Programação gráfica com Qt - Parte 4

Bem, hoje chegamos à quarta parte deste artigo tutorial sobre programação gráfica com Qt, onde trataremos do Qt Designer: uma ferramente muito útil para fazer o desenho das telas.

O QtDesigner permite que você desenhe uma janela apenas usando cliques do mouse. Serve apenas para isso (na Qt 2, pois o Qt Designer distribuído com a Qt 3 já tem editor de código integrado).

Os passos são os seguintes:

1. Desenhar uma janela
2. Definir as propriedades dos widgets
3. Definir slots para a janela
4. Definir ligação entre os sinais dos widgets e os slots
5. Criar uma nova classe C++ derivada da janela que criamos no Qt Designer
6. Escrever o código para os slots

Os passos 4 e 5 podem parecer estranhos a princípio, mas veremos tudo passo a passo.

Primeiro, precisamos entrar no Qt Designer. Normalmente ele está em $QTDIR/bin/designer.

Uma tela semelhante à figura 1 deve ser aberta:

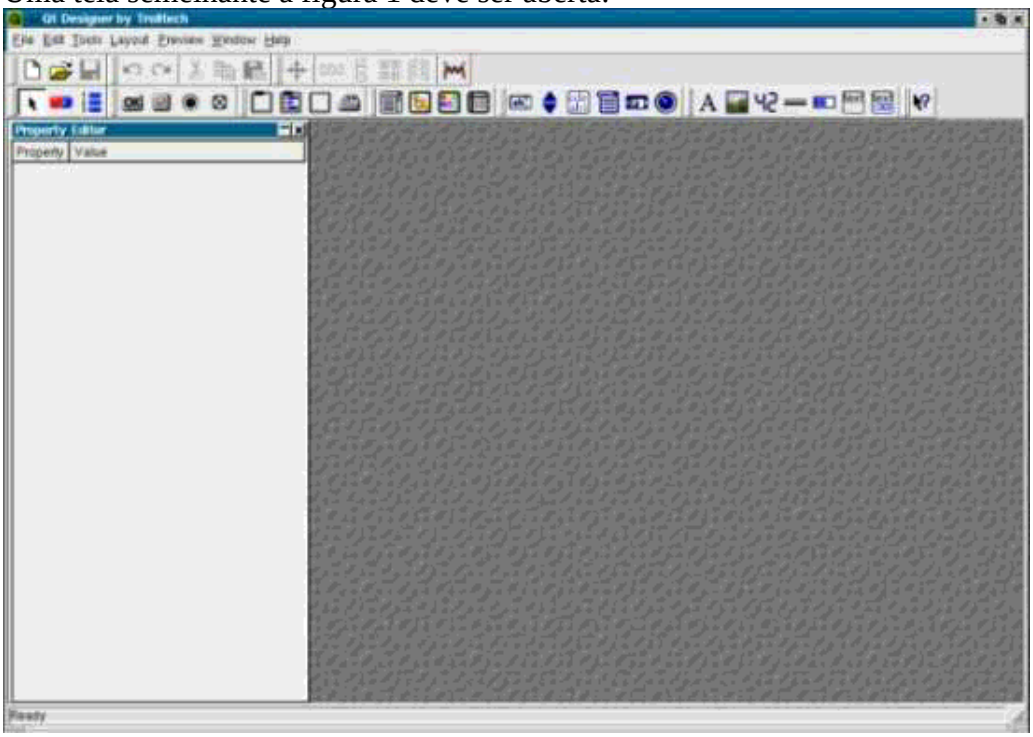

Figura 1

Agora, clicando em File->New, você verá uma janela como a da figura 2. Clique em Widget e em OK.

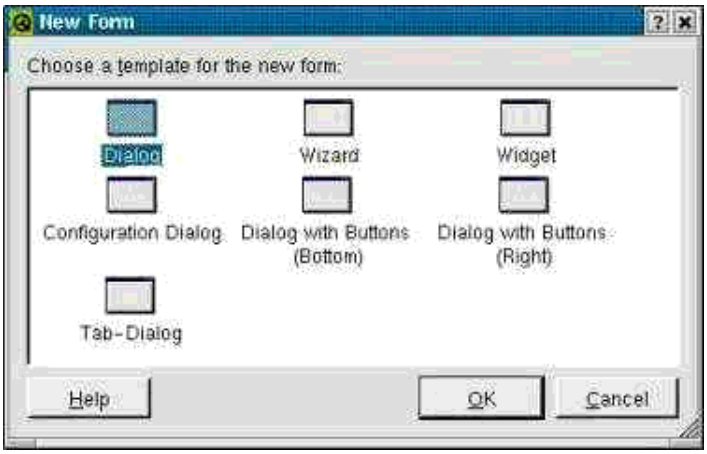

Figura 2

Pronto, você já tem uma janela em branco para desenhar.

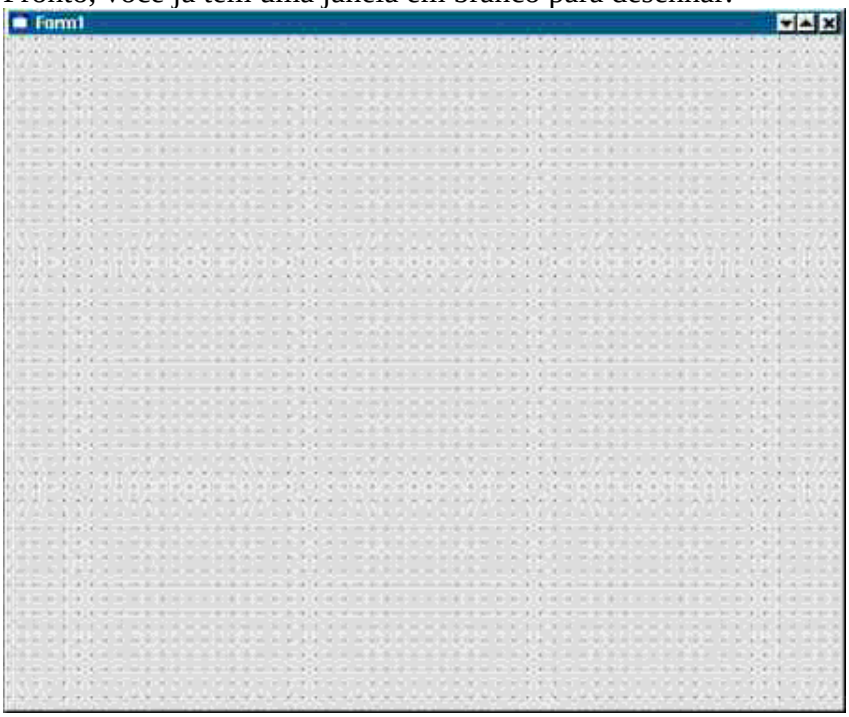

Figura 3

Você pode redimensionar a janela, incluir widgets, mudar as cores, etc. Vamos desenhar uma janelinha simples usando QPushButton, QMultiLineEdit e QLayout(s). Todos os componentes podem ser encontrados na barra de ferramentas na parte superior da janella principal do Qt Designer.


Figura 4

- QPushButton

- QMultiLineEdit


- QLayouts

Para acrescentar um Widget, basta clicar em seu ícone na barra de ferramentas e clicar na janela. O widget aparecerá e poderá ser movido e redimensionado.

Para o nosso exemplo, desenhe uma janela usando um QMultiLineEdit e dois QPushButton, de forma que fique parecida com a janela da figura 5:

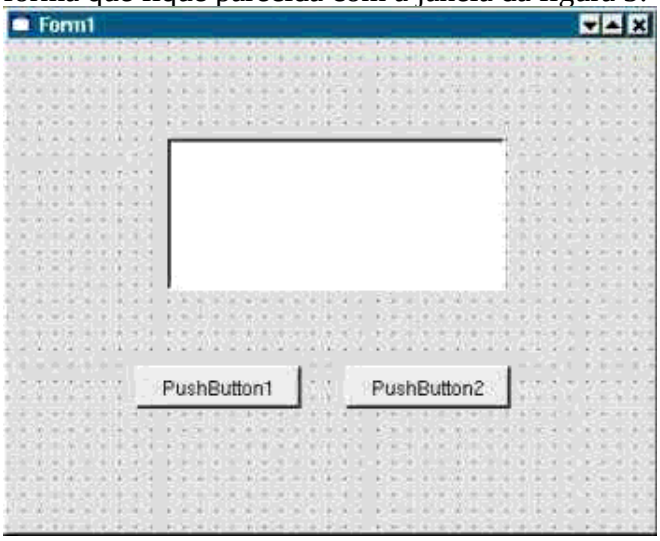

Figura 5

Agora, selecione os dois botões e clique no botão (na barra de ferramentas) para layout horizontal. É o primeiro botão da parte de layouts. Em seguida, clique numa parte vazia da janela e depois clique no botão para layout vertical (o segundo da parte de layouts). Você verá uma janela mais alinhada:

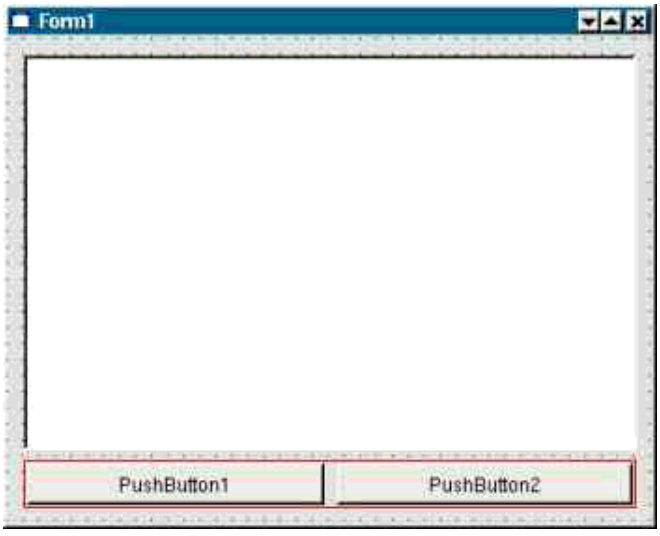

Figura 6

Já temos o desenho pronto, mas como deve imaginar, os botões deveriam conter outro texto, e não PushButtonX. Vamos mudá-los:

1. Clique em PushButton1
2. Na janela Property Editor, procure pela propriedade text
3. Troque seu valor para "&Gravar..." (sem as aspas)
4. Faça os mesmo com o PushButton2, trocando o texto para "&Fechar"

Também, troque a propriedade name de todos os componentes, para:

- texto (para o QMultiLineEdit)
- gravar
- fechar

Também troque o nome da janela, de Form1 para Editor_I (o _I no final é para indicar que esta tela trata-se somente da interface e não da implementação. No final desta página ficará claro o motivo).

Ok... nada funciona ainda. Mas vamos ver como ficou... "Vamos compilar já?" Não. Ainda não. Podemos ver o nosso trabalho em funcionamento antes de compilá-lo. Para isso temos o menu Preview, no Qt Designer. Clique nele e escolha um dos diversos estilos para a visualização (o padrão é Preview Form). Os estilos não fazem parte do escopo deste artigo.

Tudo certo? Então vamos definir a funcionalidade dos botões. Novamente, entra o conceito de sinais e slots. Vamos criar um slot para cada botão: gravarArquivo() e fecharJanela(). Podemos fazer isso clicando na opção Slots do menu Edit. Veremos uma tela como esta:

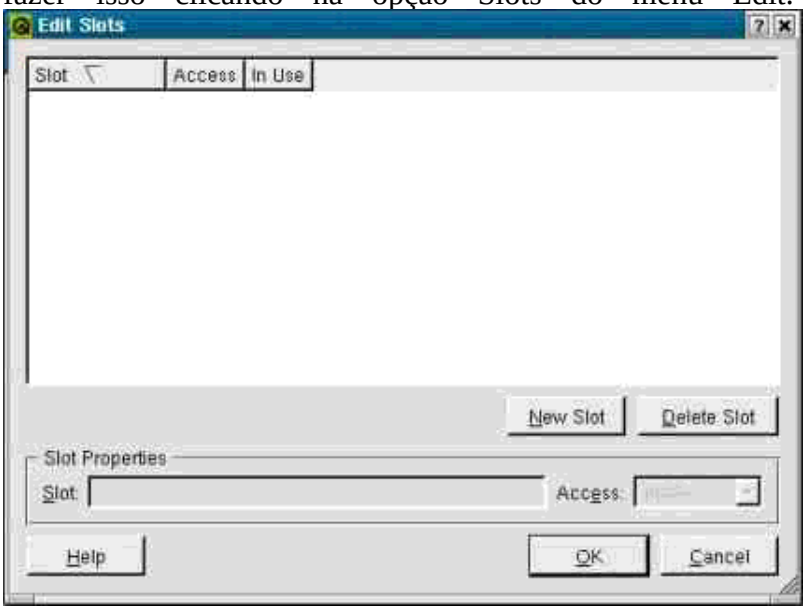

Figura 7

Clique em New Slot e em Slot digite gravarArquivo(). Faça o mesmo para o fecharJanela() e clique em OK.

Aqui nós apenas definimos a existência dos dois slots, porém ainda não criamos nenhum código para eles e também não os conectamos aos botões.

Nota: A versão 2.x da Qt ainda não permite a edição de código no Qt Designer. Isso só passou a ser possível na Qt 3, lançada em 15 de outubro de 2001. Como nosso artigo se baseia na Qt 2.x (2.3.0 para ser mais específico) não veremos a edição no Qt Designer.

O próximo passo então é conectar os slots aos botões. Bom, será que é algo como:

connect(fechar, SIGNAL(clicked()), self, SLOT(fecharJanela()));

Não. Conexões simples o Qt Designer pode fazer sem código. Rápido e fácil. Na barra de

ferramentas, procure pelo botão Connect Signal/Slots  . Clique. O cursor tomará uma forma de cruz. Clique no botão Fechar e arraste para uma área vazia da janela:

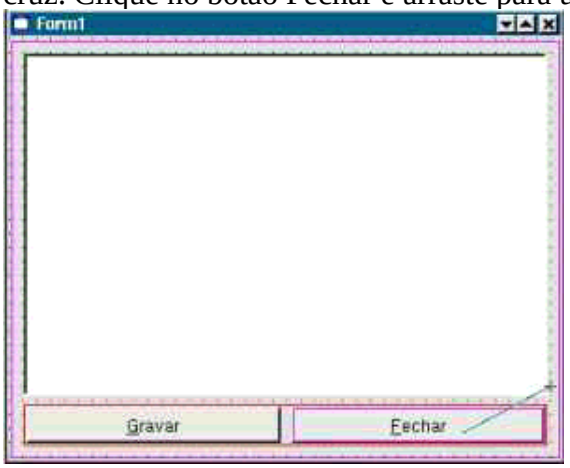

Figura 8

Você verá uma janela semelhante a esta:

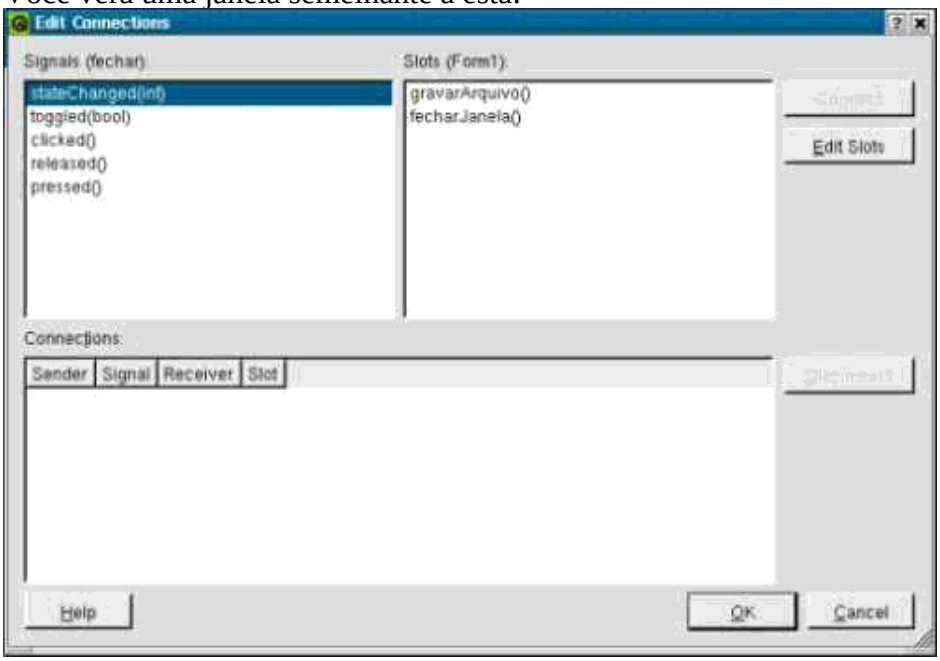

Figura 9

Basta clicarmos no sinal clicked() e no slot fecharJanela(). Você já poderá ver a conexão na lista de conexões. Faça o mesmo para o Gravar e clique em OK.

Pronto! Tudo o que poderíamos fazer dentro do Qt Designer já está feito. Grave o arquivo como editor_i.ui.

Agora temos os seguintes passos:

- Transformar a janela de .ui para .cpp
- Criar uma implementação para a janela
- Compilar tudo

Mas antes de tudo, vamos testar a nossa janela, agora já compilada. Para isso, precisamos executar o

primeiro dos passos acima. Para transformar de .ui para .cpp, execute os seguintes comandos:

uic -o editor_i.h editor_i.ui
uic -i editor_i.h -o editor_i.cpp editor_i.ui

Isso vai criar dois arquivos em seu diretório: editor_i.h e editor_i.cpp. Você pode observar que o código gerado é um código comum em C++/Qt, similar ao que vimos nas partes anteriores. Evite alterar esses arquivos manualmente, pois você pode se esquecer disso e rodar os dois comandos acima novamente, perdendo assim as alterações.

Agora, podemos usar o main.cpp da parte anterior, alterando o #include para #include "editor_i.h" e a linha MyWidget w para Editor_I w.

Também não podemos nos esquecer do .moc:

moc -o editor_i.moc.cpp editor_i.h

E compilamos tudo:

g++ -I $QTDIR -lqt -o editor editor_i.cpp editor_i.moc.cpp main.cpp ./editor

Note que se clicarmos nos botões, aparecerão mensagens como essas:

Editor_I::gravarArquivo(): Not implemented yet!
Editor_I::fecharJanela(): Not implemented yet!

Isso está implementado no arquivo editor_i.cpp. Se quisermos podemos alterá-lo, mas isso não é vantagem (veja explicação acima). Para termos a janela funcionando sem alterar o nosso editor_i.cpp, precisamos criar uma subclasse da mesma. Podemos criar uma classe chamada Editor, tendo Editor_I como classe principal. Isso faz com que todas as propriedades da classe Editor_I sejam passadas para Editor, e assim, podemos alterar a sua funcionalidade. Vejamos na prática.

Precisamos criar agora um arquivo chamado editor.h, com o seguinte código:

```
 1: #ifndef EDITOR_H
 2: #define EDITOR_H
 3: #include "editor_i.h"
 4:
 5: class Editor : public Editor_I
 6: {
 7: Q_OBJECT
 8:
 9: public:
10:   Editor(QWidget *parent=0, const char *name=0, WFlags fl=0);
11:
12: public slots:
13:   void gravarArquivo();
14:   void fecharJanela();
15: };
16: #endif // EDITOR_H
```

Veja que declaramos apenas o que vamos alterar (além do constructor Editor), que são os dois slots. Agora vamos implementá-los (apemas o fecharJanela()). Crie um arquivo editor.cpp.

```
#include "editor.h"

Editor::Editor(QWidget *parent, const char *name, WFlags fl)
    : Editor_I(parent, name, fl)
{
}

void Editor::gravarArquivo()
{

  qWarning("Ainda não estou funcionando. Vou fazer algo somente na próxima
parte...");

}

void Editor::fecharJanela()
{
  this->close();
}
```

Tudo o que digitamos acima já foi visto em partes anteriores da série de artigos. Apenas note que o constructor Editor chama Editor_I e não QWidget. Como não vamos fazer nada no início de Editor, deixamos a função em branco.

Pronto! Agora, como exercício, altere o arquivo main.cpp para que ele chame a janela através da classe Editor, e não mais Editor_I.

Agora, vamos criar um arquivo moc e compilar tudo:

moc -o editor.moc.cpp editor.h
g++ -I $QTDIR -lqt -o editor editor.cpp editor.moc.cpp editor_i.cpp editor_i.moc.cpp main.cpp
./editor

E essa foi a quarta parte do nosso artigo sobre programação gráfica com Qt. Espero que esses artigos estejam sendo úteis. Qualquer dúvida, sugestão, crítica, contato, elogio ou correção podem ser enviadas para o meu e-mail (na assinatura deste artigo). Na próxima parte daremos apenas a funcionalidade para o botão Gravar utilizando classes da Qt. Até lá.

Comunidade GNU/Linux do Brasil
Autor: Ricardo Vaz Mannrich
Email: husk-le@comlinux.com.br