



# BlackAdder Manual

## 1 Introduction

BlackAdder is designed to allow you to rapidly build professional applications using the popular Object Oriented Python scripting language. A key component of BlackAdder is PyQt, a GUI library that allows you to rapidly create cross platform applications (Linux and Windows currently). PyQt is a set of Python bindings for the popular Qt user interface library, and includes the “qt” module (standard set of Qt widgets), “qtcanvas” - a module that provides a highly optimized 2D graphic widget, “qtnetwork” - which allows you to quickly create client/server implementations, “qtable” - a module for quickly creating big tables and operations with them, “qtxml” - XML rendering engine that provides a well-formed XML parser using the DOM Level 2 (Document Object Model).

We've included our powerful PyQtDoc application, which allows you to quickly and easily navigate through hundreds of classes and functions. BlackAdder supports syntax highlighting for a variety of languages, most importantly Python. With the new project wizards you can rapidly generate a skeleton of your GUI, in addition we've included “Qt Designer” as a Form Editor for BlackAdder. Qt Designer is a high level tool for designing user interfaces with the PyQt library. The Form Editor will help you build a user interface with layout tools that allow you to move and scale your widgets (controls in Windows terminology) automatically at runtime. The “PyUic” convertor allows you to easily convert files with form interfaces to Python code with no effort on your part. With BlackAdder you can easily subclass Qt and your custom designed classes. BlackAdder is a powerful editor and IDE with many features that will allow you to build cross platforms applications faster than you ever thought possible. Detailed information about the product is in the following pages.

## 2 Installation.

### 2.1 Linux

Requirements:

On Linux you will need a working X11 system, this is typically installed by default with any system using a windowing interface such as KDE or Gnome, but note that neither of these desktops are required. All the libraries required to run BlackAdder are included in thekompany-support package which are distribution-independent, but you will need to install the version that best applies to your system. You will need to have Python version 2.2 or later installed as well, Python is not supplied with BlackAdder but is freely available from [www.python.org](http://www.python.org).

For Linux you need to download and install these packages:

**thekompany-support**  
**BlackAdder**  
**PyQt**

These packages have the following naming convention.

packagename-version-builddate.rpm or  
packagename-version-builddate.tar.gz

For example BlackAdder-1.0-030828.rpm illustrates that it was built on August 28, 2003.

The PyQt packages will have a suffix such as py22-ucs2, py22-ucs4, py23-ucs2, py23-ucs4. It means that the package can be used for Python2.2 or Python2.3. Python can be compiled in 2 variants: with unicode support ucs2 (all text symbols have 2 bytes) or with unicode support ucs4 (all text symbols have 4 bytes – the variant used in RedHat 9.0).

You can check how Python was compiled with ucs2 or ucs4 in this file:

```
/usr/include/python/pyconfig.h:
```

```
#define Py_UNICODE_SIZE 2  
or  
#define Py_UNICODE_SIZE 4
```

About installing PyQt from tarballs:

Unpack it and run `./install` script as root user.

It will ask you for PREFIX where Python was installed. (Typically it is just `/usr`, but if you built Python manually, then it could be something like `/usr/local`). Enter your path or press enter if it is `/usr`.

For installing rpms you need to use the rpm utility from your linux distribution. Note that PyQt from thekompany can conflict with any installed PyQt from your distribution (Free Edition), so you need to remove it before installing the BlackAdder version. Also make sure to remove all previous packages of BlackAdder, thekompany-support and PyQt. If you have purchased PyQtDoc perviously you need to remove the package because the application is already included in BlackAdder. For example:

```
# rpm -i thekompany-support-1.2-030828.rpm BlackAdder-1.0-030828.rpm PyQt-3.7-030828.suse82.rpm
```

To install the tar.gz package you need to unpack the contents to temporary directory with the following command:

```
# tar -zxf packagename-version-builddate.tar.gz
```

Go to the directory just created and run the install script as root user:

```
# cd packagename
```

```
# ./install.sh
```

## 2.2 Windows

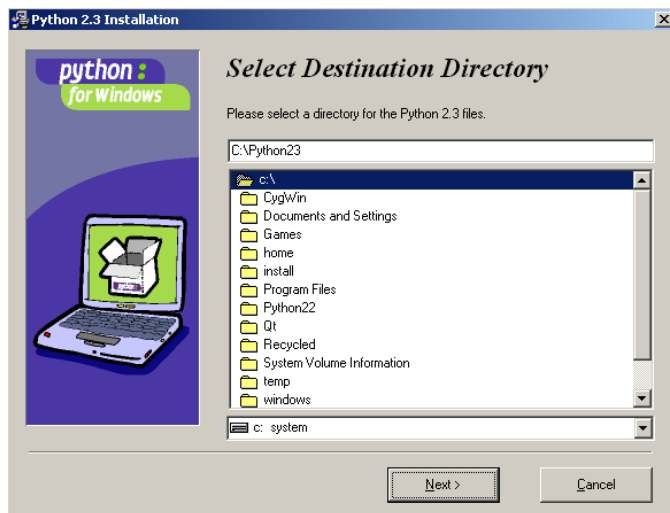
### Installation of BlackAdder 1.0 for Windows 9x/Me/NT/2000/XP

To install BlackAdder you need to install Python 2.x and PyQt 3.7.

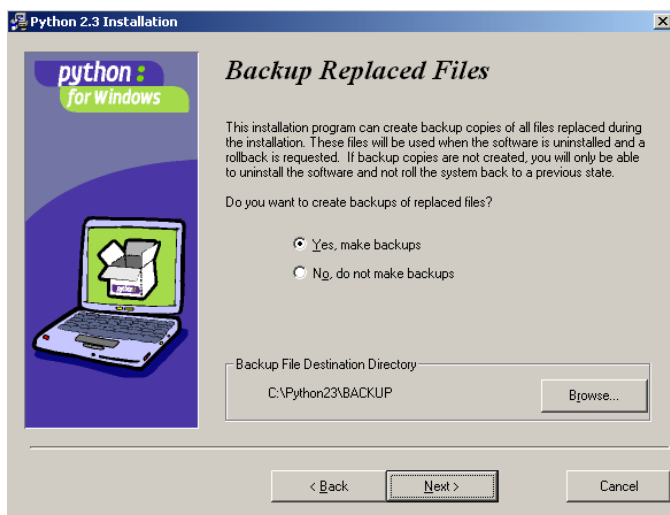
#### 1 Install Python 2.x

1.1 Download Python 2.x from <http://www.Python.org> and run the Python-2.x.exe

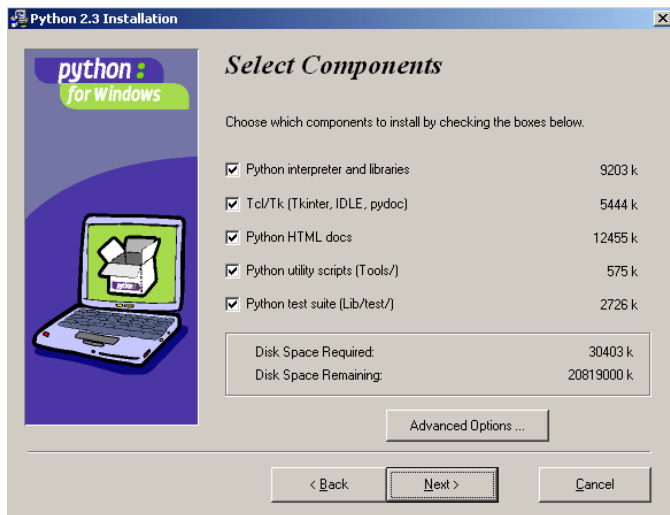
1.2 Choose a directory where you want install Python 2.x , then click “Next >”



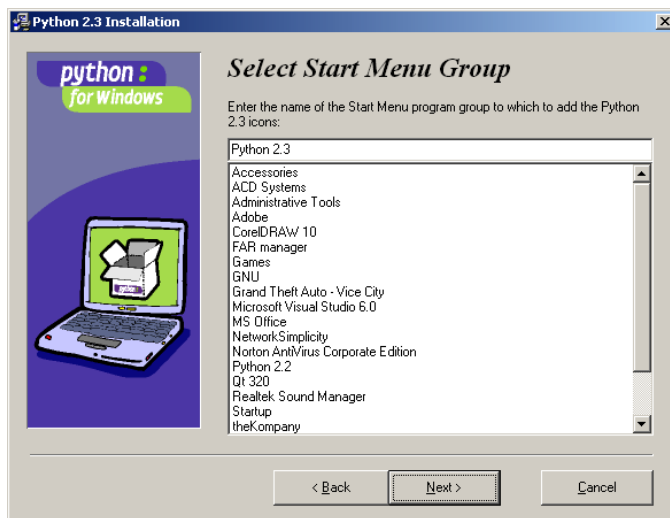
1.2 If you want to create backups file, select this option, and click “Next >”



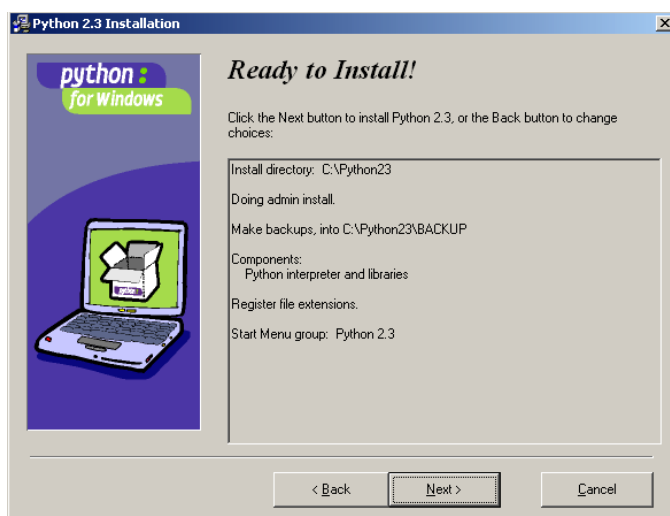
1.3 Select the Python 2.x components and click “Next >”  
Make sure that Python interpreter and libraries are selected



1.4 Select start menu group for Python 2.x and click “Next >”



1.5 You are now ready to Install Python 2.x. Click “Next >”



1.6 Wait while Python 2.x installs its own components, then click “Finish”



## 2 Install PyQt 3.7

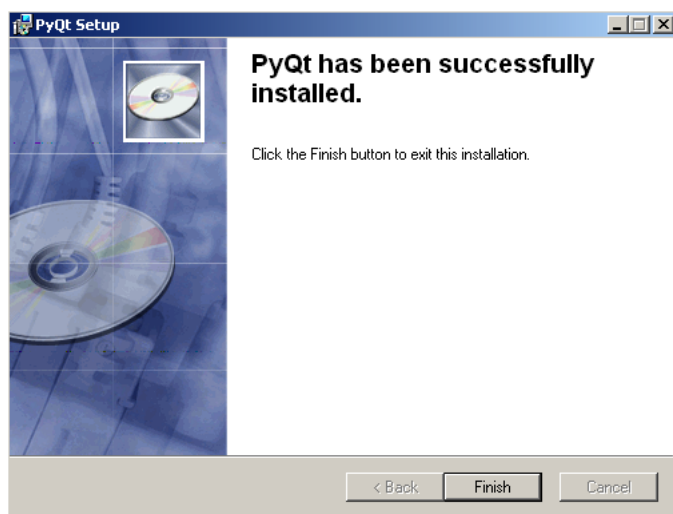
2.1 Uninstall any previous version of PyQt and make sure you have Python installed.

2.2 Download from your download basket and run PyQt-3.7-YYMMDD-Python2.x.exe

2.3 Click “Next >”



2.4 Wait while PyQt 3.7 installs its own components, then click “Finish”.

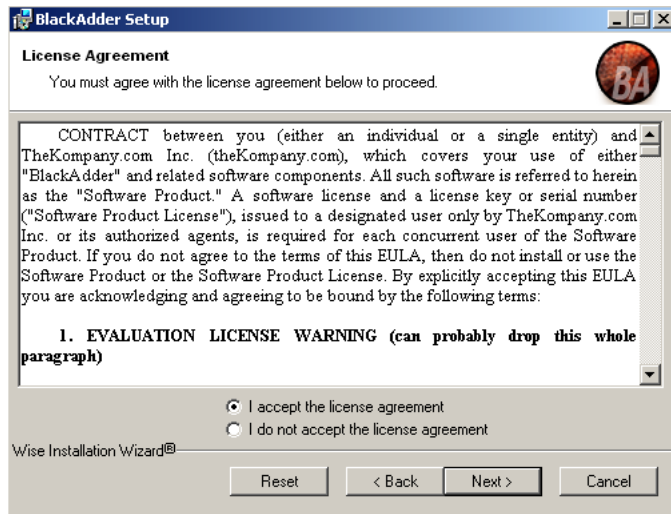


### 3 Install BlackAdder 1.0

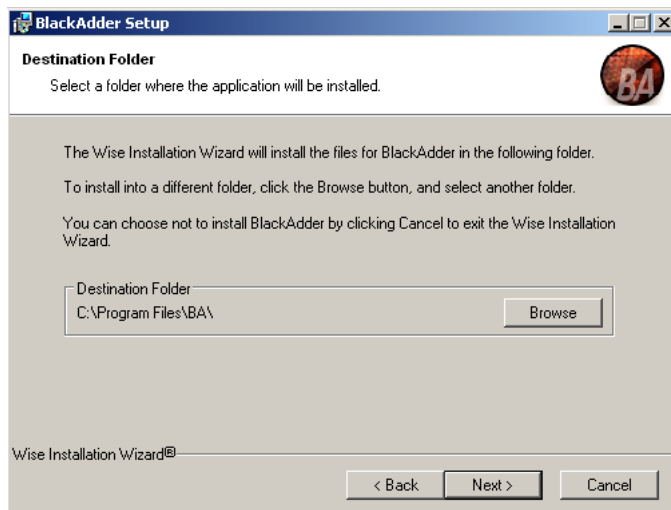
3.1 Download from download basket and run BlackAdder-1.0-YYMMDD.exe

3.2 Click “Next >”

3.3 Read license, and if you agree with the license agreement select “I accept the license agreement”, then click “Next >”

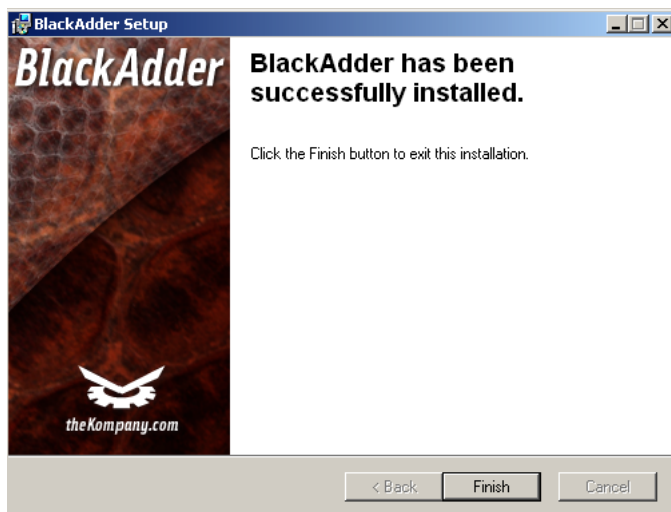


3.4 Choose a directory where you want to install BlackAdder 1.0, then click “Next >”



3.5 Click “Next >” to begin installation

3.6 Click “Finish”



## 2.3 Starting BlackAdder

To run BlackAdder on Linux you just need to run “ba” in a console:

```
# ba
```

```
or
```

```
# ba myfile.py
```

if you want to start BlackAdder with a specific file, in this case, myfile.py.

On Windows you need just run the shortcut:

**Start->Programs->theKompany->BlackAdder->BlackAdder**



### 3 Quick start guide.

#### 3.1 Introduction

We are going to create a small (and relatively useless) application to demonstrate how the tools in BlackAdder make developing professional applications in PyQt easy. Here is the process we will be following:

1. Create a new project
2. Add a form to the project
3. Place controls on the form
4. Turn the form design into a python class
5. Derive from the class to create an implementation
6. Add code to our derived class to make our application do something
7. Create a main.py to start our app
8. Run our new application

#### 3.2 Getting Started

Start the process by launching BlackAdder (BA), on Windows you do this in the normal manner - clicking on the icon on your desktop. On a Linux based os, you type ba at a command prompt in X-Windows.

#### 3.3 Creating a new project

Once BA has launched, choose menu Project -> New Project. This will launch the New Project dialog, which allows you to create a new project from scratch. In the name field, type "myfirstapp". Notice that BA will fill in the other two fields for you. Click "Next". In the next step, select the "Empty" template, click "Finish".

#### 3.4 Creating a form

Now you have been presented with a screen displaying an empty project. Choose menu item Project->Add Form. This will display the New Form dialog which asks you to choose a form template. Choose the "Dialog" template, and type "MyFormBase.ui" in the File Name field at the top of the dialog. Then click "Ok". In the project view on the left-hand side of BA, you will see your new form. Double-click on the form in the project view. This will launch BA's customized version of QT's designer.

#### 3.5 Finishing your form

You should now have a blank form open in Designer. From the toolbox on the left, drag three controls onto the form: a PushButton, a LineEdit, and a TextLabel. Arrange them any way you would like, and then select each one and make the following property changes on the property editor located on the right-hand side of the screen:

Control	Property	New Value
Form	Name	MyFormBase
	Caption	My First PyQt Application
PushButton	Text	Talk
	Name	talk
LineEdit	Name	edit
TextLabel	Text	"" (Empty String)
	Name	output

Once these changes are complete, close out of Designer, choosing "Yes" when asked if you

want to save your changes.

### 3.6 Compiling your form

You will now switch back to the main BA window. In the project window on left-hand side of the screen, right click on the form you created and choose "Generate py with pyuic". This will take the xml file created by Designer, and turn it into executable code. You won't modify this code however, because when you make changes to your form, this code will be overwritten. Instead, you will want to create a new python class that inherits from this new one that BA has created for you, which you can then add your implementation code to. We will cover this in the next step.

### 3.7 Creating a form implementation

To create a form implementation class, right-click on the "MyFormBase" dialog in the project view, and choose "inherits". You will be presented with the New Class dialog. This dialog is allowing you to derive from an existing class, in this case, the python implementation of our form. It also allows you to override methods in the base classes that your class derives from. In this case, if you wanted to re-implement methods in QDialog or QWidget, you could easily do that. For now, change the text in the "Class" field to "MyForm", and uncheck the "\_tr()" option in the Reimplement view, and then click "Ok". You should see your new class appear in the Project view.

### 3.8 Making it do something

Now we need to write some code. Open up our implementation by double-clicking on the "MyForm" class in the project view. This will open up the class in an editor window. You will notice that there is an "\_\_init\_\_" method already defined. At the end of that method add the following code:

```
connect(self.talk, SIGNAL(clicked),SLOT(slotTalk()))
```

So that the entire "\_\_init\_\_" method looks like this:

```
def __init__(self, *args):
    apply(MyFormBase.__init__,(self,) + args)
    self.connect(self.talk, SIGNAL("clicked()"),self.slotTalk)
```

This connects the clicked() signal of the button on our form to a slot we will add in a moment. **NOTE: Depending on your tab settings, you may have to tweak the indentation of the code that existed already.** Next, create a new method in this class with the following code.

```
def slotTalk(self):
    self.output.setText(self.edit.text())
```

### 3.9 Finishing up

Now we need to put the finishing touches on our application. Start by right-clicking on the project in the Project view (this is the top-level item, and looks like four colored blocks) and selecting "Add main.py". A dialog will appear asking you to choose your main form; you will want to select our implementation, called "MyForm" and choose Ok.

### 3.10 Testing our creation

Now that we're done, let's test what we have made. Click the "Run" button on the toolbar. When asked to choose the target, pick "main.py". If all has gone well, you will see the form that we created in designer. If not, review the output to determine the cause of the error (hint - check for indentation, or a problem with the name of a class). Enter some text in the line edit, and then click "Talk". You should see your text displayed in the label on your form.

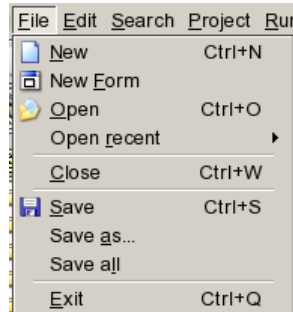
### **3.11 Summary**

You've now written a complete application in PyQt. Granted, this is very simple but it does demonstrate many of the concepts needed to build your own applications. More thorough examples are provided with BA in the examples directory.

## 4 Overview

### 4.1 Menu

BlackAdder has a main menubar and popups menus on context items. The main menu has standard items such as File, Edit, Search, Project, Run, Tool, Options, Window and Help. All these menus can be accessed by Alt+first\_symbol\_of\_menu, for example you can quickly access the “File” menu with Alt+F keyboard shortcut.



“File” menu contains actions for standard operations as

“New” – creates a new empty file.

“New Form” - creates a new form from template and edits it with the Form Editor.

“Open” - opens an existing file.

“Open recent” - small submenu for opening recent files.

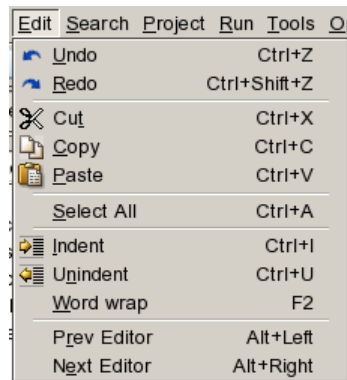
“Close” - close current document.

“Save” - save currently open document.

“Save as” - save document under new selected name.

“Save all” - saving all currently opened documents.

“Exit” - just quit application. If you have modified documents BlackAdder will prompt you to save them.



“Edit” menu contains editor actions.

“Undo” - revert the most recent undo operation in editor.

“Redo” - revert the most recent editing actions in editor.

“Cut” - cut the selected text and move it to clipboard.

“Copy” - copy selected text to clipboard.

“Paste” - paste previously copied or cut clipboard contents.

“Select All” - select entire text of current document.

“Indent” - use this to indent a selected block of text.

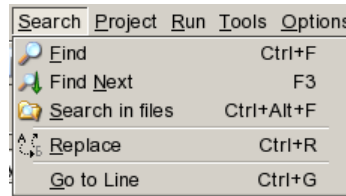
“Unindent” - use this to unindent a selected block of text.

You can configure whether tabs should be honored and used or replaced with spaces, in configuration dialog.

“Word wrap” - use this command to wrap all lines of the current document which are no longer the width of current view.

“Prev editor” - switch to next editor in list of opened files.

“Next editor” - switch to previous editor in list of opened files.



“Search” menu contains actions for searching in text, searching in files, goto to line.

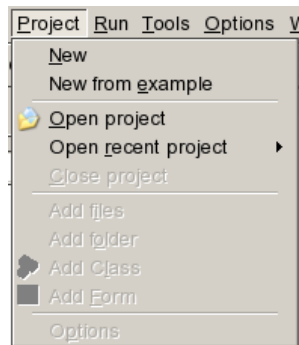
“Find” - open search dialog for searching first occurrence of text.

“Find Next” - searching next occurrence of search text.

“Search in files” - open dialog for searching in files. Output of results of searching will be in message panel – you can just clicking on results for quickly jump on file. See more info in “Dialogs” sections.

“Replace” - replace dialog.

“Go to line” - move the cursor to a specific line number.



“Project” menu contains actions for working with the files in a project.

“New” - creates a new project. Opens the new project dialog where you can select or create a directory where the project will be placed; project name and project file name (XML file where will be stored all info about your project). After these settings you can select a type for your project, and start a wizard to generate Python files if they are available.

“New from example” - opens dialog where you can select existing (predefined) example project. All files of the project will be copied to directory you will set, so you can start very quickly. Note that these predefined projects are stored in /usr/local/ba/examples/ (Linux) or C:\Program Files\BA\examples\ (Windows). Each directory contains Python files, “pro” file (XML project) and text file with description of the project (the text you are seeing “New from example” dialog)

“Open project” - open project dialog.

“Open recent project” - small popup menu with list of recently opened files.

“Close project” - close currently opened project. Note that the operation does not close your opened files.

“Add files” - adds files to current project. It opens the file select dialog. You can select multiple files by dragging the mouse or just with Shift, or Ctrl.

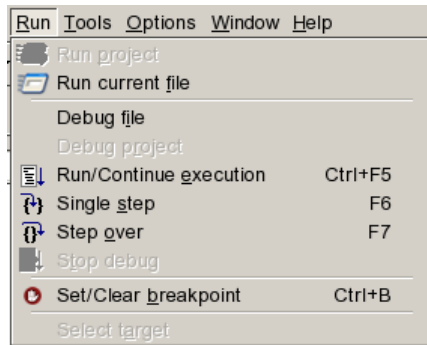
“Add folder” - adds a new folder to project. If you type in a dialog name of a non-existing folder it will be created. You can type a name with directory separators and the entire chain of sub-directories will be created.

“Add class” - opens “new class” dialog. Here you can subclass an existing qt class or one you previously created. You can also just type in the dialog select virtual functions you want to reimplement. The dialog has a combo box where you can specify a new file name or just select existing files where the class will be added. As super class you can select existing Qt class or yours.

“Add form” - adds new form to your project. It opens dialog with templates you can select. At the top you need to specify file name of your new form. The form will be added at the top of projects tree. If you want to add a new form to some folder in the project you need to use the right-button menu on project folder in project panel. Just after clicking “Ok” button it will opens Form Editor where you can advance the form template. Note that

these templates are stored in /usr/local/ba/templates/ (Linux) or C:\Program Files\BA\templates\ (Windows). Each template has three files – png with a snapshot image of template, ui form with template form and text file with descriptions (that you are seeing in “new form” dialog).

“Options” - opens projects options. Now options dialog contains project name and target (file that will be executed in Python interpreter when you will run/debug project).



“Run” menu contains actions to run Python scripts/projects or debug them.

“Run project” - execute current project. Output of application will be outputted in Message panel.

“Run current file” - execute current editor file in Python interpreter.

“Debug file” - start debug session with file opened in editor.

“Debug project” - start debug session of current project.

“Run/Continue execution” - start execution or continue if debug session was stopped on some breakpoint.

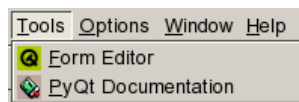
“Single step” - execute one line of Python code. It will enter in body of function if Python code of the function is in current project and can be opened.

“Step over” - execute one line of Python code but without entering in called functions.

“Stop debug” - stop debugging session.

“Set/Clear breakpoint” - set or remove breakpoint on current editor line. Note that you can use shortcut Ctrl-B do it quickly.

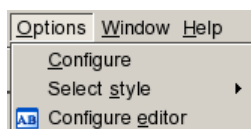
“Select target” - selecting target – file that will be executed in Python interpreter when you run/debug a project.



“Tool” menu contains actions for execute tools distributed with BlackAdder.

“Form Editor” - run Form Editor (Qt Designer)

“PyQt Documentation” - run PyQt documentation system (PyQtDoc application)

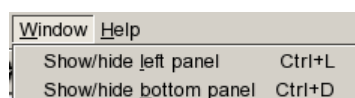


“Options” menu contains actions for configuring editors and IDE.

“Configure” - main IDE options as selecting directory where your projects located, path to Python interpreter and pyuic converter.

“Select style” - sub menu for selecting style of application widgets.

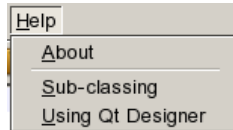
“Configure editor” - editor options as font, indent and etc.



“Window” menu contains actions for show/hide BA panels.

“Show/hide left panel” - show/hide left panel with project, fileexplorer. A good shortcut is to

use Ctrl-L when you need more space for editing.  
“Show/hide bottom panel” - show/hide bottom panel with messages and Python console.  
Shortcut Ctrl-D.



“Help” menu provides information about the IDE and help for working with BA.

“About” - dialog that provide info version of IDE, copyright and credits.

“Sub-classing” - opens help browser with instructions about subclassing in BlackAdder.  
Please see section below if you want to read more about subclassing.

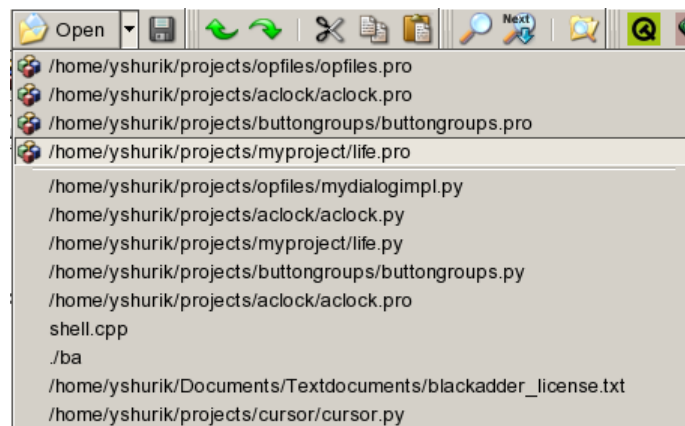
“Using Qt Designer” - open help browser with instruction about using Form Editor in BlackAdder. Please see section below if you want to read more about using Form Editor.

## 4.2 Toolbars

First look at tool bars:



You can see that the tool bar has 5 parts. The first part pertains to operations with files such as new, new project, new form, open file or project and save. Note one useful thing – you can have quick access to recent files and projects by pressing the right part (down arrow) of Open tool button. See screen shot below:



The next part of tool bar is standard editor commands is undo/redo and cut/copy/paste.

After that, you can see a tool button for search operations – find, find next and search in files dialog.

Two buttons to run the Form Editor (Qt Designer) and PyQt Documentation system create tools toolbar.

The last group of buttons is the debugger operations and run. It contains buttons to run the current project or file if a project is not open, continue operation, two buttons for single step and step over, button to stop a debug session and set/clear breakpoint.

## 4.3 Editor

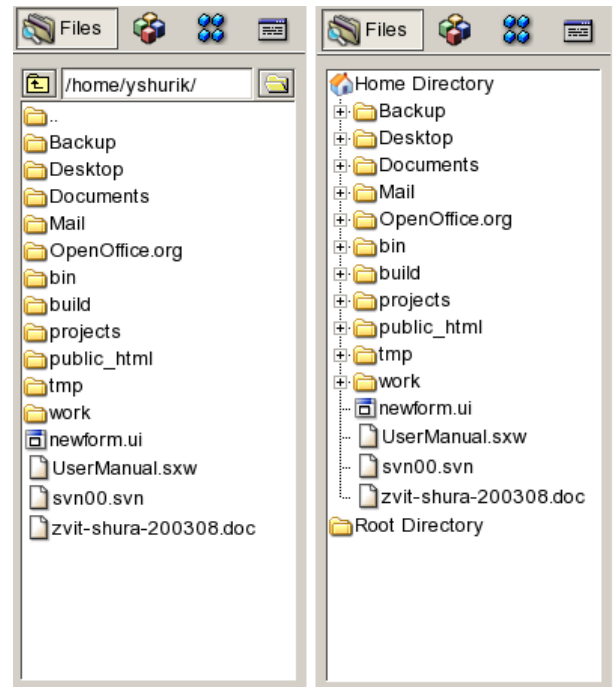
The Editor that is used in BlackAdder is another product developed at theKompany and is used in a variety of our products. The editor has standard features for editing as well as a powerful highlighting system which allow you to easily navigate in your code, on the fly checking and repair of errors.

Note: We are planning to include the Vim editor in BlackAdder as plugin. Then you will be able

to use all the power of vim editing if you have already experience working with Vim, this is optional and expected to be in version 1.1.

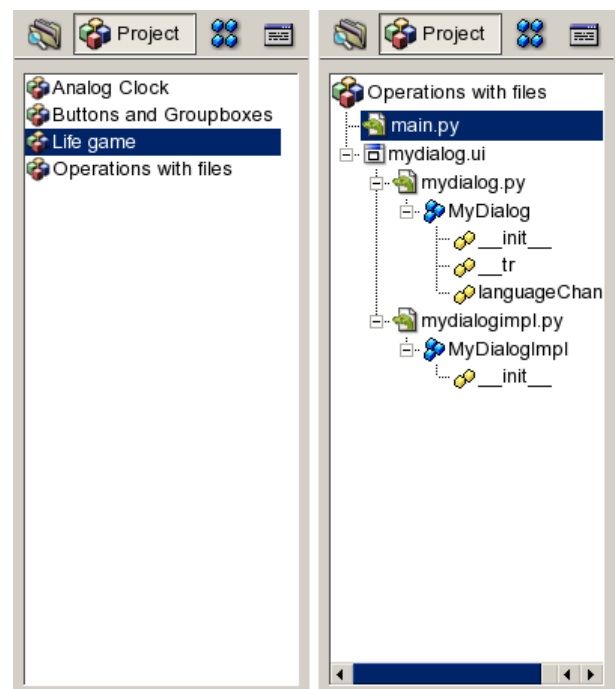
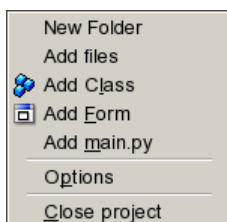
#### 4.4 File Explorer

BlackAdder has two File views. They are located on the panel named Files. The first is a list style interface, you can open a directory or select a file by single clicking on it. Here is an example: By right mouse clicking you can switch to "tree mode", which is the second interface, to do copy, paste, delete; insert file or directory in current project and reload current directory if there are new elements. Our next file style is 'Tree Mode', this is where you can expand or collapse directory structures by clicking on the icon next to the name, or double clicking on the name. While this is useful for certain types of operations, it can create a cluttered view as well.



#### 4.5 Project window

When you start BlackAdder and switch to the Project panel you can see a list of names of recently opened projects. You can open one of them by double clicking. If you want to remove a bookmark, you can do it with a right button click and remove it from the list of recent projects. After opening a project you will see a tree where on the top level you see the name of a project (root) and a tree-like list of files and folders from it. Right-button click on the root of a project to allow to do the following:



So you can add folders or files to a project directory (note: that files will be copied to the project directory if they are in another place), create new class or subclass from an existing one, add form and add main.py – similar to function main() in C – the operation will ask you for Main Window form and then generate small file with code similar to the following:

```
#!/usr/bin/env Python

import sys
from qt import *
from aclock import AnalogClock

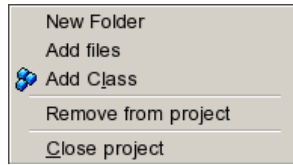
a = QApplication(sys.argv)

w = AnalogClock()
```



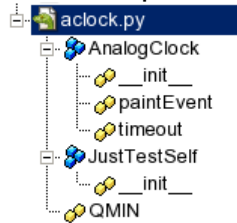
```
w.show()
a.connect(a, SIGNAL('lastWindowClosed()'), a, SLOT('quit()'))
a.exec_loop()
```

This is standard Qt code that allow you to initialize the PyQt library, and run the main form of your project. You can all call the project options dialog here. Right clicking on a folder will show you next menu:



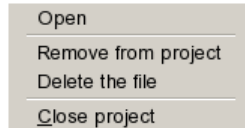
It has the same operations as a project item like new folder, add files and add project. Also it has the action “remove from project” (note: that it will not delete the directory from your file system). Note that when you create a new folder in a project then the file “\_\_init\_\_.py” will be automatically generated – then you can use “import” for files from the directory and add some initialization to the “\_\_init\_\_.py” file.

If you click the Python file you can see that “+” appears to the left of the icons – this item can be expanded to see the structure of the Python file (of course if the file doesn't contain functions and classes you will not see “+”). Here is an example:



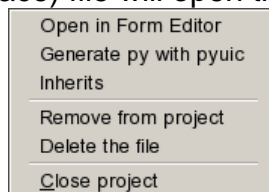
When clicking on classes or funcions the file will be opened in the editor and the cursor will go to the definition of the class or function – this is a handy feature for rapidly navigating a file.

Right mouse clicking on a Python file will show the next menu:



The menu contains operations for opening a file in the editor, remove the file from a project, and to delete it from file system.

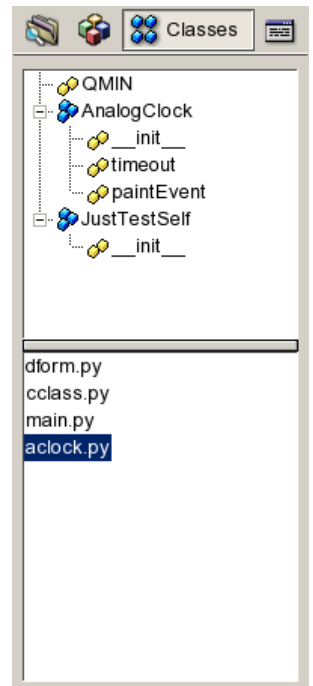
Right clicking on the UI (User Interface) file will open the following menu:



Here you can open a file in the Form Editor (Qt Designer), generate a Python file with the pyuic converter and subclass the form (form class of user interface) for enhancing the implementation of the form. Refer to this manual for using the Form Editor (Qt Designer) chapter “**The subclassing approach**” for more info about enhancing forms.

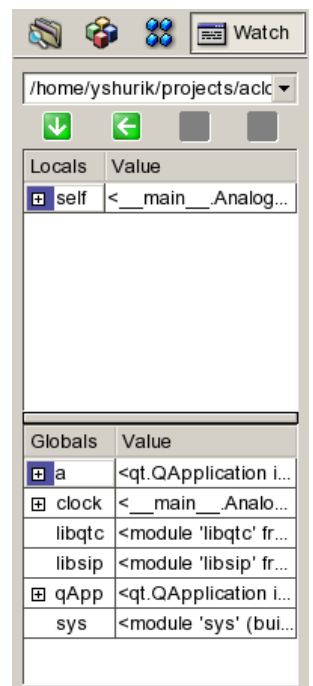
## 4.6 Classes window (Python parser)

The window shows contents of currently editing file. It is list of classes and functions. When you will click on class name or function name then cursor in editor will jump to the line of code. Below you can see list of opened files. Clicking to filename will switch editors tab to the file and you will continue editing with the file. Contents of file is updating with each 5 second, so the window follow to your changes in editor.



## 4.7 Debugger window

In debug window you can see values of variables in python debug session. There are two windows – for local variables and for global variables. Also you can navigate through python debugger frames with buttons on top (stack of calls). Some variables can be opened to values inside.



## 4.8 Messages window



Message window allow you see output of your python scripts and errors of python interpreter. Small button with icon on bottom right corner allow you clean the window.

## 4.9 Python console

```
Python 2.2.2 (#1, Mar 17 2003, 15:17:58)
[GCC 3.3 20030226 (prerelease) (SuSE Linux)] on linux2
Type "copyright", "credits" or "license" for more information.
>>> print "Hello world"
Hello world
>>>
```

The window give you access to python interpreter. Here you can experiment with small parts of code for check your thoughts. Small button with icon on bottom right corner allow you clean the window.

## 4.10 Key bindings

Here is a list of keyboard shortcuts for the menu options:

Alt-F	= "File"
Alt-FN	= "New"
Alt-FF	= "New Form"
Alt-FO	= "Open"
Alt-FR	= "Open recent"
Alt-FC	= "Close"
Alt-FS	= "Save"
Alt-FA	= "Save as"
Alt-FL	= "Save all"
Alt-FE	= "Exit"
Alt-E	= "Edit"
Alt-EU	= "Undo"
Alt-ER	= "Redo"
Alt-ET	= "Cut"
Alt-EC	= "Copy"
Alt-EP	= "Paste"
Alt-ES	= "Select All"
Alt-EI	= "Indent"
Alt-EN	= "Unindent"
Alt-EW	= "Word wrap"
Alt-ER	= "Prev editor"
Alt-EE	= "Next editor"
Alt-S	= "Search"
Alt-SF	= "Find"
Alt-SN	= "Find Next"
Alt-SS	= "Search in files"
Alt-SR	= "Replace"
Alt-SG	= "Go to line"
Alt-P	= "Project"
Alt-PN	= "New"
Alt-PE	= "New from example"
Alt-PO	= "Open project"
Alt-PR	= "Open recent project"
Alt-PC	= "Close project"
Alt-PI	= "Add files"
Alt-PO	= "Add folder"
Alt-PL	= "Add class"
Alt-PF	= "Add form"
Alt-PP	= "Options"
Alt-R	= "Run"
Alt-RP	= "Run project"
Alt-RF	= "Run current file"
Alt-RI	= "Debug file"
Alt-RR	= "Debug project"
Alt-RE	= "Run/Continue execution"
Alt-RS	= "Single step"

Alt-RO	= "Step over"
Alt-RT	= "Stop debug"
Alt-RB	= "Set/Clear breakpoint"
Alt-RA	= "Select target"
Alt-T	= "Tool"
Alt-TF	= "Form Editor"
Alt-TP	= "PyQt Documentation"
Alt-O	= "Options"
Alt-OC	= "Configure"
Alt-OS	= "Select style"
Alt-OE	= "Configure editor"
Alt-W	= "Window"
Alt-WL	= "Show/hide left panel"
Alt-WB	= "Show/hide bottom panel"
Alt-H	= "Help"
Alt-HA	= "About"
Alt-HS	= "Sub-classing"
Alt-HU	= "Using Qt Designer"

The following key bindings are also available:

*file operations:*

Ctrl-N	= New
Ctrl-O	= Open
Ctrl-W	= Close
Ctrl-S	= Save
Ctrl-Q	= Quit

*editor operations:*

Ctrl-Z	= Undo
Ctrl-Shift-Z	= Redo
Ctrl-X	= Cut
Ctrl-C	= Copy
Ctrl-V	= Paste
Ctrl-A	= Select all
Ctrl-I	= Indent
Ctrl-U	= Unindent
F2	= Word wrap
Alt-Left	= switch to previous editor
Alt-Right	= switch to next editor
Ctrl-F	= search text
F3 =	Find next
Ctrl-R	= Replace
Ctrl-G	= Go to line

*run/debug operations:*

F5	= run current project or current file if no opened project
Ctrl-F5	= run/continue execution
F6	= Single step
F7	= Step over
Ctrl-B	= Set/Clear breakpoint

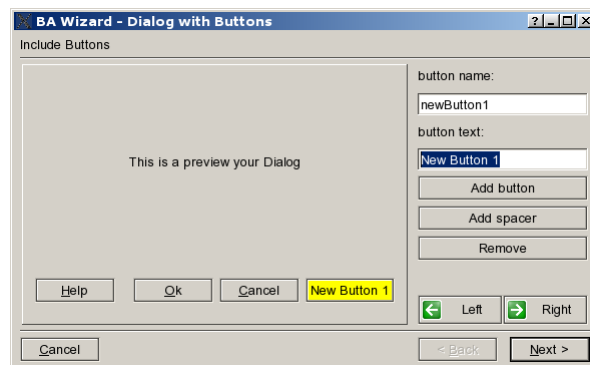
*workspace operations:*

Ctrl-L	= Show/Hide left panel
Ctrl-D	= Show/Hide bottom panel

## 4.11 Wizards

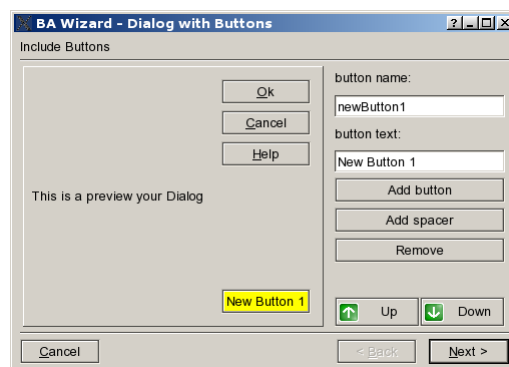
### 4.11.1 Dialog application with buttons (at bottom)

The wizard allow you quickly create a dialog with bottom line of some buttons. All just you need it is enter text for buttons and order them on bottom line. Use “Left” and “Right” to move buttons. Also you can adds spacer items for splitting groups of buttons. Next page will allow you specify name of class and filename of form.



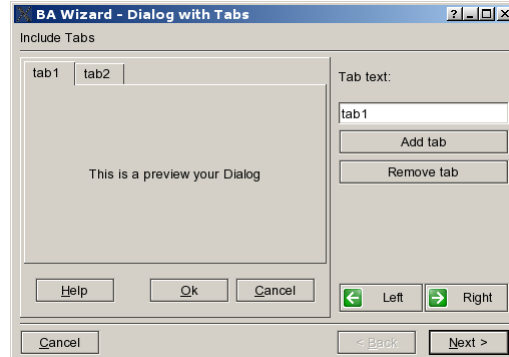
### 4.11.2 Dialog application with buttons (at right)

The wizard allow you quickly create a dialog with right line of some buttons. All just you need it is enter text for buttons and order them on bottom line. Use “Up” and “Down” to move buttons. Also you can adds spacer items for splitting groups of buttons. Next page will allow you specify name of class and filename of form.



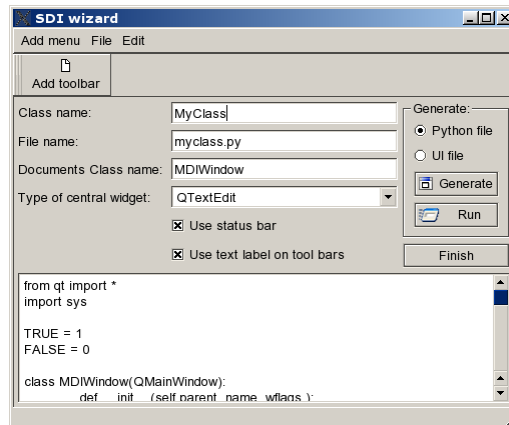
### 4.11.3 Dialog application with tabs

The dialog helps you create application based on tabbed dialog (for example some small dialog for configurable section) Here you can simply adds new tabs and move them for order what you want. Next wizard page allow you specify name of class and filename.



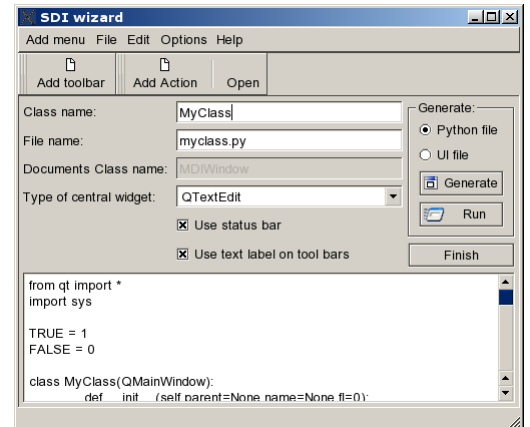
### 4.11.4 Multi document application

The wizard help you quickly create skeleton of main window of your application. Here you can quickly adds menu items, separators, tool bars and tool buttons for tool bars. When you adding some new active element it shows dialog for creating new action where you can specify action text, variable name (for QAction element), option is the action toggled, action pixmap, hotkey and connect it to slot. Almost all elements support auto completion from entering name of action. You will need specify class of Main window, filename for the class, document class name and sub class it from existing class.



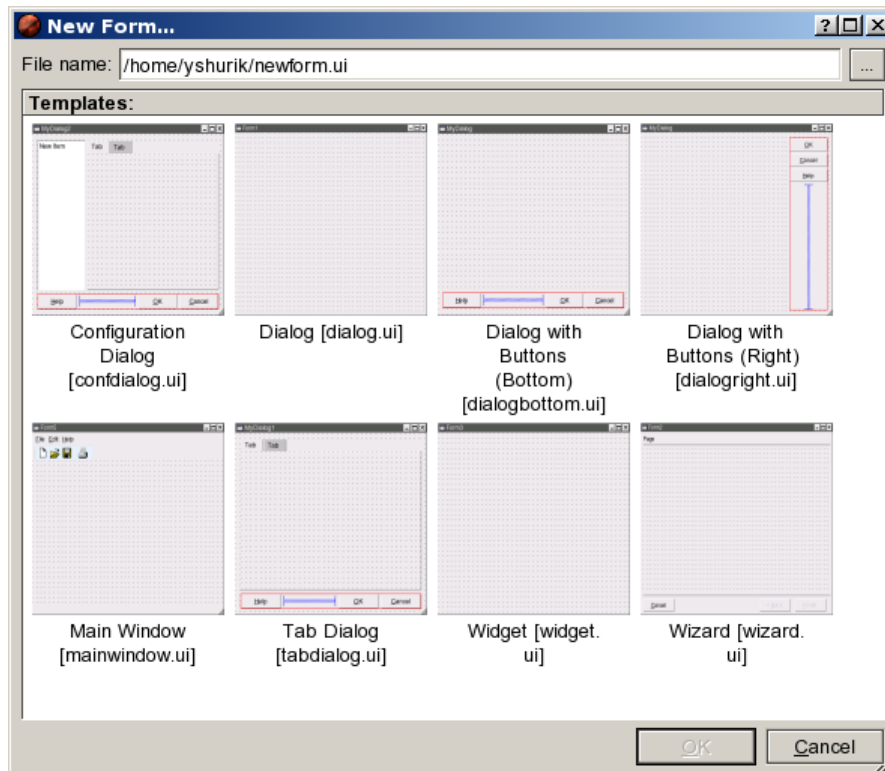
## 4.11.5 Single document application

The wizard is like same as wizard for multi document wizard. It help you quickly create skeleton of main window of your application. Here you can quickly adds menu items, separators, tool bars and tool buttons for tool bars. When you adding some new active element it shows dialog for creating new action where you can specify action text, variable name (for QAction element), option is the action toggled, action pixmap, hotkey and connect it to slot. Almost all elements support auto completion from entering name of action. You will need specify class of Main window, filename for the class.



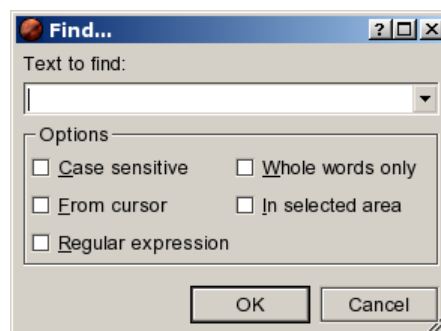
## 4.12 Dialogs

### 4.12.1 New Form



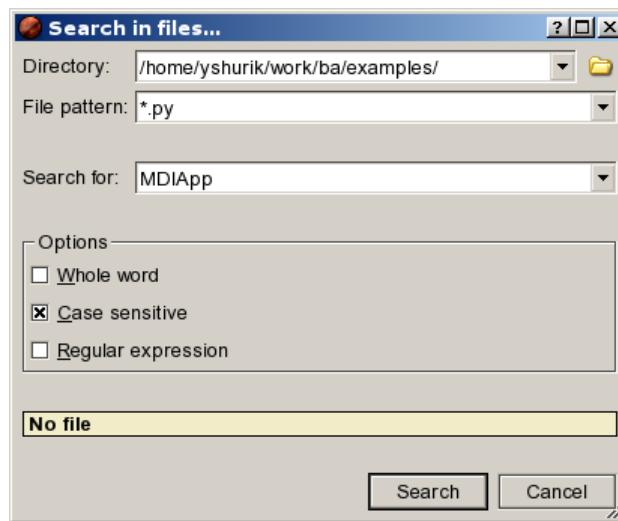
The new form dialog allows you to select a template for your form and select a file name of where to save the user interface file. After clicking “Ok” Form Editor (Qt Designer) will be run.

### 4.12.2 Find



The Find dialog is standard and very easy to use. Just type the text you are looking for and check the options that you need. You can use accelerators to quickly select options that you need – for example you can switch check box “Case sensitive” with Alt-C.

### 4.12.3 Search in files



Search in files allow you to search through a set of files using regular expressions. In the dialog you can select a directory where you want to search, file pattern and search text or regular expression. You can also use options to search for a whole word, case sensitive option if the search text is a regular expression. The yellow status line shows you the file where the search is currently. Search results will output in a message panel, as seen below:

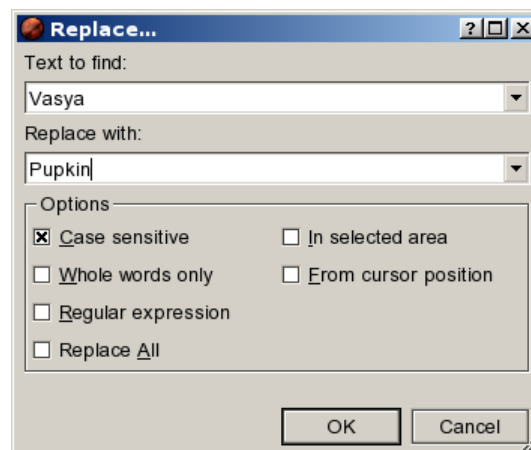
```

/home/yshurik/projects/mdi/mdi.py
195:     w = MDIWindow( self.ws, "", Qt.WDestructiveClose )
238:     "Qt's Multiple Document Interface (MDI).")
267: class MDIWindow( QMainWindow):
356:     mw.setCaption("PyQt Example - Multiple Documents Interface (MDI)")
/home/yshurik/projects/mymdi/mdi.py
195:     w = MDIWindow( self.ws, "", Qt.WDestructiveClose )

```

As you can see, after searching for the word “MDI”, there is a full report in the message file about the files and lines that contained the searched for text. You can just click on a line in the message panel and it will open BlackAdder with that text available.

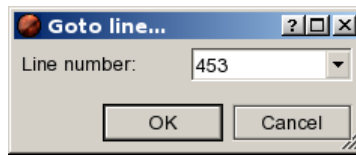
#### 4.12.4 Replace



The “Replace” dialog will allow you to search and replace text in your file. Just enter the text you want to replace and the text to replace it with. You can select a variety of options such as matching case, limiting the range of search, etc. If you enter a regular expression in to the text window, then that will be processed instead of a literal translation. It is still possible to use Undo to cancel this operation.

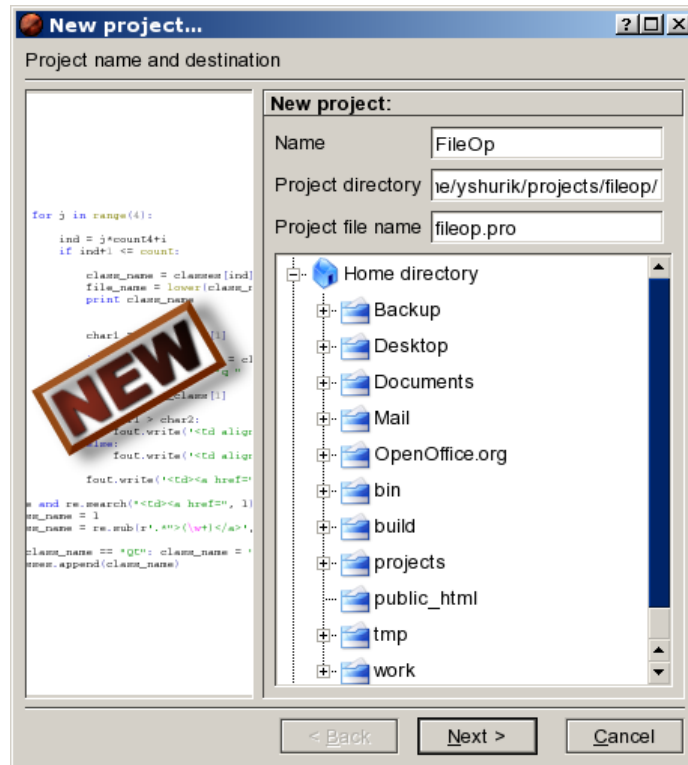
#### 4.12.5 Go to line



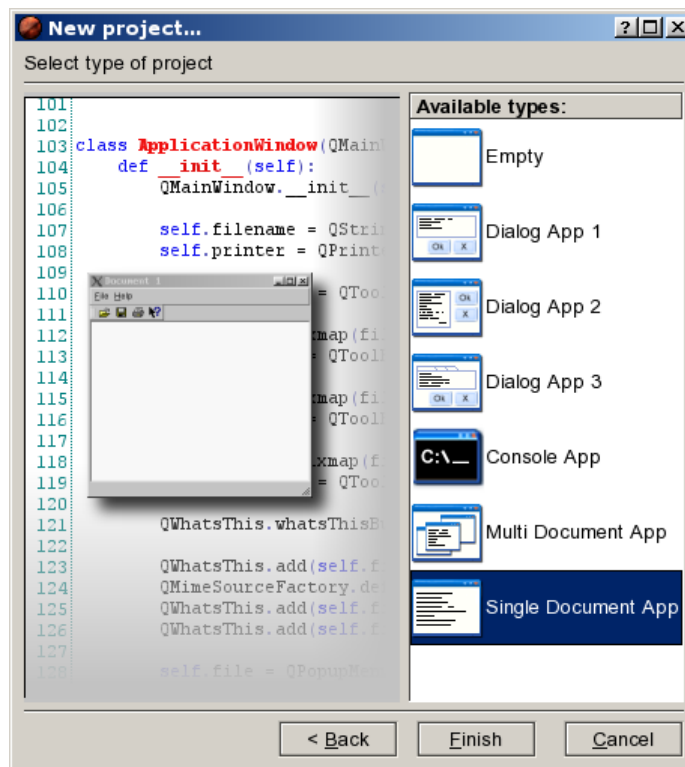


“Go to” line dialog allow you to quickly jump to an existing line with some number. Note that you can use the features by pressing “Ctrl-G”->Enter line number->press Enter – this combination allows you to very quickly jump through text.

#### 4.12.6 New project

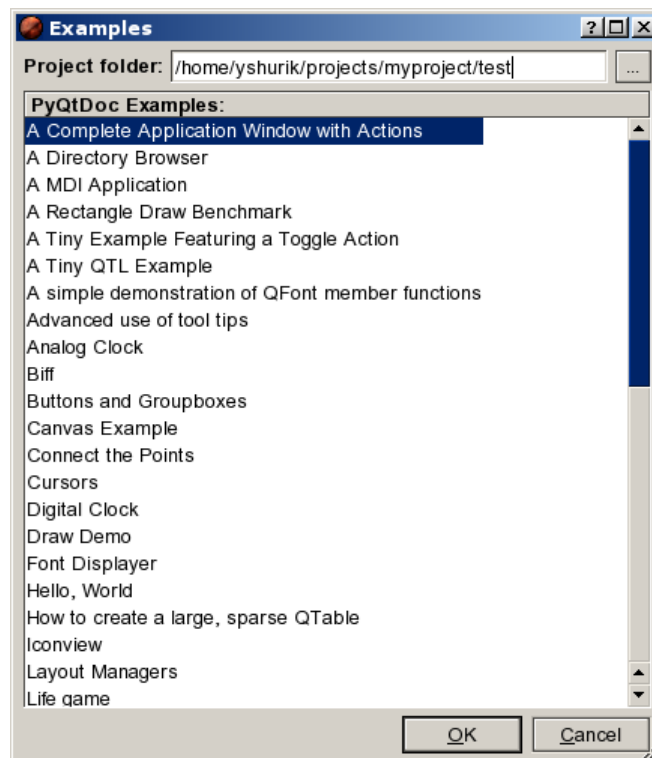


New project dialog allows you to create your new project and use a set of wizards released with BlackAdder to quickly generate skeleton code for your project. The first page of the wizard contains next a number of things you need to enter, such as: project name – the name of the project that will be shown in the list of projects in project panel. Project directory – this is the directory where all the files in your project will be stored (note: that you can't have files part of a project outside of the directory, only files included in the directory and subdirectories), and the name of the file where the project parameter XML (note: that the project file also will be included in the project directory). Here you can see an example of what it looks like:



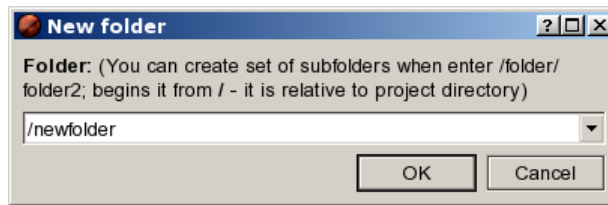
The next page allows you to create the type of project. Depending on the type you select BlackAdder will run a wizard where you can select options for generating code for your files.

#### 4.12.7 New project from example



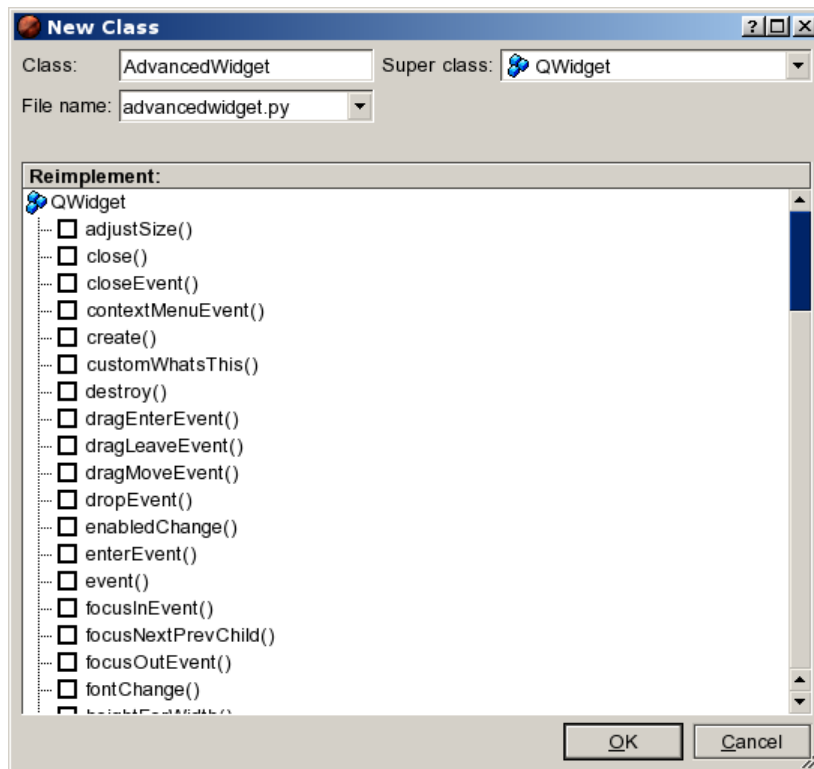
The dialog provide you a clean and quick way to start from a template. In the dialog you can select an existing template (we use templates from PyQt examples) select a directory where the files of your project will be put. After clicking on "Ok", the files will be copied to the directory and your project will be opened in BlackAdder. This is a good way to study PyQt for the first time – you can create your first projects from the examples, study the Python code and change what you need.

#### 4.12.8 Add folder



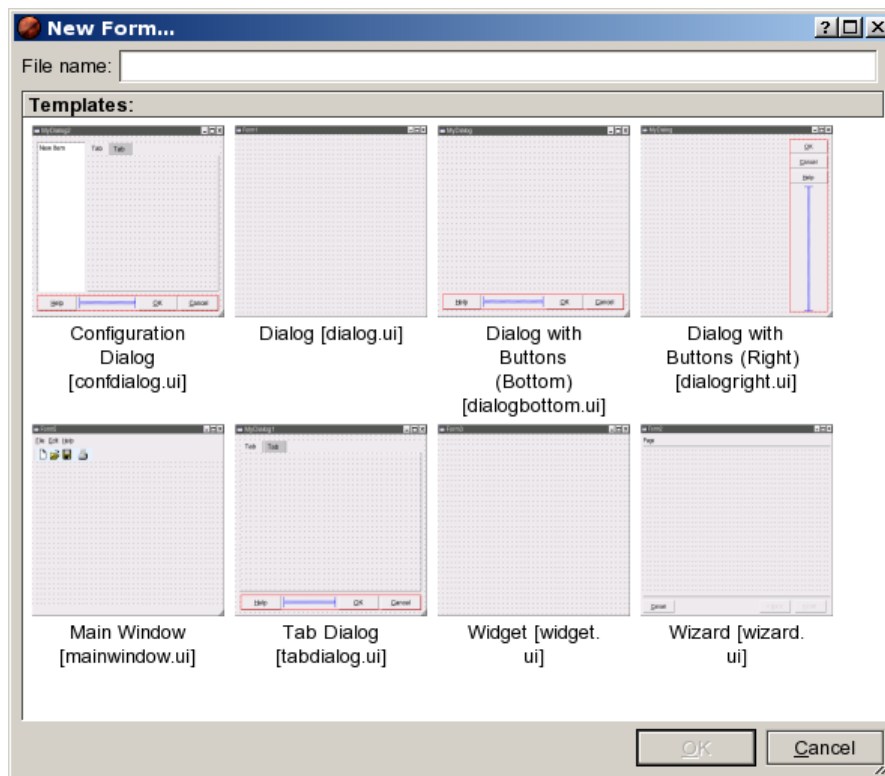
The “Add folder” dialog allows you to create a new folder in your project or whole chain of subdirectories. You just need to write the name of a folder, starting with a file system separator or all subdirectories like: /folder1/folder2/folder3 – all non-existing directories will be created.

#### 4.12.9 Add class



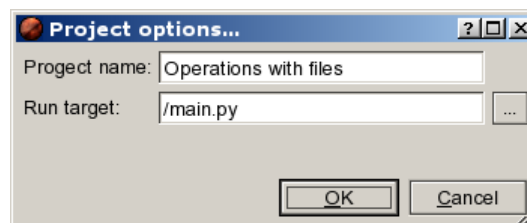
This dialog provides a wizard as a simple method to create a new class or sub class it from your existing classes or just from PyQt classes. You need to enter the name of a new class and file of where to save the class (note: that if the file already exists, it will not be overwritten, the class will be just be appended to the existing file). A combo box for super class support has auto completion so you can enter the first few letters of a class, press Enter and navigate with the arrow buttons (up, down) to select the class you want to sub class. Below is a dialog that shows a list of virtual functions you can reimplement in your new class (note: that not all functions in the PyQt library are virtual because it is not possible with bindings to Qt C++ library where not all functions are virtual, but in the list are shown only virtual functions that you can reimplement). If you scroll down the list you will see a super class which you can open and see their functions, which you also can reimplement.

#### 4.12.10 Add form



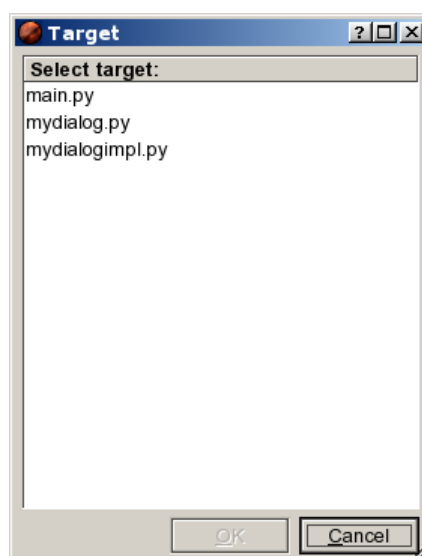
The “Add form” dialog allows you to select a template for your form and select a file name for your user interface file. After clicking on “Ok” the Form Editor (Qt Designer) will be started. Differences in the “New Form” dialog are that you don't need to enter the whole path to the file, you need to just enter the short name. After you click on “Ok”, the file will be added to the project.

#### 4.12.11 Project options



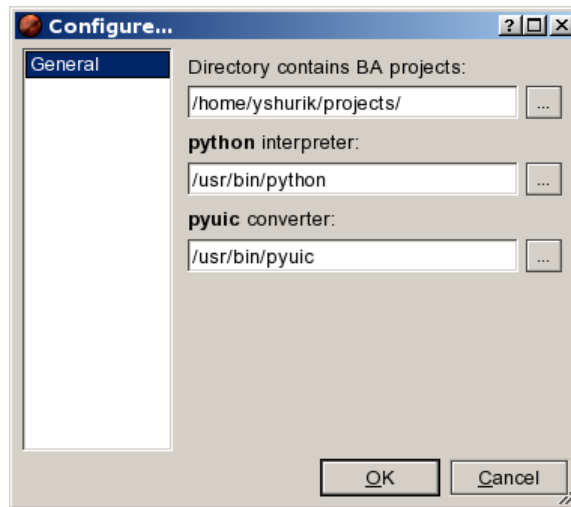
This dialog will allow you to change the project name and target (file that will be executed in Python interpreter when you will run/debug project).

#### 4.12.12 Select target



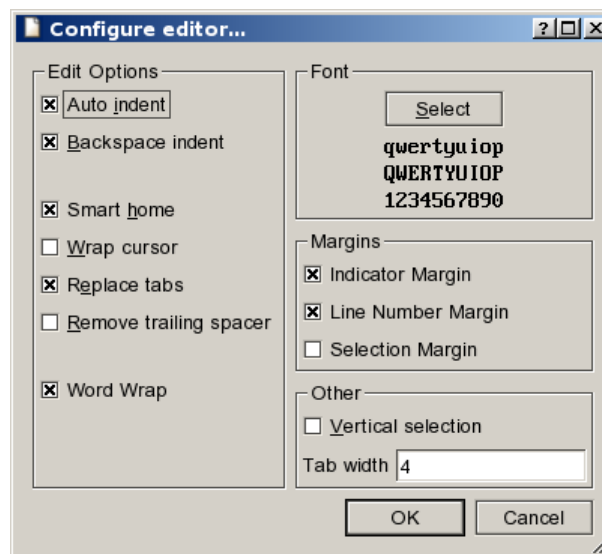
Select target dialog contains a list of project files. You need to select a file that will be executed when you run your project or when you start debugging your project.

#### 4.12.13 Configure



The configure dialog will allow you to change various IDE variables as a directory for your projects. All new projects you will create will be at default stored in the directory. You can also set the file name of the Python interpreter and pyuic converter if the default are not found.

#### 4.12.14 Configure editor



In this dialog you can select the behavior of the BlackAdder editor. You can select the font of editor text, margins which you want to see – indicator margin, that shows breakpoint icons, line number margins and selection margin - you can select blocks of texts with margin and use drag and drop to move blocks. Below you can select tabulation width and options for vertical selection (not too useful for Python coding, but can be useful in some situations). Edit options contains group of indent options – auto indent (first text symbol of new line starts at same position as line above – but if you end line above with “:” it will also add an indent), backspace indent is an option if all indent space can be cleaned with one backspace. Smart home is options for behavior of action on pressing Home button. If the options checked then first press of Home will set cursor at first non-space symbol of line (not to just first symbol of line). Replace tabs allows you to change tabs to spaces. The word wrap option applies to opening files – if the option is set then new editor window will be opened with word wrap. Note that Menu->Editor->Word wrap applies to current editor document only, new files will be opened in word wrap mode from the dialog option.

## 5 Using Qt Designer

### 5.1 Creating forms

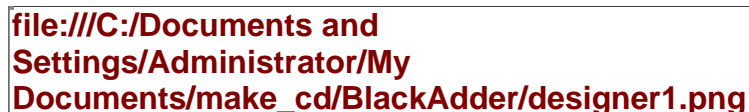
Creating a form in BlackAdder is very simple. There are two ways: first is just create from without project – then you need to just click on “New Form” on the tool bar – it will open a dialog where you need to select a filename and an existing template for the form. A second way is when you have a project opened – then you can create a new form from the project menu or by right-button popup menu on projet panel (you need press the right-button on the folder item or on project). After “New Form” dialog Form Editor will be executed.

### 5.2 Export forms from other projects

You can use forms from other C++/Python projects without restrictions. It is the same ui file interface created with Qt Designer, so it is no problem importing user interface from other projects.

### 5.3 Extending the functionality of a form

First let's look at a small figure that shows the relationship between .ui files, generated code and application code:



```
file:///C:/Documents and
Settings/Administrator/My
Documents/make_cd/BlackAdder/designer1.png
```

*Qt Designer* reads and writes .ui files, e.g. `form.ui`. The user interface compiler, `pyuic`, creates an implementation file, e.g. `form.py`, from the .ui file. The application code in `main.py` imports a `form`. Typically `main.py` is used to instantiate the `QApplication` object and start off the event loop.

While this approach is simple, it isn't sufficient for more complex dialogs. Complex dialogs tend to have quite a lot of logic attached to the form's widgets, more logic than can usually be expressed with predefined signals and slots. One way of handling this extra logic is to write a controller class in the application code that adds functionality to the form. This is possible because `pyuic` generated classes expose a form's controls and their signals to the public space. The big disadvantage of this method is that it's not exactly Qt-style. If you were not using *Qt Designer*, you would almost always add the logic to the form itself, where it belongs.

This is why the capability of adding custom slots and member variables to a form was added to *Qt Designer* early on. The big additional benefit with this approach is that you can use *Qt Designer* to connect signals to those custom slots, in the same elegant graphical way that is used to connect signals to predefined slots. The `uic` then adds an empty stub for each custom slot to the generated `form.py` implementation file.

The big question now is how to add custom implementation code to those custom slots. Adding code to the generated `form.py` is not an option, as this file gets recreated by `pyuic` whenever the form changes -- and we don't want a combination of generated and handwritten code. There is an alternative solution, which we'll cover next.

### 5.4 The subclassing approach

A very clean way to implement custom slots for generated forms is via Python inheritance as shown in the next figure:



file:///C:/Documents and Settings/Administrator/My Documents/make\_cd/BlackAdder/designer2.png

Here the user wrote an additional class **FormImpl**, and the implementation file `formimpl.py`. The file imports the `pyuic` generated `form` and reimplements all the custom slots. This is possible because `uic` generated custom slots are virtual. In addition to implementing custom slots, this approach gives the user a way to do extra initialization work in the constructor of the subclass, and extra cleanups in the destructor.

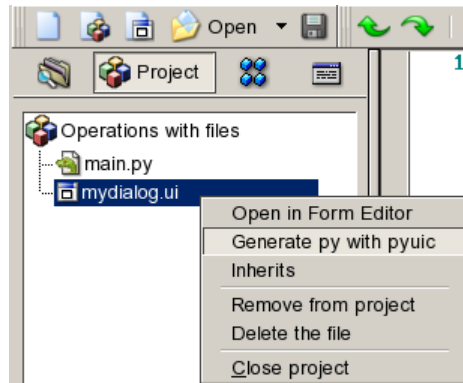
Because of these benefits and its flexibility, this approach became the primary way of using *Qt Designer* in Qt 2.x.

**Note:** To keep the namespace clean, most users did not follow the `Form` and `FormImpl` naming scheme shown in the figure, but instead named their *Qt Designer* forms `FormBase` and their subclasses `Form`. This made a lot of sense, because they always subclassed and were using those subclasses in application code.

## 5.5 Step by step

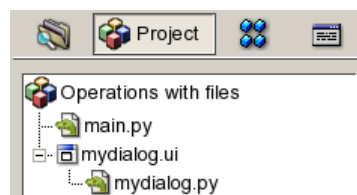
### Step 1:

You have created your form in Qt Designer. You include the form in your project (or create with Project->Add form). You need to generate a Python implementation of the form. To do it, press the right mouse button on the ui file in the project panel and select "Generate py with pyuic".



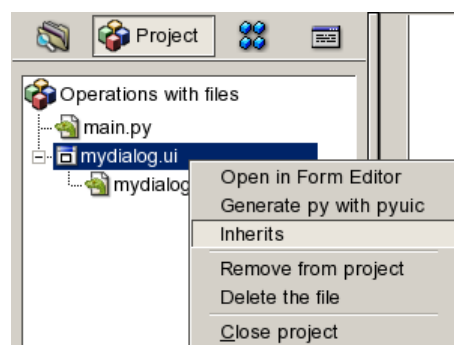
### Step 2:

Now you can see in the project panel sub item "mydialog.py". It is implemented as a sub item for splitting your project files and generated files. Useful when projects have a lot of forms and your Python files.



### Step 3:

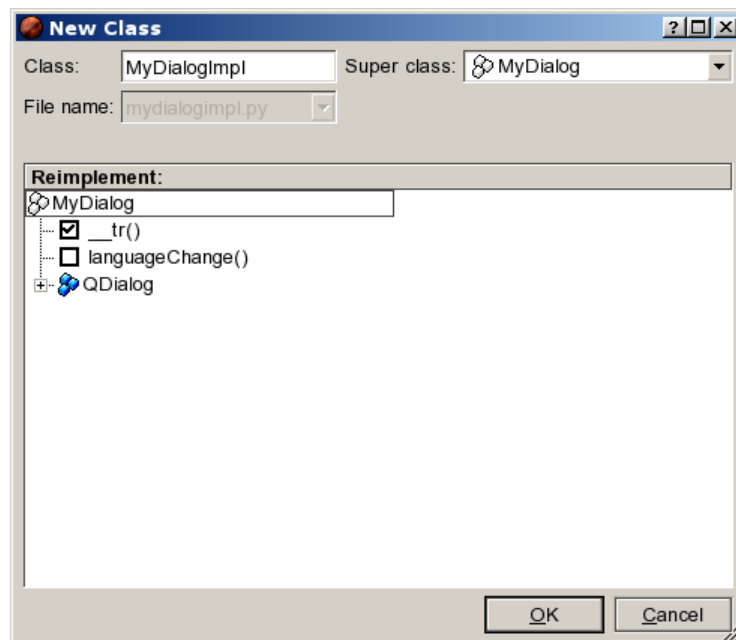
You have created an implementation file. You need to sub-class the class of the designer form. To do it press the right mouse button on ui file in project panel and select "Inherits".



### Step 4:

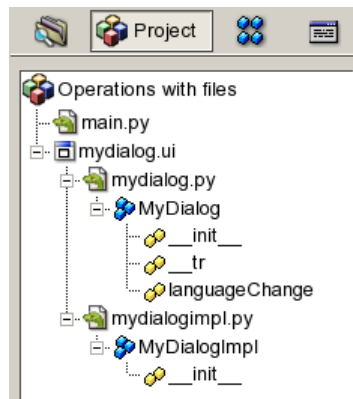
Now you see a small wizard for subclassing classes created with pyuic. Here you can enter the name of your new class and select a list of virtual functions which you want to reimplement. You can open a tree of inheritance and see all the functions that you can reimplement (qt virtual functions).





**Step 5:**

After these operations you will have the following structure of classes:



## 6 Questions and answers

### 6.1 Known bugs and bug reporting

If you have found a bug or have a feature request that you want to see in an upcoming version of BlackAdder you can post it in our bugtracking system – Mantis online bug reporting system to tell us about the bug. This is the fastest and easiest way to let us know about it, and to provide us with the needed information to fix it as quickly as possible.

Mantis bug reporting system

[https://www.thekompany.com/mantis/login\\_page.php](https://www.thekompany.com/mantis/login_page.php)

Use this for reporting all bugs to us, for any theKompany.com software (both commercial software and open source projects).

Mantis documentation

<https://www.thekompany.com/mantis/doc/documentation.html>

If you need help using Mantis, use the link above.

Just type in your browser:

[https://www.thekompany.com/mantis/login\\_page.php](https://www.thekompany.com/mantis/login_page.php),

register there and post your bug or feature request.

### 6.2 BlackAdder maillist.

For communication with other BlackAdder users or developers you can use our [maillist](mailto:ba@thekompany.com) (ba@thekompany.com) where you can talk about your ideas or questions with other users and our BlackAdder developers.

For subscribing you need check <http://www.thekompany.com/mailman/listinfo/ba>

### 6.3 BlackAdder/PyQt forum.

For exchanging information we created a PyQt forum where you can post your problem or just search for existing problems and solutions. Also we are posting announcements and news to the forum.

You can access the forum at <http://www.thekompany.com/pyqt/>

### 6.4 Where to find more information.

As you can see ahead you can read our maillist and our pyqt forum. There is PyKDE maillist which is related to sip/pyqt/pykde problems only. There is also a book about PyQt programming (but related to old beta4 of BA) You can access it online at <http://www.opendocs.org/pyqt/>.

There are few a articles that you can find on official site of PyQt.

Check <http://www.riverbankcomputing.co.uk/articles.php>

## 6.5 How to buy BlackAdder.

You can order BlackAdder applications from  
<http://www.thekompany.com/products/blackadder/>

We accept payment by:  
Credit card (*VISA, American Express, Mastercard, Discover*)  
Check (*US or international*)

Checks and order forms can also be sent by ordinary mail to:

theKompany.com  
PO Box 80265  
Rancho Santa Margarita, CA 92688  
USA

For questions regarding purchase and pricing, please contact us at  
[sales@thekompany.com](mailto:sales@thekompany.com) or phone (949) 713-3276.

## 6.6 Product support

theKompany.com is offering support contracts for all its products. As part of your support contract you will receive 24 hour turn around on your initial request.

Our support services are broken into two categories, developer tools and desktop applications. For each category there is one of three support tiers, Per Incident, Personal Annual and Corporate Annual.

As an introductory offer, the Personal Annual support contract for desktop applications is being offered at half price and will include any and all products that come out in this category even if they are released after you purchase your contract. Pricing is as follows:

Per Incident **\$15 USD**

Personal Annual **\$99 USD** for Developer Tools

Personal Annual **\$49.95 USD** for Desktop Applications

Corporate Annual **\$3,000 USD** which includes up to 10 registered users

You can purchase support with any of our supported Credit Cards, a personal check, a money order, electronic wire, bank draft or Purchase Order. All forms of payment subject to funds clearing before the contract is activated, Credit Cards contracts are approved immediately.

## 7 Authors

### **BlackAdder:**

Oleksandr Yakovlyev <[yshurik@thekompany.com](mailto:yshurik@thekompany.com)>  
Dmitry Poplavsky <[dima@thekompany.com](mailto:dima@thekompany.com)>  
Igor Skrygun <[bender@thekompany.com](mailto:bender@thekompany.com)>

### **PyQt:**

Phil Thompson <[phil@riverbankcomputing.co.uk](mailto:phil@riverbankcomputing.co.uk)>

### **PyQtDoc:**

Oleksandr Yakovlyev <[yshurik@thekompany.com](mailto:yshurik@thekompany.com)>

### **PyQt Documentation:**

Oleksandr Yakovlyev <[yshurik@thekompany.com](mailto:yshurik@thekompany.com)>  
Igor Skrygun <[bender@thekompany.com](mailto:bender@thekompany.com)>  
Aleksander Kravchuk <[krav@thekompany.com](mailto:krav@thekompany.com)>