضغط فالمستخدم لو ضغط من هنا لمليون سنة قدام مش هيحصل تغيير فكل اللى عليك تنشئ method او handler يعنى وتعرفها على انها SLOT وتربطها بال SIGNAL دى فيتم تنفيذها كل مرة تتبعت ال SIGNAL دى..
هنشوف حالا فى المثال دا

```
class MyWidget < Qt::Widget
  slots 'onBtnClickMeClicked()'

  def initialize
    super()
    @times=0
    init_comp
  end

  def init_comp
    setWindowTitle('Hello, World!')
    #resize(200, 300) #w, h
    setGeometry(300, 300, 250, 150) #x, y, w, h
    setToolTip("Simple Widget")

    vlay=Qt::VBoxLayout.new
    #Add a button..
    @btnclickme=Qt::PushButton.new("Hi")
    connect(@btnclickme, SIGNAL('clicked()'),  SLOT('onBtnClickMeClicked()'))
    vlay.addWidget(@btnclickme)
    setLayout(vlay)

  end

  def onBtnClickMeClicked
      Qt::MessageBox::information(self, "Hello!", "Welcome to Qt!")
```

```
end
```

لاحظ السطر

```
slots 'onBtnClickMeClicked()'
```

هنا بنعرف مصفوفة من الميثودز اللى عايزينها يتم اعتبارهم SLOTs
انشأنا layout رأسى وضفنا ليه button وضفناه لل layout وحددنا ال layout دا بإنه الأساسى للويدجت

ربطنا الحدث clicked المرسل من ال @btnclickme تحديدا بال slot onBtnClickMeClicked كالتالى

```
connect(@btnclickme, SIGNAL('clicked()'),  SLOT('onBtnClickMeClicked()'))
```

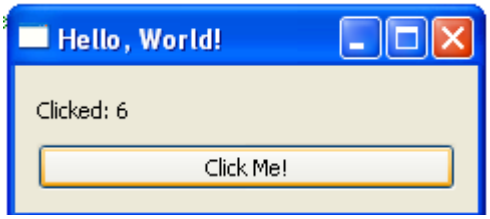
ال onBtnClickMeClicked معرفة كالتالى

```
def onBtnClickMeClicked
    Qt::MessageBox::information(self, "Hello!", "Welcome to Qt!")
end
```

بتعرض لينا رسالة فى ال title مكتوب Hello! وفى المحتوى مكتوب Welcome to Qt و self هى بوينتر للأب

المثال الشهير Click Me

بعد الضغط 6 مرات



الفكرة العامة ان يبقة عندنا متغير نحسب فيه عدد الضغطات ودى نقدر نحسبها بإننا نتتبع كل ضغطة تتم على ال button وبعد كدا نعدل التكست على label معين

```
class MyWidget < Qt::Widget
  slots 'onBtnClickMeClicked()'

  def initialize
    super()
    @times=0
```

```
    init_comp
  end

  def init_comp
    setWindowTitle('Click Me!')
    #resize(200, 300) #w, h
    setGeometry(300, 300, 250, 150) #x, y, w, h
    setToolTip("Simple Widget")

    vlay=Qt::VBoxLayout.new
    #Add counter label..
    @lblcounter=Qt::Label.new
    vlay.addWidget(@lblcounter)
    #Add a button..
    @btnclickme=Qt::PushButton.new("Click Me!")
    connect(@btnclickme, SIGNAL('clicked()'),  SLOT('onBtnClickMeClicked()'))
    vlay.addWidget(@btnclickme)
    setLayout(vlay)

  end

  def onBtnClickMeClicked()
    @times += 1 #Inc times.
    updateCounterLabel
  end

  private
  def updateCounterLabel
    @lblcounter.setText("Clicked: #{@times}")
  end
end
```

وايضا عرفنا متغير @times لنحسب عدد الضغطات

انشأنا layout رأسى لنضع فيه ال widgets بتاعتنا

```
  vlay=Qt::VBoxLayout.new
```

انشأنا ال label, button وضفناهم ليه

```
    @lblcounter=Qt::Label.new
    vlay.addWidget(@lblcounter)
    #Add a button..
    @btnclickme=Qt::PushButton.new("Click Me!")
    vlay.addWidget(@btnclickme)
```

ربطنا الحدث clicked المرسل من ال @btnclickme تحديدا بال slot onBtnClickMeClicked كالتالى

```
    connect(@btnclickme, SIGNAL('clicked()'),  SLOT('onBtnClickMeClicked()'))
```

حددنا ال layout الأساسى للبرنامج

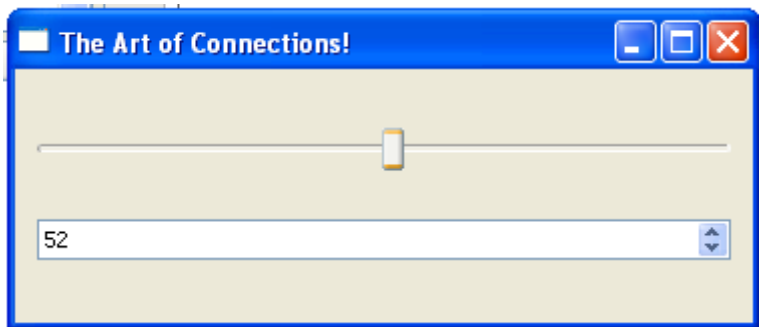
```
  setLayout(vlay)
```

تمام تعريف ال slot بقة

```
def onBtnClickMeClicked
  @times += 1 #Inc times.
  updateCounterLabel
end
```

هتقوم بزيادة المتغير @times بمقدار 1 عند كل استدعاء وعمل تحديث لل label بإستدعاء updateCounterLabel المعرفة كالتالى

```
private
def updateCounterLabel
  @lblcounter.setText("Clicked: #{@times}")
end
```



هنا عندنا slider وعندنا spinner تم انشائهم كالتالى

```
  slider=Qt::Slider.new(Qt::Horizontal)
  slider.setRange(0, 100)

  spinner=Qt::SpinBox.new
  spinner.setRange(0, 100)
```

تم تحديد مداهم بإستخدام ال setRange وبتاخد معاملين قيمة صغرى وقيمة كبرى
لاحظ ال slider ممكن يكون افقى او رأسى ودى بتحددها من خلال المشيد بتحديد الإتجاه من خلال Qt::Horizontal او Qt::Vertical

نربط حدوث اى تغيير فى القيمة الخاصة بال slider بإنها تحدث القيمة دى على ال spinner بإستخدام ال valueChanged, setValue

```
connect(slider, SIGNAL('valueChanged(int)'), spinner, SLOT('setValue(int)'))
```

ونفس الشئ نربط اى تغيير فى القيمة الخاصة بال spinner بتحديث القيمة على ال slider كالتالى

```
connect(spinner, SIGNAL('valueChanged(int)'), slider, SLOT('setValue(int)'))
```

نضيفهم لل layout

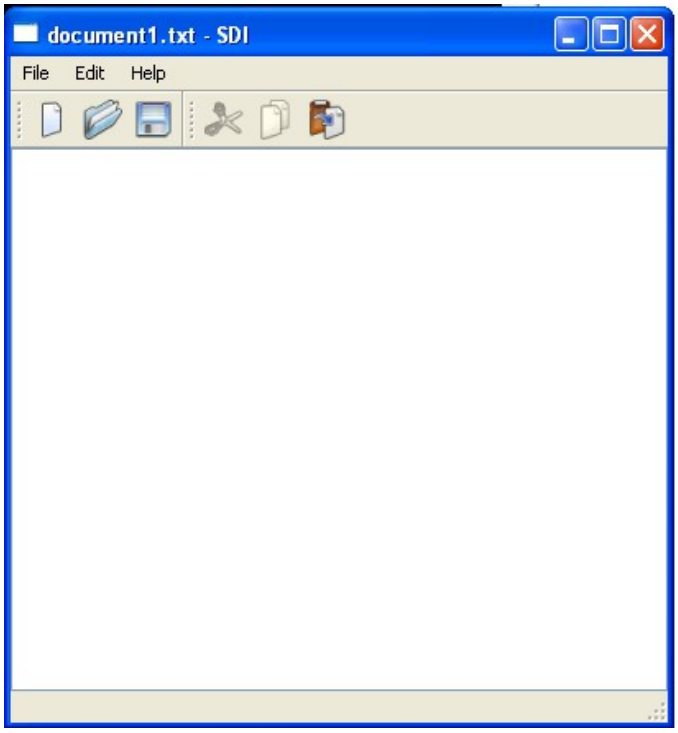
```
  vlay.addWidget(slider)
  vlay.addWidget(spinner)
```

نحدد ال layout الأساسى

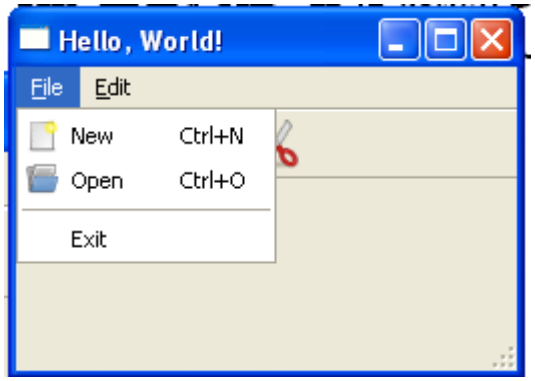
```
  setLayout(vlay)
```

Menus/ToolBars

تمام برامج كتير بتعرض وظايفها على صورة منيو واهم الوظايف فيها على صورة toolbar مثلا اى تكست اديتور



تمام Qt بتنتهج اسلوب للتعامل اسمه ال actions ال action بيعبر عن اى item ظاهر فى منيو او فى تولبار بيسمى action ال action دا ليه عدة وظايف مثلا صورة عليه.. اختصار.. نص مساعدة!
فى المثال القادم هننشئ حاجة مشابهه لكدا



عندنا 2 menu وتولبار
اول خطوة انشاء ال actions

```
newaction=Qt::Action.new(Qt::Icon.new('actions/document-new.png'),"New",self)
 #set shortcut
 newaction.setShortcut("Ctrl+N")
 newaction.setStatusTip("Create a new file.")
 connect(newaction, SIGNAL("triggered()"), SLOT("onNew()"))

 openaction=Qt::Action.new(Qt::Icon.new('actions/document-open.png'),"Open",self)
 #set shortcut
 openaction.setShortcut("Ctrl+O")
 openaction.setStatusTip("Open a file")
 connect(openaction, SIGNAL("triggered()"), SLOT("onOpen()"))

 copyaction=Qt::Action.new(Qt::Icon.new('actions/edit-copy.png'),"Copy",self)
 #set shortcut
 copyaction.setShortcut("Ctrl+C")
 copyaction.setStatusTip("Copy")
 connect(copyaction, SIGNAL("triggered()"), SLOT("onCopy()"))

 cutaction=Qt::Action.new(Qt::Icon.new('actions/edit-cut.png'),"Cut",self)
 #set shortcut
 cutaction.setShortcut("Ctrl+X")
 cutaction.setStatusTip("Cut")
 connect(cutaction, SIGNAL("triggered()"), SLOT("onCut()"))


 exitaction=Qt::Action.new("Exit", self)
 exitaction.setStatusTip("Exit..")
 connect(exitaction, SIGNAL('triggered()'), $qApp, SLOT('closeAllWindows()'))
```

كلهم متشابهين عشان كدا هنشرح اول واحدة واخر واحدة

```
newaction=Qt::Action.new(Qt::Icon.new('actions/document-new.png'),"New",self)
 #set shortcut
 newaction.setShortcut("Ctrl+N")
 newaction.setStatusTip("Create a new file.")
```

```
connect(newaction, SIGNAL("triggered()"), SLOT("onNew()"))
```

لإنشاء ال Action استخدم Action class المشيد الخاص بيه بياخد 3 معاملات الأول عبارة عن الأيكون
والتانى التسكت والتالت هو النافذة اللى بينتمى ليها
لتحديد ال Shortcut استخدم setShortcut الموجود بال action object
لتحديد ال status tip استخدم setStatusTip الموجودة بال action object
وال action ليه ال SGINAL المسماه ب triggered ودى هى اللى بنربطها فى حال انشاء ال connections

```
exitaction=Qt::Action.new("Exit", self)
exitaction.setStatusTip("Exit..")
connect(exitaction, SIGNAL('triggered()'), $qApp, SLOT('closeAllWindows()'))
```

نفس السابقة انشأنا Action بالتكست فقط والنافذة الأب
حددنا ال status tip
اخيرا ربطنا ال triggered signal الصادرة من ال action بال SLOT closeAllWindows الخاصة بال $qApp

ماهو ال $qApp ؟ متغير عام بيشير لل application object...
ال closeAllWindows هى SLOT معرفة مسبقا لغلق كل النوافذ ومن ثم غلق الأبلكيشن

statusBar عند اول استدعاء ليها هتقوم بإظهار ال statusbar فى الويدجت
إضافة منيو

```
mbar=menuBar #fetch it.

filemenu=mbar.addMenu("&File")
filemenu.addAction(newaction)
filemenu.addAction(openaction)
filemenu.addSeparator
filemenu.addAction(exitaction)


editmenu=mbar.addMenu("&Edit")
editmenu.addAction(copyaction)
editmenu.addAction(cutaction)
```

اولا خد رفرنس من ال menuBar ميثود او تقدر تستخدمها مباشرة فى كل مرة حسب راحتك..
لإنشاء منيو استخدم .addMenu الموجودة بال menuBar object
لإضافة action لمنيو معينة استخدم addAction وباصي ليها ال action اللى جهزته مسبقا
لإضافة separator استخدم .addSeparator
لإضافة منيو داخلية استخدم .addMenu من ال menu object

التعامل مع التولبار
لإضافة جزء جديد بالتولبار بيعبر عن منطقة معينة مثلا ال file menu او كدا استخدم addToolBar مع
اسم المنطقة (لاحظ بين كل استدعاء ليها هيتم انشاء separator بين عناصر المناطق)
لإضافة actions ليها استخدم addAction الموجودة بال toolbar object

```
fbar=addToolBar('file') #Toolbar for file menu items..
fbar.addAction(newaction)
fbar.addAction(openaction)
```

```
    ebar=addToolBar("edit") #Toolbar for edit menu items..
    ebar.addAction(copyaction)
    ebar.addAction(cutaction)
```

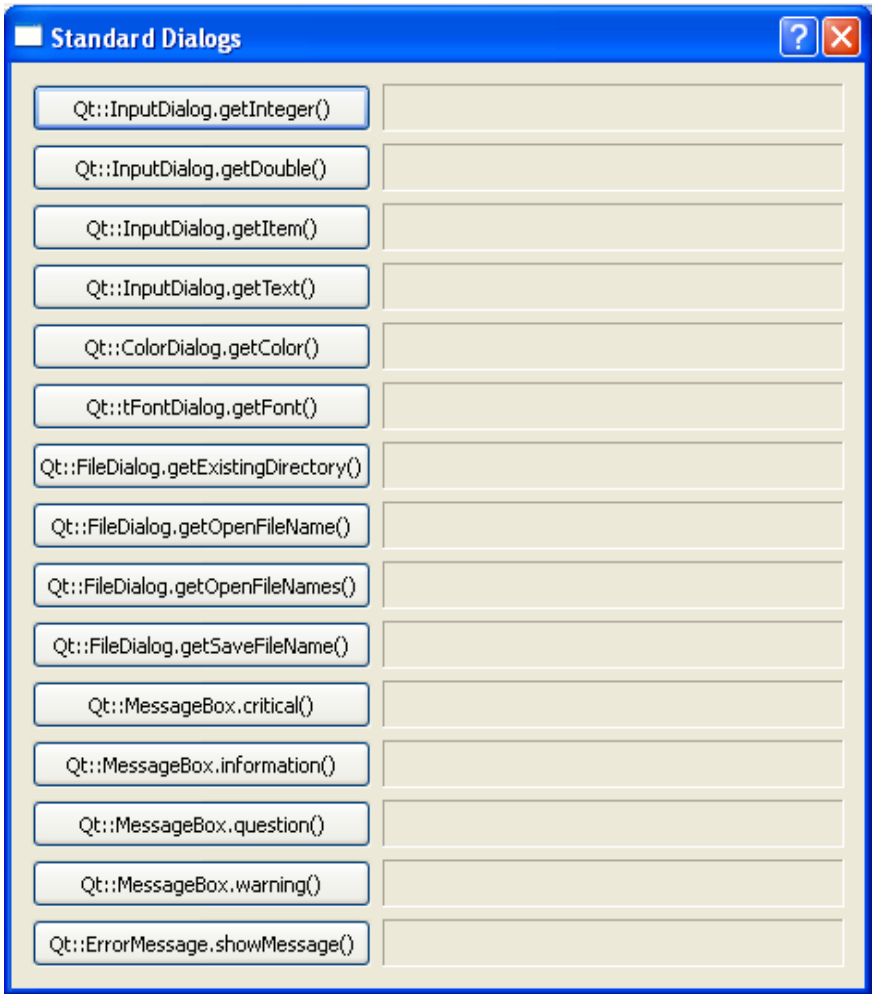
كود المثال

```
class MyWindow < Qt::MainWindow
  slots 'onNew()', 'onOpen()', 'onCopy()', 'onCut()'

  def initialize
    super()
    init_comp
  end

  def init_comp
    setWindowTitle('Hello, World!')
    #resize(200, 300) #w, h
    setGeometry(300, 300, 250, 150) #x, y, w, h
    setToolTip("Simple Widget")

    newaction=Qt::Action.new(Qt::Icon.new('actions/document-new.png'),"New",self)
    #set shortcut
    newaction.setShortcut("Ctrl+N")
    newaction.setStatusTip("Create a new file.")
    connect(newaction, SIGNAL("triggered()"), SLOT("onNew()"))

    openaction=Qt::Action.new(Qt::Icon.new('actions/document-open.png'),"Open",self)
    #set shortcut
    openaction.setShortcut("Ctrl+O")
    openaction.setStatusTip("Open a file")
    connect(openaction, SIGNAL("triggered()"), SLOT("onOpen()"))

    copyaction=Qt::Action.new(Qt::Icon.new('actions/edit-copy.png'),"Copy",self)
    #set shortcut
    copyaction.setShortcut("Ctrl+C")
    copyaction.setStatusTip("Copy")
    connect(copyaction, SIGNAL("triggered()"), SLOT("onCopy()"))

    cutaction=Qt::Action.new(Qt::Icon.new('actions/edit-cut.png'),"Cut",self)
    #set shortcut
    cutaction.setShortcut("Ctrl+X")
    cutaction.setStatusTip("Cut")
    connect(cutaction, SIGNAL("triggered()"), SLOT("onCut()"))


    exitaction=Qt::Action.new("Exit", self)
```

```
  exitaction.setStatusTip("Exit..")
  connect(exitaction, SIGNAL('triggered()'), $qApp, SLOT('closeAllWindows()'))


  statusBar #Show up the statusBar..

  #Add to the main menubar and a toolbar..
  mbar=menuBar #fetch it.

  filemenu=mbar.addMenu("&File")
  filemenu.addAction(newaction)
  filemenu.addAction(openaction)
  filemenu.addSeparator
  filemenu.addAction(exitaction)


  editmenu=mbar.addMenu("&Edit")
  editmenu.addAction(copyaction)
  editmenu.addAction(cutaction)

  fbar=addToolBar('file') #Toolbar for file menu items..
  fbar.addAction(newaction)
  fbar.addAction(openaction)

  ebar=addToolBar("edit")
  ebar.addAction(copyaction)
  ebar.addAction(cutaction)
end

def onNew
  puts "New action!"
end

def onOpen
  puts "Open action!"
end

def onCopy
  puts "Copy action!"
end

def onCut
  puts "Cut action!"
end
```

end

ال Dialogs



التعامل مع ال Dialogs كتير سهل مع ال Qt فهنشرح المثال الموجود بال standard dialogs

QinputDialog

هو كلاس بيسهل التعامل مع المستخدم واخذ قيمة معينة منه سواء Integer, Double, Text, Item

getInteger

التعريف العام ليها

int **getInteger** (QWidget * *parent*, const QString & *title*, const QString & *label*, int *value* = 0, int *minValue* = -2147483647, int *maxValue* = 2147483647, int *step* = 1, bool * *ok* = 0, Qt::WindowFlags *f* = 0)

اول معامل مع النافذة الأب

الثانى هو عنوان الديالوج

الثالث هو الرسالة

الرابع هو القيمة المختارة

الخامس اقل قيمة

السادس اكبر قيمة

السابع مقدار الزيادة

الثامن هو مؤشر للمتغير اللى هيتم تخزين فيه ناتج تعامل المستخدم مع الديالوج

التاسع هو ال WindowFlags (يستحسن مراجعة ال enum من ال Qt Assistant)

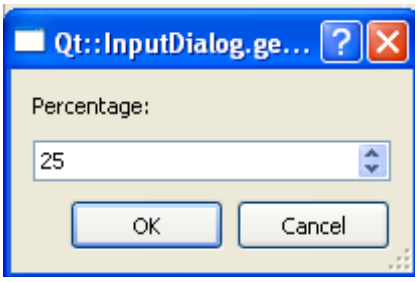
التعامل من خلال روبى

```
def setInteger()
  ok = Qt::Boolean.new
  i = Qt::InputDialog.getInteger(self, tr("Qt::InputDialog.getInteger()"),
                                 tr("Percentage:"), 25, 0, 100, 1, ok)
  if ok
    @integerLabel.text = tr("%d%" % i)
  end
end
```

لاحظ انشأنا okمن النوع Qt::Boolean ليتم مباصاته لل getInteger لحساب قيمة التفاعل مع المستخدم

استدعينا الديالوج من خلال ال static getInteger واخدت معاملات زى ماذكرنا الأب والعنوان والرسالة والقيمة المختارة والقيمة الصغرى والقيمة الكبرى ومقدار الزيادة ومتغير ok ليتم وضع ناتج التفاعل فيه

فى حال ان ال ok دا قيمته اعيدت ب true هيتم وضع تكست على label مسمى @integerLabel



getDouble

نفس تعريف سابقتها ولكن بتعمل ريترن ب double

double **getDouble** (QWidget * *parent*, const QString & *title*, const QString & *label*, double *value* = 0, double *minValue* = -2147483647, double *maxValue* = 2147483647, int *decimals* = 1, bool * *ok* = 0, (Qt::WindowFlags *f* = 0

اول معامل مع النافذة الأب

الثانى هو عنوان الديالوج

الثالث هو الرسالة

الرابع هو القيمة المختارة

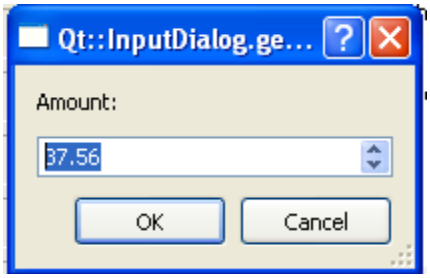
الخامس اقل قيمة

السادس اكبر قيمة

السابع مقدار الزيادة

الثامن هو مؤشر للمتغير اللى هيتم تخزين فيه ناتج تعامل المستخدم مع الديالوج

التاسع هو ال WindowFlags (يستحسن مراجعة ال enum من ال Qt Assistant)



التعامل من خلال روبى

```
def setDouble()
  ok = Qt::Boolean.new
  d = Qt::InputDialog.getDouble(self, tr("Qt::InputDialog.getDouble()"),
                                tr("Amount:"), 37.56, -10000, 10000, 2, ok)
  if ok
    @doubleLabel.text = "$%f" % d
  end
end
```

استدعينا الديالوج من خلال ال static getDouble واخدت معاملات زى ماذكرنا الأب والعنوان والرسالة والقيمة المختارة والقيمة الصغرى والقيمة الكبرى ومقدار الزيادة ومتغير ok ليتم وضع ناتج التفاعل فيه

فى حال ان ال ok دا قيمته اعيدت ب true هيتم وضع تكست على label مسمى @doubleLabel

getItem

QString **getItem** (QWidget * *parent*, const QString & *title*, const QString & *label*, const QStringList & *list*, int *current* = 0, bool *editable* = true, bool * *ok* = 0, Qt::WindowFlags *f* = 0)

parent: هو الأب

title: العنوان

label: الرسالة

list: الإختيارات المتاحة

current: رقم الإختيار الحالى (اللى هيتم عرضه) index

editable: هل المستخدم يقدر يكتب فى مكان الإختيار او لأ

ok: مؤشر للريترن

يفضل مراجعة ال WindowFlags مع ال Qt Assistant

التعامل من خلال روبى

```
def setItem()
   items = []
   items << tr("Spring") << tr("Summer") << tr("Fall") << tr("Winter")

   ok = Qt::Boolean.new
   item = Qt::InputDialog.getItem(self, tr("Qt::InputDialog.getItem()"),
                                  tr("Season:"), items, 2, false, ok)
   if ok && !item.nil?
      @itemLabel.text = item
   end
end
```

بننشئ list بإسم items مثلا ونحط فيها القيم

ونستدعى الديالوج بإستخدام ال static getItem ونباصى ليها رفرنس للأب والعنوان ومحتوى الرسالة والإختيارات والإختيار المنشط و سماحية الكتابة ورفرنس لناتج التفاعل

نختبر العائد من التفاعل والقيم اللى اختارها المستخدم وبناء عليها هنعدل @itemLabel بحيث انه يظهر الإختيار ...

getText

معرفة كالتالى

QString **getText** (QWidget * *parent*, const QString & *title*, const QString & *label*, QLineEdit::EchoMode *mode* = QLineEdit::Normal, const QString & *text* = QString(), bool * *ok* = 0, Qt::WindowFlags *f* = 0)

ميثود بتحصل على string من المستخدم

parent: الأب

title: العنوان

label: محتوى الرسالة

mode: بيعبر عن ال echo mode المستخدم اهمهم

Normal: عرض الحروف جميعا

NoEcho: عدم العرض

Password: إستبدال الحروف بنجوم

text: هو النص الإفتراضى الموجود بال LineEdit

ok: ناتج التفاعل مع المستخدم

التعامل من خلال روبى

```
def setText()
   ok = Qt::Boolean.new
   text = Qt::InputDialog.getText(self, tr("Qt::InputDialog.getText()"),
                                  tr("User name:"), Qt::LineEdit::Normal,
                                  Qt::Dir::home().dirName(), ok)
   if ok && !text.nil?
      @textLabel.text = text
   end
end
```

ColorDialog

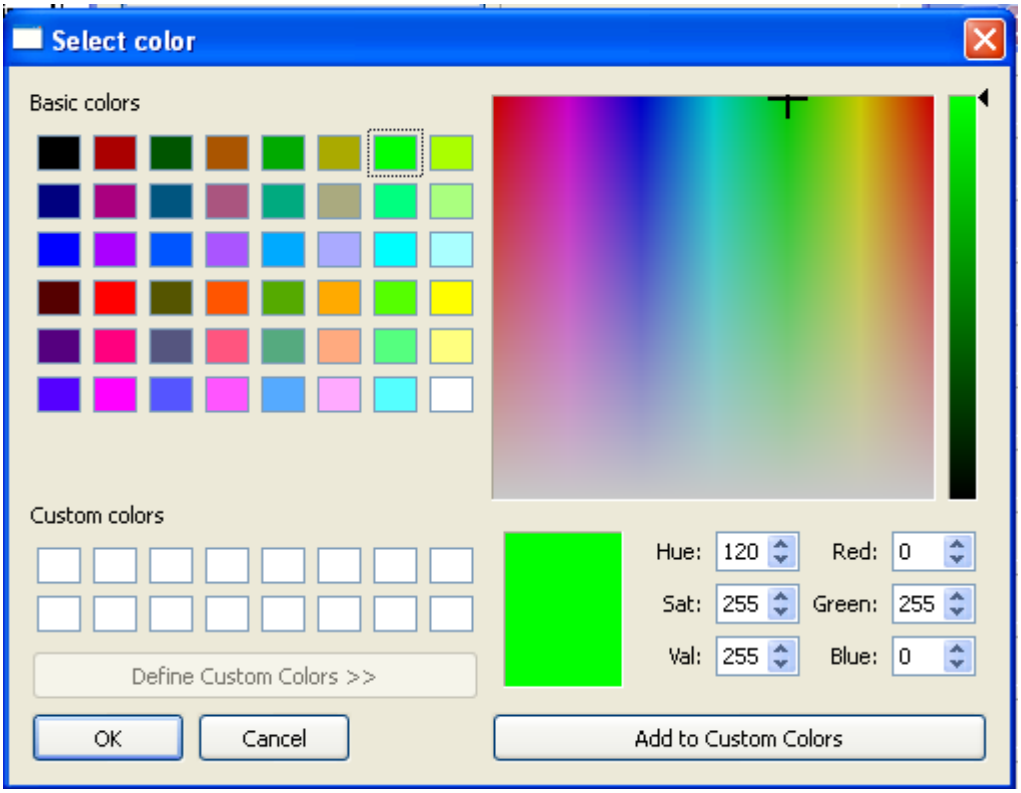
هو كلاس بيسهل علينا الحصول على لون معين من اختيار المستخدم ودا من خلال ال static getColor المعرفة كالتالى

QColor **getColor** (const QColor & *initial* = Qt::white, QWidget * *parent* = 0)

بتعرض ديالوج فيه الإختيار الأولى initial وبينتمى الديالوج ل parent (الأب)

لو المستخدم ضغط cancel هيتم اعادة invalid فللإختبار استخدم color.isValid ميثود

الإستخدام من خلال روبى




```
def setColor()
    color = Qt::ColorDialog.getColor(Qt::Color.new(Qt::green), self)
    if color.isValid()
        @colorLabel.text = color.name
    end
end
```

هنا انشأنا ديالوج بإختيار افتراضى هو اللون الأخضر ورفرنس للأب self

```
color = Qt::ColorDialog.getColor(Qt::Color.new(Qt::green), self)
```

بنختبر هل اللون isValid او لأ لو valid هيتم عرض ال name الخاص بيه

للحصول على نسبة الأخضر والأحمر والأزرق استخدم

.red

.green

,blue

FontDialog

getFont

فى الواقع الميثود دى معرفة اكثر من مرة overloaded فإحنا هناخد دى

QFont **getFont** (bool * *ok*, const QFont & *initial*, QWidget * *parent* = 0)

هنا ok بتعبر عن ناتج التفاعل مع المستخدم

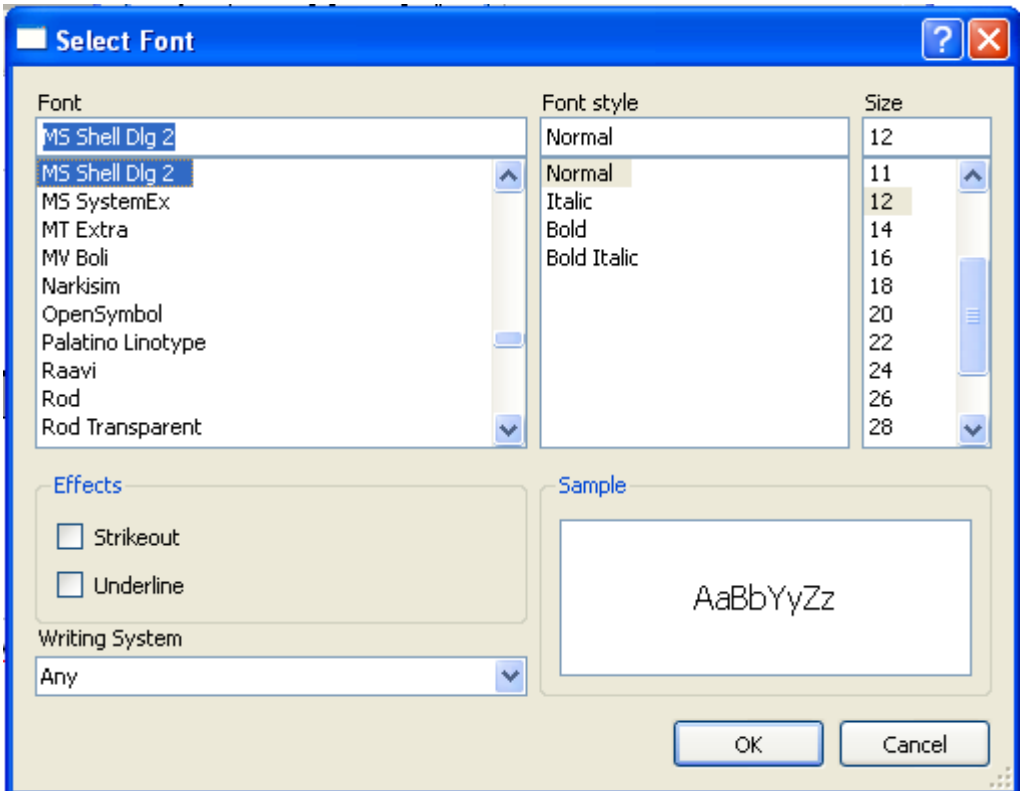
initial: الفونت اللى هيتم وضعه كبداية

parent: النافذة الأب

او تقدر تستدعيه كالتالى

QFont **getFont** (bool * *ok*, QWidget * *parent* = 0)

وهيتم وضع ال font الإفتراضى للأبلكيشن ك initial



التعامل مع روبى

```
def setFont()
  ok = Qt::Boolean.new
  font = Qt::FontDialog.getFont(ok, Qt::Font.new(@fontLabel.text), self)
  if ok
    @fontLabel.text = font.key()
  end
end
```

ok متغير ليعبر عن الحالة

نعمل run للديالوج بإستخدام ال static getFont واللى هتاخد ok و Font Object ليعبر عن الفونت الأولى
وهنا بدل مانكتب التعريف ناخده مباشرة من fontLabel و self رفرنس للأب

اختبر ال ok

او تقدر تستخدمه كما ذكرنا بالأعلى

```
def setFont()
    ok = Qt::Boolean.new
    font = Qt::FontDialog.getFont(ok, self)
    if ok
        @fontLabel.text = font.key()
    end
end
```

FileDialog

كلاس بيتيح ليك بسهولة امكانية اختيار ملفات او مجلدات

getDir

QString **getExistingDirectory** (QWidget * *parent* = 0, const QString & *caption* = QString(), const QString & *dir* = QString(), Options *options* = ShowDirsOnly)

للحصول على مجلد موجود من قبل

parent: النافذة الأب

caption ال caption

dir: الفولدر اللى هيتم البدأ منه

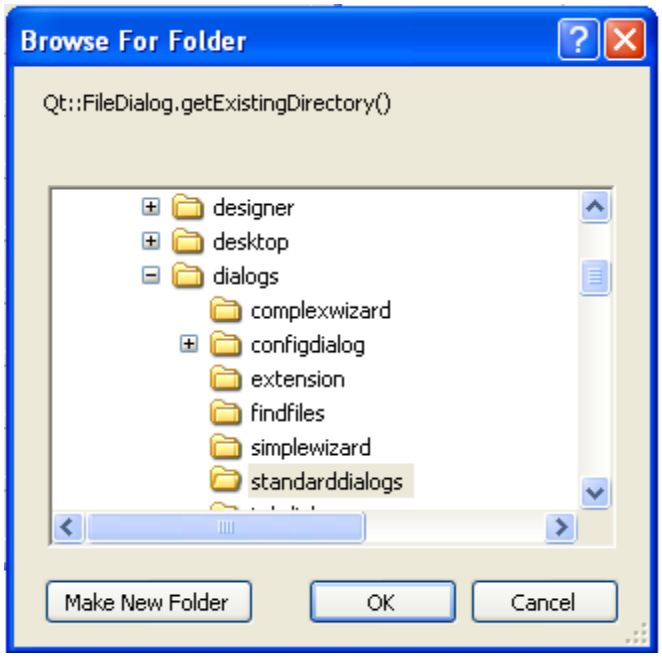
options: مجموعة من الأوبشنز الإفتراضى عرض الفولدرات فقط

اهمهم

ShowDirsOnly, DontResolveSymLinks

لمنع حل ال symlinks وهى بيتم حلها افتراضيا...

التعامل من خلال روبى




```
def setExistingDirectory()
   directory = Qt::FileDialog.getExistingDirectory(self,
                        tr("Qt::FileDialog.getExistingDirectory()"),
                        @directoryLabel.text,
                        Qt::FileDialog::DontResolveSymlinks |
                        Qt::FileDialog::ShowDirsOnly)
   if !directory.nil?
      @directoryLabel.text = directory
   end
end
```

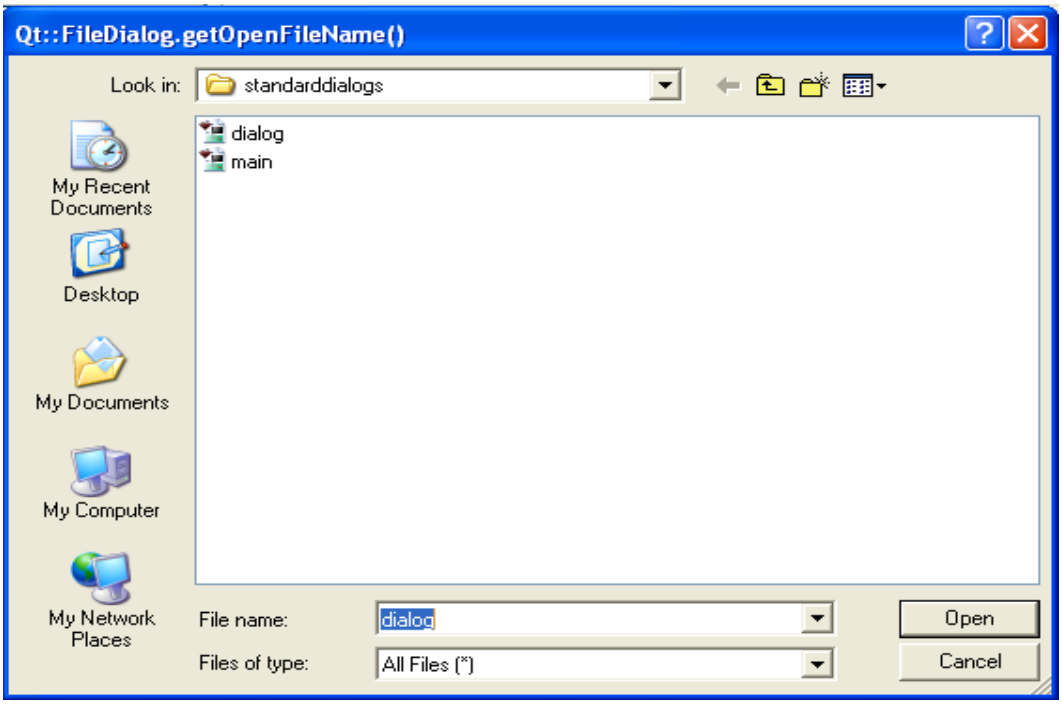
.getOpenFileName

لإختيار ملف واحد

QString **getOpenFileName** (QWidget * *parent* = 0, const QString & *caption* = QString(), const QString & *dir* = QString(), const QString & *filter* = QString(), QString * *selectedFilter* = 0, Options *options* = 0)

parent: الأب
caption: ال caption
dir: محل البداية
filter: الفيلتر اللى هيتم بناء عليه عرض الملفات
selectedFiler: الفلتر المختار
ال Options: مشابه لسابقتها ماعدا ال DirsOnly

التعامل من خلال روبى




```
def setOpenFileName()
   fileName = Qt::FileDialog.getOpenFileName(self,
                        tr("Qt::FileDialog.getOpenFileName()"),
                        @openFileNameLabel.text,
                        tr("All Files (*);;Text Files (*.txt)"))
   if !fileName.nil?
      @openFileNameLabel.text = fileName
   end
end
```

لاحظ ال @openFileLabel.text قد يكون فيه تكست او لأ فهى محطوطة بالمثال لتتبع اخر اختيار وهكذا فى باقى الأمثلة

عمل رن للديالوج بإستخدام ال static getOpenFileName وباصينا ال caption والملف الأساسى والفلتر
لاحظ الفلتر مكتوب كالتالى

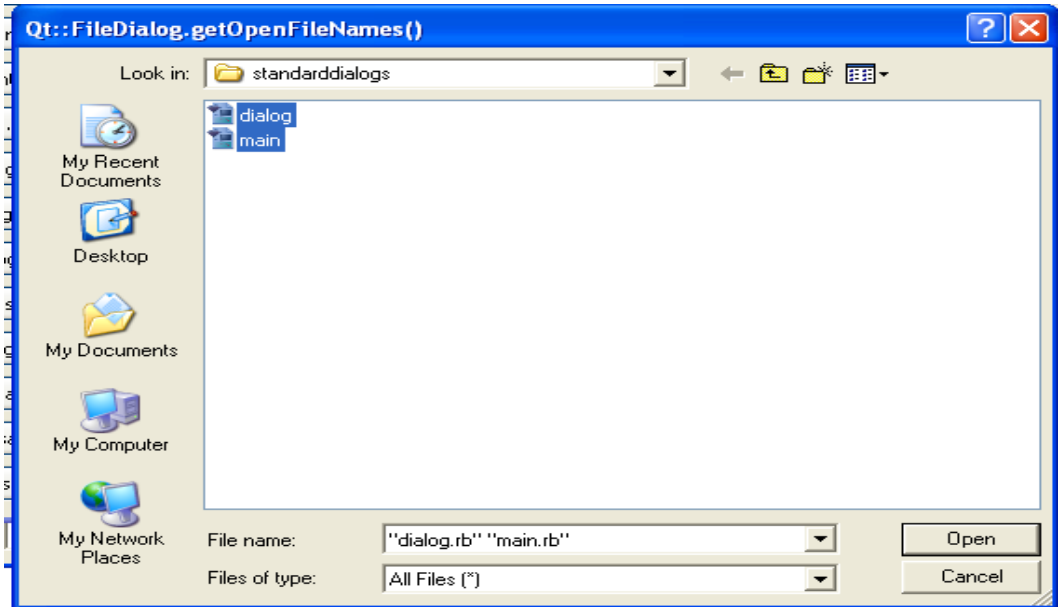
"All Files (*);;Text Files (*.txt)"

هيقدم اختيارين All Files ودى كل الفايلات *
و text Files ودى كل الفايلات اللى بتعمل match مع *.txt
لاحظ بتفصل بين كل واحد بإستخدام ;;

.getOpenFileNames

QStringList **getOpenFileNames** (QWidget * *parent* = 0, const QString & *caption* = QString(), const QString & *dir* = QString(), const QString & *filter* = QString(), QString * *selectedFilter* = 0, Options *options* = 0)

لإختيار اكتر من ملف

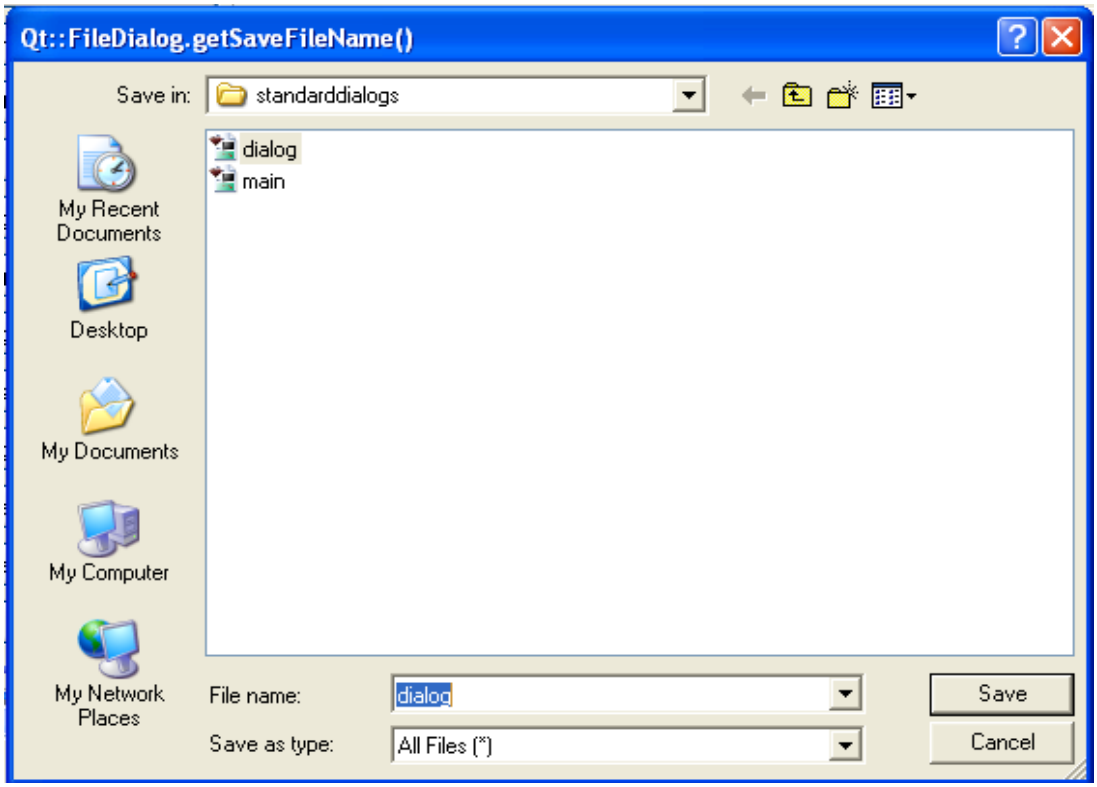



```
def setOpenFileNames()
    files = Qt::FileDialog.getOpenFileNames(
                        self, tr("Qt::FileDialog.getOpenFileNames()"),
                        @openFilesPath,
                        tr("All Files (*);;Text Files (*.txt)"))
    if files.length != 0
        @openFilesPath = files[0]
        @openFileNamesLabel.text = "[%s]" % files.join(", ")
    end
end
```

.getSaveFileName

لحفظ ملف معين

QString **getSaveFileName** (QWidget * *parent* = 0, const QString & *caption* = QString(), const QString & *dir* = QString(), const QString & *filter* = QString(), QString * *selectedFilter* = 0, Options *options* = 0)



كسابقاتها ولكنها للحفظ وليست لفتح ملف/ات

```
def setSaveFileName()
  fileName = Qt::FileDialog.getSaveFileName(self,
                    tr("Qt::FileDialog.getSaveFileName()"),
                    @saveFileNameLabel.text,
                    tr("All Files (*);;Text Files (*.txt)"))
  if !fileName.nil?
    @saveFileNameLabel.text = fileName
  end
end
```

QmessageBox

فى 4 مستويات لل MessageBox

1- Question ودى للسؤال

2- Information لإعلام بشئ معين

3- Warning تحذير

4- Critical حرج

QmessageIcon

لعرض ايكون مخصصة

QMessageBox::NoIcon	0
QMessageBox::Question	4
QMessageBox::Information	1
QMessageBox::Warning	2
QMessageBox::Critical	3

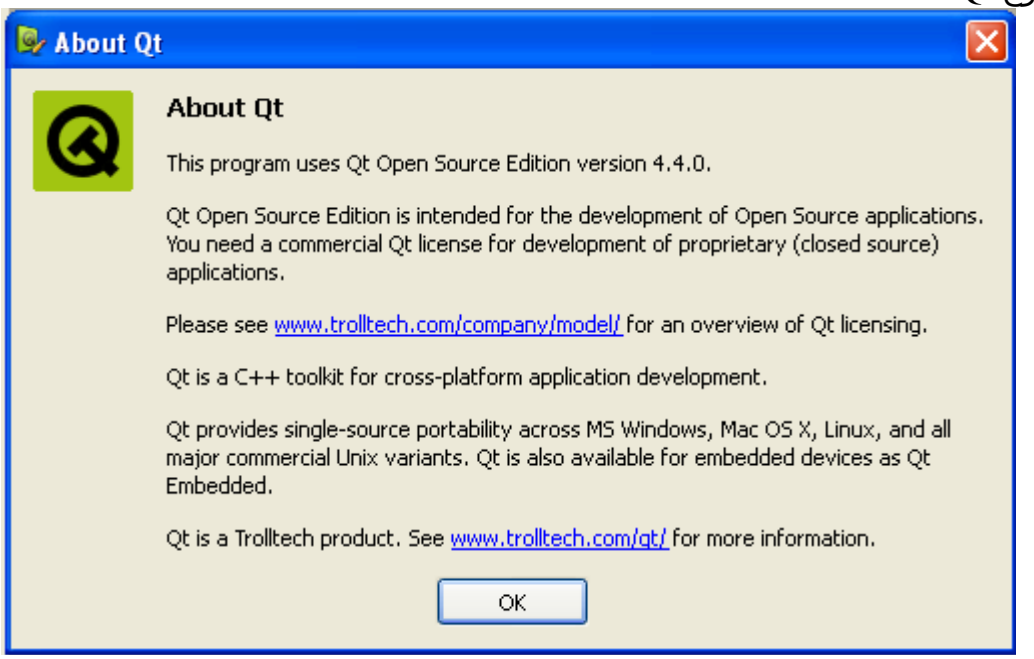
اهم الزراير

QMessageBox::Ok	زر OK
QMessageBox::Open	زر Open
QMessageBox::Save	زر Save
QMessageBox::Cancel	زر Cancel
QMessageBox::Close	زر Close
QMessageBox::Discard	زر Discard
QMessageBox::Apply	زر Apply
QMessageBox::Reset	زر Reset
QMessageBox::RestoreDefaults	زر Restore Defaults
QMessageBox::Help	زر Help
QMessageBox::SaveAll	زر Save All
QMessageBox::Yes	زر Yes
QMessageBox::YesToAll	زر Yes to all
QMessageBox::No	زر No
QMessageBox::NoToAll	زر No to all
QMessageBox::Abort	زر Abort
QMessageBox::Retry	زر Retry
QMessageBox::Ignore	زر Ignore

اهم الميثودز
ملحوظة فى كل الأمثلة التالية
parent: النافذة الأب
title: ال عنوان
text: محتوى الرسالة
buttons: تجميعة الأزرار الموضوعة على الديالوج (سبق ذكرها فى الجدول السابق)
defaultButton: الزر الإفتراضى

void **aboutQt** (QWidget * *parent*, const QString & *title* = QString())

لعرض ديالوج About Qt



StandardButton **critical** (QWidget * *parent*, const QString & *title*, const QString & *text*, StandardButtons *buttons* = Ok, StandardButton *defaultButton* = NoButton)

لعرض رسالة Critical

```
def criticalMessage()
   reply = Qt::MessageBox::critical(self, tr("Qt::MessageBox.showCritical()"),
                                    @message,
                                    Qt::MessageBox::Abort,
                                    Qt::MessageBox::Retry,
                                    Qt::MessageBox::Ignore)
   if reply == Qt::MessageBox::Abort
      @criticalLabel.text = tr("Abort")
   elsif reply == Qt::MessageBox::Retry
      @criticalLabel.text = tr("Retry")
   else
      @criticalLabel.text = tr("Ignore")
   end
end
```

StandardButton **information** (QWidget * *parent*, const QString & *title*, const QString & *text*, StandardButtons *buttons* = Ok, StandardButton *defaultButton* = NoButton)

لعرض رسالة معلومات

```
def informationMessage()
```

```
    Qt::MessageBox::information(self, tr("Qt::MessageBox.showInformation()"), @message)
    @informationLabel.text = tr("Closed with OK or Esc")
  end
```

StandardButton **question** (QWidget * *parent*, const QString & *title*, const QString & *text*, StandardButtons *buttons* = Ok, StandardButton *defaultButton* = NoButton)

لعرض سؤال

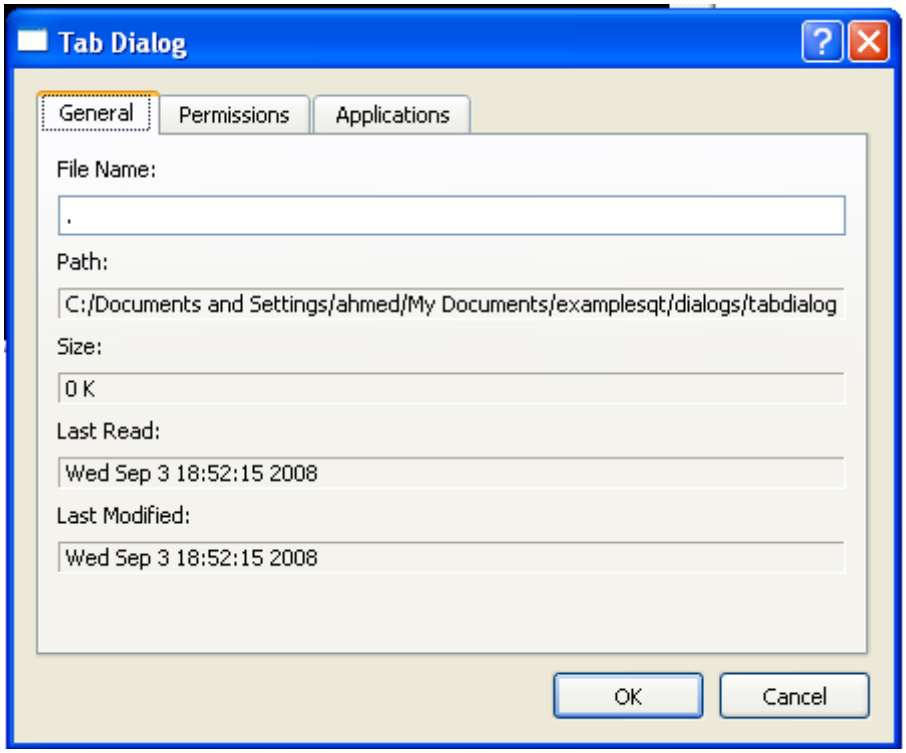
```
  def questionMessage()
    reply = Qt::MessageBox.question(self, tr("Qt::MessageBox.showQuestion()"),
                                    @message,
                                    Qt::MessageBox::Yes,
                                    Qt::MessageBox::No,
                                    Qt::MessageBox::Cancel)
    if reply == Qt::MessageBox::Yes
      @questionLabel.text = tr("Yes")
    elsif reply == Qt::MessageBox::No
      @questionLabel.text = tr("No")
    else
      @questionLabel.text = tr("Cancel")
    end
  end
```

StandardButton **warning** (QWidget * *parent*, const QString & *title*, const QString & *text*, StandardButtons *buttons* = Ok, StandardButton *defaultButton* = NoButton)

لعرض تحذير

```
  def warningMessage()
    reply = Qt::MessageBox.warning(self, tr("Qt::MessageBox.showWarning()"),
                                   @message,
                                   tr("Save &Again"),
                                   tr("&Continue"))
    if reply == 0
      @warningLabel.text = tr("Save Again")
    else
      @warningLabel.text = tr("Continue")
    end
  end
```

Tab



هنا فى الصورة السابقة عندنا 3 tabs بإسم general, permissions, applications
الفكرة العامة ان يبقة عندك 3 layout كل واحد فى ويدجت معين وتقدر تحدد هتعرض مين

هتنشئ 3 كلاسات كل واحد فيهم بيعبر عن tab من السابقين

```
class GeneralTab < Qt::Widget
        #Code omitted
end

class PermissionsTab < Qt::Widget
        #Code omitted..
end

class ApplicationsTab < Qt::Widget
        #Code omitted..
end
```

اوكى نبدأ بالتصميم للديالوج بتاعنا اللى فيه ال 3tabs دول وفيه ال general tab هو الديفولت...

```
class TabDialog < Qt::Dialog

   def initialize(fileName, parent = nil)
      super(parent)
      fileInfo = Qt::FileInfo.new(fileName)

      @tabWidget = Qt::TabWidget.new
      @tabWidget.addTab(GeneralTab.new(fileInfo), tr("General"))
      @tabWidget.addTab(PermissionsTab.new(fileInfo), tr("Permissions"))
```

```
    @tabWidget.addTab(ApplicationsTab.new(fileInfo), tr("Applications"))

    okButton = Qt::PushButton.new(tr("OK"))
    cancelButton = Qt::PushButton.new(tr("Cancel"))

    connect(okButton, SIGNAL('clicked()'), self, SLOT('accept()'))
    connect(cancelButton, SIGNAL('clicked()'), self, SLOT('reject()'))

    buttonLayout = Qt::HBoxLayout.new do |b|
      b.addStretch(1)
      b.addWidget(okButton)
      b.addWidget(cancelButton)
    end

    self.layout = Qt::VBoxLayout.new do |m|
      m.addWidget(@tabWidget)
      m.addLayout(buttonLayout)
    end

    self.windowTitle = tr("Tab Dialog")
  end
end
```

ال fileinfo دا متغير عشان نباصيه لكل كلاس السابقين عشان يكون ليه دور ولو بسيط حتى..

انشئ tab widget بإستخدام TabWidget.new وضيف ال tabs بإستخدام addTab

```
    @tabWidget = Qt::TabWidget.new
    @tabWidget.addTab(GeneralTab.new(fileInfo), tr("General"))
    @tabWidget.addTab(PermissionsTab.new(fileInfo), tr("Permissions"))
    @tabWidget.addTab(ApplicationsTab.new(fileInfo), tr("Applications"))
```

لاحظ ان هنا هيبقة الديفولت هو ال General ..

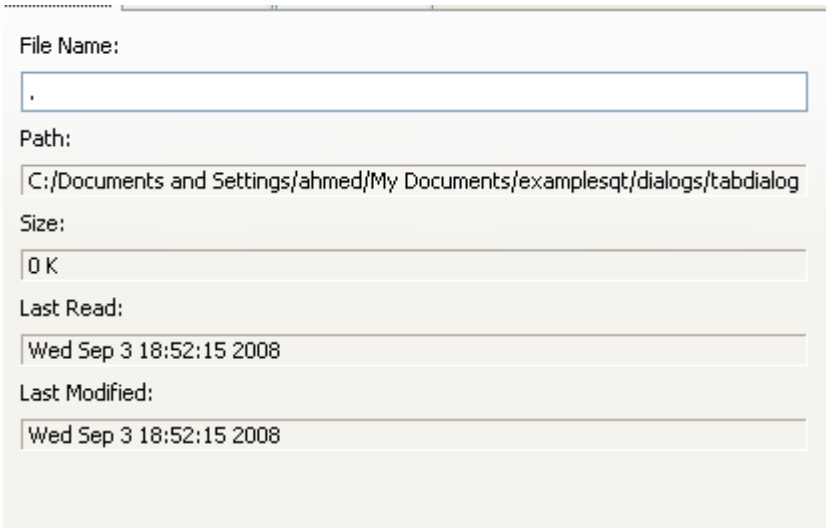
اخيرا نضيف ال buttons ال OK, CANCEL بإستخدام hboxlayout لأنهم مرصوصين افقى

```
    buttonLayout = Qt::HBoxLayout.new do |b|
      b.addStretch(1)
      b.addWidget(okButton)
      b.addWidget(cancelButton)
    end
```

انشئ VboxLayout وضيف فيه ال tab widget وال buttons layout

```
    self.layout = Qt::VBoxLayout.new do |m|
      m.addWidget(@tabWidget)
      m.addLayout(buttonLayout)
    end
```

تصميم ال General Tab



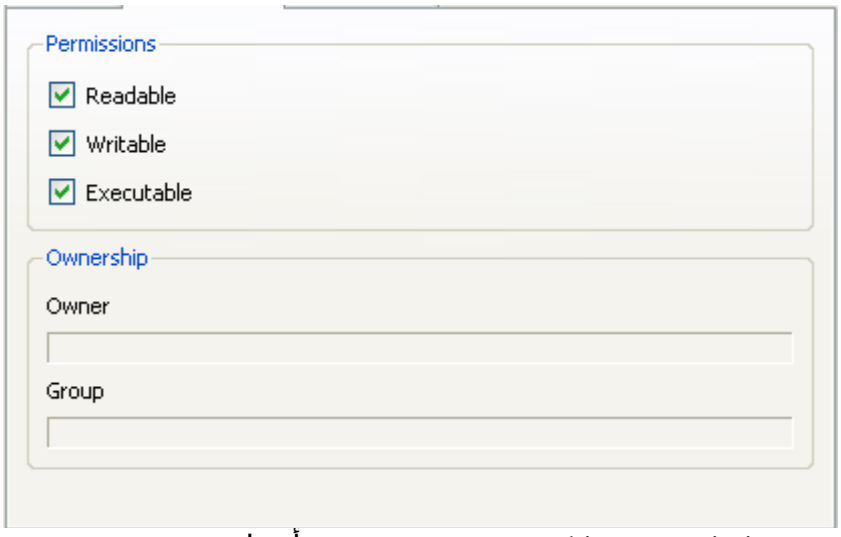
File Name -> Label
fileNameEdit-> LineEdit
pathLabel-> Label
pathValueLabel -> Label
sizeLabel-> Label
sizeValueLabel->Label
lastReadLabel->Label
lastReadValueLabel->Label
lastModLabel->Label
lastModValueLabel->Label

وتم رصهم رأسيا بإستخدام VBox Layout

```
class GeneralTab < Qt::Widget

  def initialize(fileInfo, parent = nil)
    super(parent)
    fileNameLabel = Qt::Label.new(tr("File Name:"))
    fileNameEdit = Qt::LineEdit.new(fileInfo.fileName())

    pathLabel = Qt::Label.new(tr("Path:"))
    pathValueLabel = Qt::Label.new(fileInfo.absoluteFilePath())
    pathValueLabel.frameStyle = Qt::Frame::Panel | Qt::Frame::Sunken

    sizeLabel = Qt::Label.new(tr("Size:"))
    size = fileInfo.size()/1024
    sizeValueLabel = Qt::Label.new("%d K" % size)
    sizeValueLabel.frameStyle = Qt::Frame::Panel | Qt::Frame::Sunken

    lastReadLabel = Qt::Label.new(tr("Last Read:"))
    lastReadValueLabel = Qt::Label.new(fileInfo.lastRead().toString())
```

```
    lastReadValueLabel.frameStyle = Qt::Frame::Panel | Qt::Frame::Sunken

    lastModLabel = Qt::Label.new(tr("Last Modified:"))
    lastModValueLabel = Qt::Label.new(fileInfo.lastModified().toString())
    lastModValueLabel.frameStyle = Qt::Frame::Panel | Qt::Frame::Sunken

    self.layout = Qt::VBoxLayout.new do |m|
      m.addWidget(fileNameLabel)
      m.addWidget(fileNameEdit)
      m.addWidget(pathLabel)
      m.addWidget(pathValueLabel)
      m.addWidget(sizeLabel)
      m.addWidget(sizeValueLabel)
      m.addWidget(lastReadLabel)
      m.addWidget(lastReadValueLabel)
      m.addWidget(lastModLabel)
      m.addWidget(lastModValueLabel)
      m.addStretch(1)
    end
  end
end
```

تصميم ال Permissions Tab

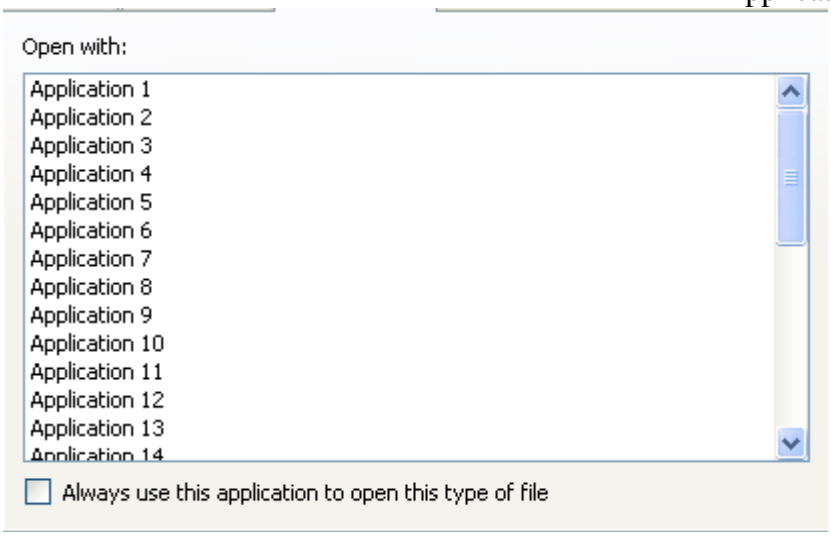


فى 2 groupbox بإسم permissions و ownership مرصوصين رأسيا
ال permissions بتشمل 3 check boxes مرصوصين رأسيا
ال ownership بتشمل label owner و LineEdit و label group ول LineEdit ليها ومرصوصين رأسيا

```
class PermissionsTab < Qt::Widget

  def initialize(fileInfo, parent = nil)
    super(parent)
    permissionsGroup = Qt::GroupBox.new(tr("Permissions"))
```

```
      readable = Qt::CheckBox.new(tr("Readable"))
      if fileInfo.readable?
        readable.checked = true
      end

      writable = Qt::CheckBox.new(tr("Writable"))
      if fileInfo.writable?
        writable.checked = true
      end

      executable = Qt::CheckBox.new(tr("Executable"))
      if fileInfo.executable?
        executable.checked = true
      end

      ownerGroup = Qt::GroupBox.new(tr("Ownership"))

      ownerLabel = Qt::Label.new(tr("Owner"))
      ownerValueLabel = Qt::Label.new(fileInfo.owner())
      ownerValueLabel.frameStyle = Qt::Frame::Panel | Qt::Frame::Sunken

      groupLabel = Qt::Label.new(tr("Group"))
      groupValueLabel = Qt::Label.new(fileInfo.group())
      groupValueLabel.frameStyle = Qt::Frame::Panel | Qt::Frame::Sunken

      permissionsGroup.layout = Qt::VBoxLayout.new do |p|
        p.addWidget(readable)
        p.addWidget(writable)
        p.addWidget(executable)
      end

      ownerGroup.layout = Qt::VBoxLayout.new do |o|
        o.addWidget(ownerLabel)
        o.addWidget(ownerValueLabel)
        o.addWidget(groupLabel)
        o.addWidget(groupValueLabel)
      end

      self.layout = Qt::VBoxLayout.new do |m|
        m.addWidget(permissionsGroup)
        m.addWidget(ownerGroup)
        m.addStretch(1)
      end
    end
  end
```

تصميم ال Applications Tab



topLabel-> Label, text (Open with)
applicationsListBox -> ListWidget, 30 Applications
alwaysCheckBox ->CheckBox

ومرصوصين رأسيا بإستخدام VBoxLayout

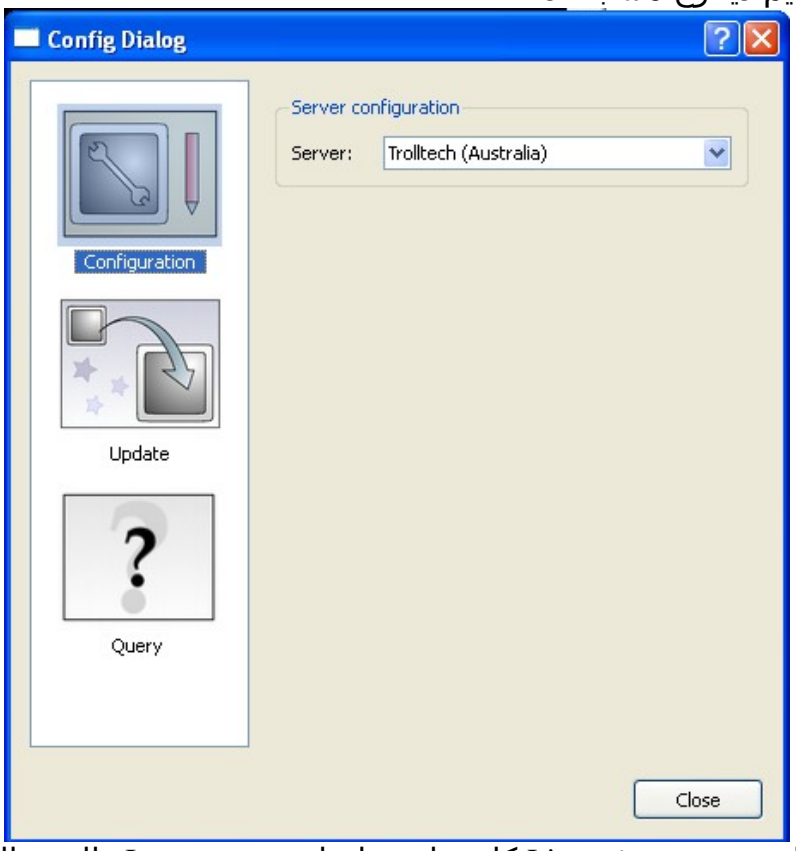
```
class ApplicationsTab < Qt::Widget

  def initialize(fileInfo, parent = nil)
    super(parent)
    topLabel = Qt::Label.new(tr("Open with:"))

    applicationsListBox = Qt::ListWidget.new
    applications = []
    (1..30).each do |i|
      applications.push tr("Application %d" % i)
    end
    applicationsListBox.insertItems(0, applications)

    if fileInfo.suffix.nil?
      alwaysCheckBox = Qt::CheckBox.new(tr("Always use this application to " +
        "open this type of file"))
    else
      alwaysCheckBox = Qt::CheckBox.new(tr("Always use this application to " +
        "open files with the extension '%s'" % fileInfo.suffix()))
    end

    self.layout = Qt::VBoxLayout.new do |l|
      l.addWidget(topLabel)
      l.addWidget(applicationsListBox)
      l.addWidget(alwaysCheckBox)
    end
```

```
    end
end
```

لاحظ ان كل ال Tabs اخدت fileInfo فى المشيد الخاص بيها (راجع اول فقرة)

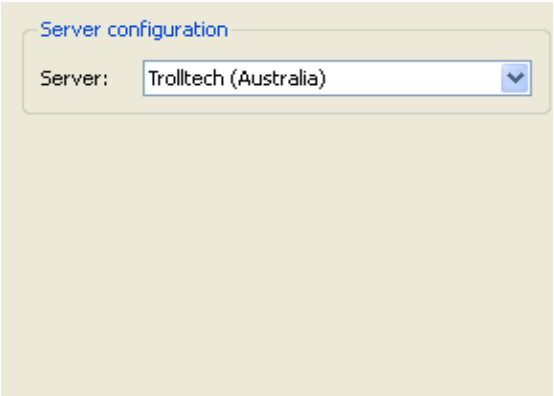
ConfigDialog

الخطوة القادمة تصميم ديالوج مشابه لدا



الفكرة كلها اننا عندنا جزئين جزء فيه List كل عناصرها على صورة Icons والجزء التانى فيه StackedWidget ودا مشابه لل Tab ولكن غير ظاهر وبيشمل ويدجتس ايضا الفكرة هنا اننا هنربط التغيير فى العنصر المختار من ال List بنفس الترتيب على ال StackedWidget ليتم اظهاره

تمام الأول هننشئ الصفحات دى



تصميم ال config page

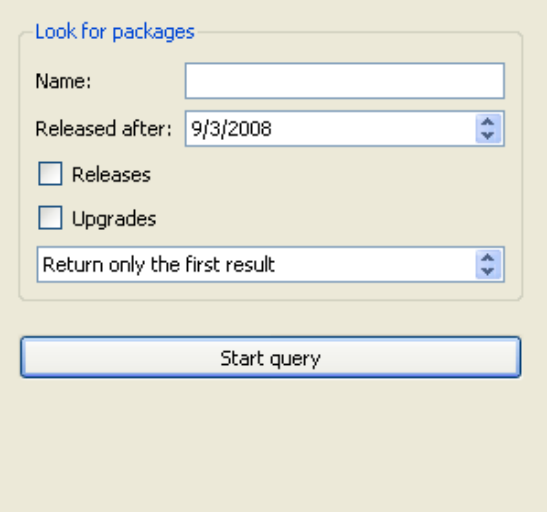
تصميمها بسيط مجرد groupbox فيها label و combobox فيها عدة عناصر وهما مرصوصين افقيا وال

group box مرصوص بصورة رأسية

```
class ConfigurationPage < Qt::Widget

  def initialize(parent = nil)
    super(parent)
    configGroup = Qt::GroupBox.new(tr("Server configuration"))

    serverLabel = Qt::Label.new(tr("Server:"))
    serverCombo = Qt::ComboBox.new do |c|
      c.addItem(tr("Trolltech (Australia)"))
      c.addItem(tr("Trolltech (Norway)"))
      c.addItem(tr("Trolltech (People's Republic of China)"))
      c.addItem(tr("Trolltech (USA)"))
    end

    serverLayout = Qt::HBoxLayout.new do |s|
      s.addWidget(serverLabel)
      s.addWidget(serverCombo)
    end

    configGroup.layout = Qt::VBoxLayout.new do |c|
      c.addLayout(serverLayout)
    end

    self.layout = Qt::VBoxLayout.new do |m|
      m.addWidget(configGroup)
      m.addStretch(1)
    end
  end
end
```

لإضافة عنصر إلى comboBox استخدم addItem الموجودة بال object..

تصميم ال update page



تصميمها مباشر جدا groupbox مرصوصة رأسيا مع button
وداخل ال groupbox العناصر مرصوصة فى grid (هنشرح ال gridlayout فى المثال)

```
class UpdatePage < Qt::Widget

  def initialize(parent = nil)
    super(parent)
    packagesGroup = Qt::GroupBox.new(tr("Look for packages"))

    nameLabel = Qt::Label.new(tr("Name:"))
    nameEdit = Qt::LineEdit.new

    dateLabel = Qt::Label.new(tr("Released after:"))
    dateEdit = Qt::DateTimeEdit.new(Qt::Date.currentDate())

    releasesCheckBox = Qt::CheckBox.new(tr("Releases"))
    upgradesCheckBox = Qt::CheckBox.new(tr("Upgrades"))

    hitsSpinBox = Qt::SpinBox.new do |h|
      h.prefix = tr("Return up to ")
      h.suffix = tr(" results")
      h.specialValueText = tr("Return only the first result")
      h.minimum = 1
      h.maximum = 100
      h.singleStep = 10
    end

    startQueryButton = Qt::PushButton.new(tr("Start query"))

    packagesGroup.layout = Qt::GridLayout.new do |p|
      p.addWidget(nameLabel, 0, 0)
      p.addWidget(nameEdit, 0, 1)
```

```
      p.addWidget(dateLabel, 1, 0)
      p.addWidget(dateEdit, 1, 1)
      p.addWidget(releasesCheckBox, 2, 0)
      p.addWidget(upgradesCheckBox, 3, 0)
      p.addWidget(hitsSpinBox, 4, 0, 1, 2)
    end

    self.layout = Qt::VBoxLayout.new do |m|
      m.addWidget(packagesGroup)
      m.addSpacing(12)
      m.addWidget(startQueryButton)
      m.addStretch(1)
    end
  end
end
```

1- انشأنا ال groupbox وحددنا ال تكست الظاهر عليها بإستخدام المشيد لل GroupBox class

```
    packagesGroup = Qt::GroupBox.new(tr("Look for packages"))
```

2- انشأنا ال Name Label وال Name LineEdit

```
    nameLabel = Qt::Label.new(tr("Name:"))
    nameEdit = Qt::LineEdit.new
```

3- انشأنا ال Date label و ال DataTime Edit

```
    dateLabel = Qt::Label.new(tr("Released after:"))
    dateEdit = Qt::DateTimeEdit.new(Qt::Date.currentDate())
```

4- انشأنا ال Check Boxes بإستخدام ال CheckBox class وبتباصى التكست الظاهر جمب الصندوق فى المشيد ليه

```
    releasesCheckBox = Qt::CheckBox.new(tr("Releases"))
    upgradesCheckBox = Qt::CheckBox.new(tr("Upgrades"))
```

5- انشأنا ال SpinBox

```
    hitsSpinBox = Qt::SpinBox.new do |h|
      h.prefix = tr("Return up to ")
      h.suffix = tr(" results")
      h.specialValueText = tr("Return only the first result")
      h.minimum = 1
      h.maximum = 100
      h.singleStep = 10
    end
```

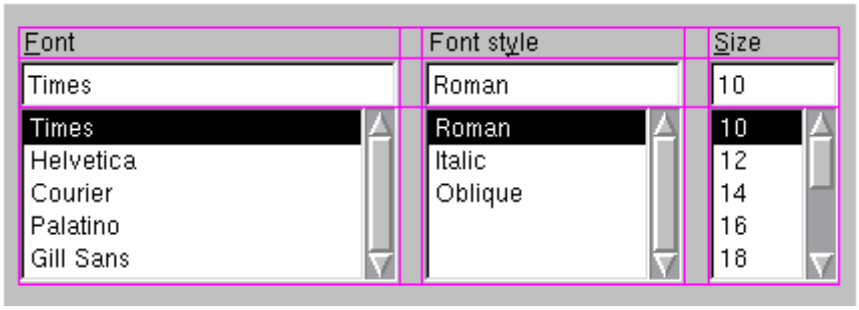
ال prefix هو ماقبل القيمة الظاهرة
ال suffix هو مابعد القيمة الظاهرة
ال specialValueText هو string تقدر تظهره عن اصغر قيمة مقبولة
ال minimum اصغر قيمة مقبولة
ال maximum اكبر قيمة مقبولة

ال singleStep مقدار الزيادة عند كل ضغطة

نيجى لجزئية ال GridLayout

```
packagesGroup.layout = Qt::GridLayout.new do |p|
  p.addWidget(nameLabel, 0, 0)
  p.addWidget(nameEdit, 0, 1)
  p.addWidget(dateLabel, 1, 0)
  p.addWidget(dateEdit, 1, 1)
  p.addWidget(releasesCheckBox, 2, 0)
  p.addWidget(upgradesCheckBox, 3, 0)
  p.addWidget(hitsSpinBox, 4, 0, 1, 2)
end
```

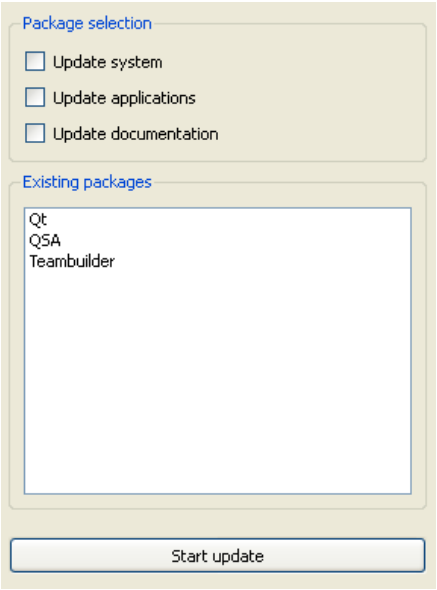
هنا بنقول ان فى GridLayout داخل ال packagesGroup وال GridLayout بيسمحلك تضيف الويدجتس فى grid حاجة مشابهه لجدول وكل مافيها خلايا بيبدأ من اعلى اليسار ودى بتبقة الخلية فى الصف 0 والعمود 0



مثلا هنا دا عبارة عن 3 صفوف و 5 اعمدة
للإضافة داخل grid استخدم addWidget (هتلقيها overloaded)

```
addWidget(widget, row, col)
addWidget(row, col, rospan, col, colspan)
```

حيث
row: رقم الصف
col: رقم العمود
rowspan: ال row span
colspan: ال colspan



تصميم ال query page
هى عبارة عن 2 groupbox و button مرصوصين رأسيا
اول groupbox تشمل 3 check boxes مرصوصين رأسيا داخلها
الثانية بتشمل ListWidget

```
class QueryPage < Qt::Widget

  def initialize(parent = nil)
    super(parent)
    updateGroup = Qt::GroupBox.new(tr("Package selection"))
    systemCheckBox = Qt::CheckBox.new(tr("Update system"))
    appsCheckBox = Qt::CheckBox.new(tr("Update applications"))
    docsCheckBox = Qt::CheckBox.new(tr("Update documentation"))

    packageGroup = Qt::GroupBox.new(tr("Existing packages"))

    packageList = Qt::ListWidget.new
    qtItem = Qt::ListWidgetItem.new(packageList)
    qtItem.text = tr("Qt")
    qsaItem = Qt::ListWidgetItem.new(packageList)
    qsaItem.text = tr("QSA")
    teamBuilderItem = Qt::ListWidgetItem.new(packageList)
    teamBuilderItem.text = tr("Teambuilder")

    startUpdateButton = Qt::PushButton.new(tr("Start update"))

    updateGroup.layout = Qt::VBoxLayout.new do |u|
      u.addWidget(systemCheckBox)
      u.addWidget(appsCheckBox)
      u.addWidget(docsCheckBox)
    end
```

```
    packageGroup.layout = Qt::VBoxLayout.new do |p|
      p.addWidget(packageList)
    end

    self.layout = Qt::VBoxLayout.new do |m|
      m.addWidget(updateGroup)
      m.addWidget(packageGroup)
      m.addSpacing(12)
      m.addWidget(startUpdateButton)
      m.addStretch(1)
    end
  end
end
```

لاحظ لإضافة عنصر جديد لل List

```
    packageList = Qt::ListWidget.new
    qtItem = Qt::ListWidgetItem.new(packageList)
    qtItem.text = tr("Qt")
    qsaItem = Qt::ListWidgetItem.new(packageList)
    qsaItem.text = tr("QSA")
    teamBuilderItem = Qt::ListWidgetItem.new(packageList)
    teamBuilderItem.text = tr("Teambuilder")
```

هنا انشأنا ال ListWidget بإسم packagesList
ولإنشاء عنصر داخلها بنستخدم ListWidgetItem ونباصى للمشيد ال listWidget object اللى هيكون
جواها.. وبعد كدا قم بتحديد التكست الظاهر من خلال ال text=

```
pages.rb
lass ConfigurationPage < Qt::Widget

  def initialize(parent = nil)
    super(parent)
    configGroup = Qt::GroupBox.new(tr("Server configuration"))

    serverLabel = Qt::Label.new(tr("Server:"))
    serverCombo = Qt::ComboBox.new do |c|
      c.addItem(tr("Trolltech (Australia)"))
      c.addItem(tr("Trolltech (Norway)"))
      c.addItem(tr("Trolltech (People's Republic of China)"))
      c.addItem(tr("Trolltech (USA)"))
    end

    serverLayout = Qt::HBoxLayout.new do |s|
```

```
        s.addWidget(serverLabel)
        s.addWidget(serverCombo)
      end

      configGroup.layout = Qt::VBoxLayout.new do |c|
        c.addLayout(serverLayout)
      end

      self.layout = Qt::VBoxLayout.new do |m|
        m.addWidget(configGroup)
        m.addStretch(1)
      end
    end
end

class QueryPage < Qt::Widget

    def initialize(parent = nil)
      super(parent)
      updateGroup = Qt::GroupBox.new(tr("Package selection"))
      systemCheckBox = Qt::CheckBox.new(tr("Update system"))
      appsCheckBox = Qt::CheckBox.new(tr("Update applications"))
      docsCheckBox = Qt::CheckBox.new(tr("Update documentation"))

      packageGroup = Qt::GroupBox.new(tr("Existing packages"))

      packageList = Qt::ListWidget.new
      qtItem = Qt::ListWidgetItem.new(packageList)
      qtItem.text = tr("Qt")
      qsaItem = Qt::ListWidgetItem.new(packageList)
      qsaItem.text = tr("QSA")
      teamBuilderItem = Qt::ListWidgetItem.new(packageList)
      teamBuilderItem.text = tr("Teambuilder")

      startUpdateButton = Qt::PushButton.new(tr("Start update"))

      updateGroup.layout = Qt::VBoxLayout.new do |u|
        u.addWidget(systemCheckBox)
        u.addWidget(appsCheckBox)
        u.addWidget(docsCheckBox)
      end

      packageGroup.layout = Qt::VBoxLayout.new do |p|
        p.addWidget(packageList)
      end
```

```
        self.layout = Qt::VBoxLayout.new do |m|
            m.addWidget(updateGroup)
            m.addWidget(packageGroup)
            m.addSpacing(12)
            m.addWidget(startUpdateButton)
            m.addStretch(1)
        end
    end
end

class UpdatePage < Qt::Widget

    def initialize(parent = nil)
        super(parent)
        packagesGroup = Qt::GroupBox.new(tr("Look for packages"))

        nameLabel = Qt::Label.new(tr("Name:"))
        nameEdit = Qt::LineEdit.new

        dateLabel = Qt::Label.new(tr("Released after:"))
        dateEdit = Qt::DateTimeEdit.new(Qt::Date.currentDate())

        releasesCheckBox = Qt::CheckBox.new(tr("Releases"))
        upgradesCheckBox = Qt::CheckBox.new(tr("Upgrades"))

        hitsSpinBox = Qt::SpinBox.new do |h|
            h.prefix = tr("Return up to ")
            h.suffix = tr(" results")
            h.specialValueText = tr("Return only the first result")
            h.minimum = 1
            h.maximum = 100
            h.singleStep = 10
        end

        startQueryButton = Qt::PushButton.new(tr("Start query"))

        packagesGroup.layout = Qt::GridLayout.new do |p|
            p.addWidget(nameLabel, 0, 0)
            p.addWidget(nameEdit, 0, 1)
            p.addWidget(dateLabel, 1, 0)
            p.addWidget(dateEdit, 1, 1)
            p.addWidget(releasesCheckBox, 2, 0)
            p.addWidget(upgradesCheckBox, 3, 0)
            p.addWidget(hitsSpinBox, 4, 0, 1, 0)
        end
```

```
      self.layout = Qt::VBoxLayout.new do |m|
         m.addWidget(packagesGroup)
         m.addSpacing(12)
         m.addWidget(startQueryButton)
         m.addStretch(1)
      end
   end
end
```

نيجى لل Config Dialog بقة

configdialog

```
require 'pages.rb'

class ConfigDialog < Qt::Dialog

   slots 'changePage(QListWidgetItem*, QListWidgetItem*)'

   def initialize()
      super
      @contentsWidget = Qt::ListWidget.new do |c|
         c.viewMode = Qt::ListView::IconMode
         c.iconSize = Qt::Size.new(96, 84)
         c.movement = Qt::ListView::Static
         c.maximumWidth = 128
         c.spacing = 12
      end

      @pagesWidget = Qt::StackedWidget.new do |p|
         p.addWidget(ConfigurationPage.new)
         p.addWidget(UpdatePage.new)
         p.addWidget(QueryPage.new)
      end

      closeButton = Qt::PushButton.new(tr("Close"))

      createIcons()
      @contentsWidget.currentRow = 0

      connect(closeButton, SIGNAL('clicked()'), self, SLOT('close()'))

      horizontalLayout = Qt::HBoxLayout.new do |h|
         h.addWidget(@contentsWidget)
         h.addWidget(@pagesWidget, 1)
      end
```

```
    buttonsLayout = Qt::HBoxLayout.new do |b|
        b.addStretch(1)
        b.addWidget(closeButton)
    end

    self.layout = Qt::VBoxLayout.new do |m|
        m.addLayout(horizontalLayout)
        m.addStretch(1)
        m.addSpacing(12)
        m.addLayout(buttonsLayout)
    end

    self.windowTitle = tr("Config Dialog")
end

def createIcons
    configButton = Qt::ListWidgetItem.new(@contentsWidget) do |c|
        c.icon = Qt::Icon.new("images/config.png")
        c.text = tr("Configuration")
        c.textAlignment = Qt::AlignHCenter
        c.flags = Qt::ItemIsSelectable | Qt::ItemIsEnabled
    end

    updateButton = Qt::ListWidgetItem.new(@contentsWidget) do |u|
        u.icon = Qt::Icon.new("images/update.png")
        u.text = tr("Update")
        u.textAlignment = Qt::AlignHCenter
        u.flags = Qt::ItemIsSelectable | Qt::ItemIsEnabled
    end

    queryButton = Qt::ListWidgetItem.new(@contentsWidget) do |q|
        q.icon = Qt::Icon.new("images/query.png")
        q.text = tr("Query")
        q.textAlignment = Qt::AlignHCenter
        q.flags = Qt::ItemIsSelectable | Qt::ItemIsEnabled
    end

    connect(@contentsWidget,
            SIGNAL('currentItemChanged(QListWidgetItem*, QListWidgetItem*)'),
            self, SLOT('changePage(QListWidgetItem*, QListWidgetItem*)'))
end

def changePage(current, previous)
    if current.nil?
        current = previous
    end
```

```
    @pagesWidget.currentIndex = @contentsWidget.row(current)
  end
end
```

تمام الموضوع بسيط هنعلق على اجزاء منه فقط

1- انشاء ال Content Widget (ال List)

```
    @contentsWidget = Qt::ListWidget.new do |c|
      c.viewMode = Qt::ListView::IconMode
      c.iconSize = Qt::Size.new(96, 84)
      c.movement = Qt::ListView::Static
      c.maximumWidth = 128
      c.spacing = 12
    end
```

حددنا فيها عدة خصائص اهمها ان ال viewMode هيكون IconMode ليتم عرض ايكونز وحددنا حجم الأيكون بإستخدام iconSize

2- عملنا ال StackedWidget object بتاعنا اللى تحدثنا عنه من قبل

```
    @pagesWidget = Qt::StackedWidget.new do |p|
      p.addWidget(ConfigurationPage.new)
      p.addWidget(UpdatePage.new)
      p.addWidget(QueryPage.new)
    end
```

وضفناله كل ال widgets (pages) اللى كتبناها قبل كدا!

3- انشاء ال button واستدعاء createIcons (هى ميثود مسئولة عن ملأ ال ListWidget الخاص بأرقام الصفحات)

```
    closeButton = Qt::PushButton.new(tr("Close"))

    createIcons()
```

4- تحديد المكان الحالى فى ال contentsWidget لأول عنصر

```
    @contentsWidget.currentRow = 0
```

ربط ال clicked signal الخاصة بال closeButton بال slot clos

```
    connect(closeButton, SIGNAL('clicked()'), self, SLOT('close()'))
```

5- إضافة ال Layouts عادى جدا ولكن عايزين نطلع على createIcons

```
  def createIcons
    configButton = Qt::ListWidgetItem.new(@contentsWidget) do |c|
      c.icon = Qt::Icon.new("images/config.png")
      c.text = tr("Configuration")
      c.textAlignment = Qt::AlignHCenter
      c.flags = Qt::ItemIsSelectable | Qt::ItemIsEnabled
    end
```

```
    #code omitted...
end
```

1-بننشئ Item فى اللل List زى ماتعلمنا بإننا نباصي ال List للمشيد الخاص ب ListWidgetItem
2- نحدد ال icon المستخدمة بإننا نباصي Icon object
3-نحدد ال text الظاهر مع ال icon بإستخدام .text=
4- نحدد المحاذاةبإستخدام textAlignment=
5- نحدد ال flags بالقابلية للإختيار والإتاحة للمستخدم بإستخدام flags=
وهكذا مع باقى العناصر....

اخيرا ننفذ الفكرة اللى قلناها فى بداية المثال

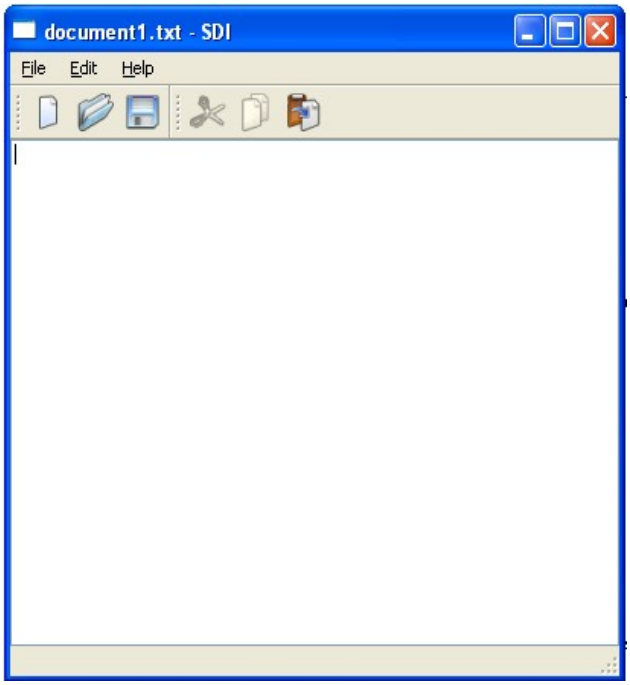
```
  connect(@contentsWidget,
        SIGNAL('currentItemChanged(QListWidgetItem*, QListWidgetItem*)'),
        self, SLOT('changePage(QListWidgetItem*, QlistWidgetItem*)'))
```

نربط ال signal القادمة من contentsWidget الخاصة بتغير الإختيار currentItemChanged ب SLOT احنا هنعرفها بإسم changePage

```
 def changePage(current, previous)
   if current.nil?
      current = previous
   end

   @pagesWidget.currentIndex = @contentsWidget.row(current)
end
```

وبس كدا :)

Simple SDI Application



هننشئ Text Editor بسيط فيه منيو وتولبار و text Edit و statusbar
اولا ال slots اللى هيدعمها ال application

```
slots   'newFile()',
        'open()',
        'save()',
        'saveAs()',
        'about()',
        'documentWasModified()'
```

newFile هتتنفذ عند محاولة انشاء ملف فارغ
open عند محاولة فتح ملف
save, save as عند الحفظ
about عند محاولة عرض about عن البرنامج
documentWasModified عند حدوث اى تعديل

```
attr_reader :curFile

@@sequenceNumber = 0
```

curFile: بيعبر عن ال file الحالى
@@sequenceNumber: عبارة عن class variable بيعبر عن رقم تلقائى ياخده الملف الجديد

```
def initialize(fileName = "")
        super()
         init()

        if fileName.empty?
                setCurrentFile("")
        else
          loadFile(fileName)
        end
end
```

init بتجهز الأبلكيشن

```
def init()
   setAttribute(Qt::WA_DeleteOnClose)

   @isUntitled = true

   @textEdit = Qt::TextEdit.new
   setCentralWidget(@textEdit)

   createActions()
   createMenus()
   createToolBars()
   createStatusBar()

   readSettings()

   connect(@textEdit.document(), SIGNAL('contentsChanged()'),
        self, SLOT('documentWasModified()'))
```

```
end
```

WA_DeleteOnClose: بتخلى Qt تحذف الويدجت عند قبول ال closeEvent (بيحدث عند محاولة قفل الويدجت)
setCenteralWidget تحديد ال ويدجت الرئيسى اللى هو التكست اديت!
@isUntitled لتتبع هل تم الحفظ اولأ
createActions هى ميثود كتبناها لإنشاء ال actions للى ممكن تضاف فى المنيوز والتولبار
createMenus لإنشاء المنيوز file, edit, about
createToolBars لإنشاء كل اقسام التولبار
createStatusBar لإنشاء ال statusBar
readSettings قراءة الإعدادات الخاصة بالبرنامج(هنتعرض ليها لاحقا)

اولا createActions لننشئ ال actions ونربطها بال slots المناسبة لها

```
def createActions()
  @newAct = Qt::Action.new(Qt::Icon.new("images/new.png"), tr("&New"), self)
  @newAct.shortcut = Qt::KeySequence.new( tr("Ctrl+N") )
  @newAct.statusTip = tr("Create a file.new")
  connect(@newAct, SIGNAL('triggered()'), self, SLOT('newFile()'))

  @openAct = Qt::Action.new(Qt::Icon.new("images/open.png"), tr("&Open..."), self)
  @openAct.shortcut = Qt::KeySequence.new( tr("Ctrl+O") )
  @openAct.statusTip = tr("Open an existing file")
  connect(@openAct, SIGNAL('triggered()'), self, SLOT('open()'))

  @saveAct = Qt::Action.new(Qt::Icon.new("images/save.png"), tr("&Save"), self)
  @saveAct.shortcut = Qt::KeySequence.new( tr("Ctrl+S") )
  @saveAct.statusTip = tr("Save the document to disk")
  connect(@saveAct, SIGNAL('triggered()'), self, SLOT('save()'))

  @saveAsAct = Qt::Action.new(tr("Save &As..."), self)
  @saveAsAct.statusTip = tr("Save the document under a name.new")
  connect(@saveAsAct, SIGNAL('triggered()'), self, SLOT('saveAs()'))

  @closeAct = Qt::Action.new(tr("&Close"), self)
  @closeAct.shortcut = Qt::KeySequence.new( tr("Ctrl+W") )
  @closeAct.statusTip = tr("Close self window")
  connect(@closeAct, SIGNAL('triggered()'), self, SLOT('close()'))

  @exitAct = Qt::Action.new(tr("E&xit"), self)
  @exitAct.shortcut = Qt::KeySequence.new( tr("Ctrl+Q") )
  @exitAct.statusTip = tr("Exit the application")
  connect(@exitAct, SIGNAL('triggered()'), $qApp, SLOT('closeAllWindows()'))

  @cutAct = Qt::Action.new(Qt::Icon.new("images/cut.png"), tr("Cu&t"), self)
  @cutAct.shortcut = Qt::KeySequence.new( tr("Ctrl+X") )
  @cutAct.setStatusTip(tr("Cut the current selection's contents to the " +
```

```
                    "clipboard"))
        connect(@cutAct, SIGNAL('triggered()'), @textEdit, SLOT('cut()'))

        @copyAct = Qt::Action.new(Qt::Icon.new("images/copy.png"), tr("&Copy"), self)
        @copyAct.shortcut = Qt::KeySequence.new( tr("Ctrl+C") )
        @copyAct.setStatusTip(tr("Copy the current selection's contents to the " +
                    "clipboard"))
        connect(@copyAct, SIGNAL('triggered()'), @textEdit, SLOT('copy()'))

        @pasteAct = Qt::Action.new(Qt::Icon.new("images/paste.png"), tr("&Paste"), self)
        @pasteAct.shortcut = Qt::KeySequence.new( tr("Ctrl+V") )
        @pasteAct.setStatusTip(tr("Paste the clipboard's contents into the current " +
                    "selection"))
        connect(@pasteAct, SIGNAL('triggered()'), @textEdit, SLOT('paste()'))

        @aboutAct = Qt::Action.new(tr("&About"), self)
        @aboutAct.statusTip = tr("Show the application's About box")
        connect(@aboutAct, SIGNAL('triggered()'), self, SLOT('about()'))

        @aboutQtAct = Qt::Action.new(tr("About &Qt"), self)
        @aboutQtAct.statusTip = tr("Show the Qt library's About box")
        connect(@aboutQtAct, SIGNAL('triggered()'), $qApp, SLOT('aboutQt()'))


        @cutAct.enabled = false
        @copyAct.enabled = false
        connect(@textEdit, SIGNAL('copyAvailable(bool)'),
            @cutAct, SLOT('setEnabled(bool)'))
        connect(@textEdit, SIGNAL('copyAvailable(bool)'),
            @copyAct, SLOT('setEnabled(bool)'))
    end
```

لاحظ ال textEdit ليه slots معرفة فيه زى cut, copy, paste وبتتهندل تلقائيا..
اخيرا نربط ال copyAvailable slot الخاصة بال textEdit بإتاحة الأكشن وإلا هيكون disabled

انشاء المنيوز

```
def createMenus()
        @fileMenu = menuBar().addMenu(tr("&File"))
        @fileMenu.addAction(@newAct)
        @fileMenu.addAction(@openAct)
        @fileMenu.addAction(@saveAct)
        @fileMenu.addAction(@saveAsAct)
        @fileMenu.addSeparator()
        @fileMenu.addAction(@closeAct)
        @fileMenu.addAction(@exitAct)

        @editMenu = menuBar().addMenu(tr("&Edit"))
```

```
    @editMenu.addAction(@cutAct)
    @editMenu.addAction(@copyAct)
    @editMenu.addAction(@pasteAct)

    menuBar().addSeparator()

    @helpMenu = menuBar().addMenu(tr("&Help"))
    @helpMenu.addAction(@aboutAct)
    @helpMenu.addAction(@aboutQtAct)
end
```

انشاء التولبارز

```
def createToolBars()
    @fileToolBar = addToolBar(tr("File"))
    @fileToolBar.addAction(@newAct)
    @fileToolBar.addAction(@openAct)
    @fileToolBar.addAction(@saveAct)

    @editToolBar = addToolBar(tr("Edit"))
    @editToolBar.addAction(@cutAct)
    @editToolBar.addAction(@copyAct)
    @editToolBar.addAction(@pasteAct)
end
```

ننشئ ال statusBar

```
def createStatusBar()
    statusBar().showMessage(tr("Ready"))
end
```

loadFile لقراءة ملف ما ووضعه على ال textEdit

```
def loadFile(fileName)
    file = Qt::File.new(fileName)
    if !file.open(Qt::File::ReadOnly | Qt::File::Text)
        Qt::MessageBox.warning(self, tr("SDI"),
                        tr("Cannot read file %s:\n%s." % [fileName, file.errorString]))
        return
    end

    inf = Qt::TextStream.new(file)
    Qt::Application.overrideCursor = Qt::Cursor.new(Qt::WaitCursor)
    @textEdit.plainText = inf.readAll
    Qt::Application::restoreOverrideCursor

    setCurrentFile(fileName)
```

```
    statusBar().showMessage(tr("File loaded"), 2000)
end
```

هنا بنفتح الفايل للقراءة فقط وبعدها نباصيه ل TextStream لإمكانيات افضل فى القراءة ولحين انتهاء قراة الملف وعرضه فى ال textEdit بنعدل ال cursor إلى wait وبعدها نستعيده
نحدد ال currentFile ونعرض رسالة فى ال statusBar

تجهيز الفايل الحالى

```
def setCurrentFile(fileName)
    @isUntitled = fileName.empty?
    if @isUntitled
        @curFile = tr("document%d.txt" % (@@sequenceNumber += 1))
    else
        @curFile = Qt::FileInfo.new(fileName).canonicalFilePath()
    end

    @textEdit.document().modified = false
    setWindowModified(false)

    setWindowTitle(tr("%s[*] - %s" % [strippedName(@curFile), tr("SDI")]))
end
```

لاحظ استخدامنا ل strippedName ودى للحصول على اسم الفايل المجرد

```
def strippedName(fullFileName)
    return Qt::FileInfo.new(fullFileName).fileName()
end
```

لحفظ فايل معين

```
def saveFile(fileName)
    file = Qt::File.new(fileName)
    if !file.open(Qt::File::WriteOnly | Qt::File::Text)
        Qt::MessageBox.warning(self, tr("SDI"),
                        tr("Cannot write file %s:\n%s." % [fileName, file.errorString]))
        return
    end

    outf = Qt::TextStream.new(file)
    Qt::Application.overrideCursor = Qt::Cursor.new(Qt::WaitCursor)
    outf << @textEdit.toPlainText
          outf.flush
    Qt::Application.restoreOverrideCursor

    setCurrentFile(fileName)
    statusBar().showMessage(tr("File saved"), 2000)
end
```

نفس السابق ولكن بطريقة عكسية فتح للكتابة ونباصيها ل TextStream ونكتب جواها للفايل

انشاء ملف جديد ال SLOT newFile

```
def newFile()
```

```
   other = MainWindow.new
   other.move(x() + 40, y() + 40)
   other.show()
end
```

فتح ملف SLOT open

```
def open()
   fileName = Qt::FileDialog.getOpenFileName(self)
   if !fileName.nil?
      existing = findMainWindow(fileName)
      if !existing.nil?
         existing.show()
         existing.raise()
         existing.activateWindow()
         return
      end

      if @isUntitled && @textEdit.document().empty? &&
            !isWindowModified()
         loadFile(fileName)
      else
         other = MainWindow.new(fileName)
         if other.isUntitled
            other.dispose
            return
         end
         other.move(x() + 40, y() + 40)
         other.show()
      end
   end
end
```

لاحظ استخدام ال findMainWindow فى الميثود السابقة

```
def findMainWindow(fileName)
   canonicalFilePath = Qt::FileInfo.new(fileName).canonicalFilePath()

   $qApp.topLevelWidgets.each do |widget|
      if widget.kind_of?(MainWindow) && widget.curFile == canonicalFilePath
         return mainWin
                  end
   end
   return nil
end
```

save, saveAs

بيتبعو نفس الستراتيجى ولكن بيختلفو عند هل الملف اول مرة يتم حفظه او لأ

```
def save()
```

```
    if @isUntitled
        return saveAs()
    else
        return saveFile(@curFile)
    end
end

def saveAs()
    fileName = Qt::FileDialog.getSaveFileName(self, tr("Save As"),
                                              @curFile)
    if fileName.nil?
        return false
            end

    return saveFile(fileName)
end
```

SIGNALS/SLOTS

Label

Edit

ComboBox

وبس کدا :)

1-RAD – QtDesigner

2-QtRQamoos