Programação gráfica com Qt - Parte5

Hoje vamos concluir a série de tutoriais da biblioteca gráfica Qt. Na última parte, escrita há muito tempo atrás, vimos como usar o Qt Designer para o desenho das nossas janelas, bem como dar funcionalidade aos Widgets (no caso, o QButton). Faltou apenas o botão Gravar. Hoje então faremos:

- Abrir uma janela perguntando o nome do arquivo onde o texto deve ser gravado
- Gravar o arquivo usando classes da Qt

A primeira coisa que vamos precisar é fazer o programa abrir uma janela perguntando o nome do arquivo assim que clicamos (está certo assim?) no botão Gravar. Muita gente me pergunta como abrir uma segunda janela na Qt. Acredito que o uso da QFileDialog possa ajudar. Bem, como vamos usar mais uma classe, precisamos incluir o seu cabeçalho no início do nosso editor.cpp:

#include <qfiledialog.h>

O QFileDialog é responsável pelas janelas de Abrir e Gravar.

Além do cabeçalho, precisamos usá-lo. Se você pesquisou outro material a respeito de Qt (e até mesmo C++) viu que existem várias maneiras de se utilizar um objeto. No nosso caso, como o QFileDialog vai ser usado apenas no *Editor::gravarArquivo()*, vamos usar esta classe com escopo local. Ou seja, apenas dentro de *Editor::gravarArquivo()*. Na parte anterior, *gravarArquivo()* estava assim:

```
void Editor::gravarArquivo()
{
qWarning("Ainda não estou funcionando. Vou fazer algo somente na próxima par-
te...");
}
```

Agora, vamos mudar para:

```
void Editor::gravarArquivo()
{
QFileDialog *dlg;

dlg=new QFileDialog(this, 0, TRUE);
dlg->getSaveFileName("new.txt", "Arquivo texto (*.txt)");
}
```

Note os parâmetros que foram passados. Em *dlg= new QFileDialog(this, 0, TRUE)*, estamos informando que o pai da janela será o nosso editor (this).

Na linha seguinte, quando abrimos a janela: *dlg->getSaveFileName("new.txt", "Arquivo texto (*.txt)", this)*, informamos um nome inicial para o arquivo (new.txt), o filtro (será exibito somente aruivos com final .txt: Arquivo texto (*.txt)).

Agora, compile o programa (como faz tempo que passei a última parte, vou colocar todos os comandos para a compilação. Vale lembrar que este não é o melhor método para se compilar tudo. Veja o utilitário make):

uic -o editor_i.h editor_i.ui
uic -o editor_i.cpp -i editor_i.h editor_i.ui
moc -o editor_i.moc.cpp editor_i.h
moc -o editor.moc.cpp editor.h
g++ -I$QTDIR/include -L$QRDIR/lib -lqt -o editor editor.cpp editor.moc.cpp editor_i.cpp editor_i.moc.cpp main.cpp

Execute-o:

./editor

Digite qualquer coisa na área de texto e clique no botão Gravar. Você verá uma tela como esta:

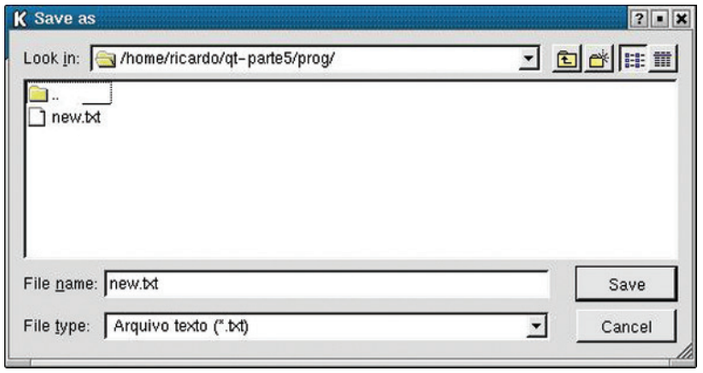

Figura 1

Tanto faz clicar em *Ok* ou *Cancelar*, pois não vai acontecer nada mesmo. Bom, se não vai acontecer nada, temos que fazer acontecer. Primeiro de tudo, vamos fazer o nosso programa mostrar o nome do arquivo onde será gravado o texto. O retorno da função *QFileDialog::getSaveFileName(...)* é um QString. Como vamos usar esse QString, precisamos incluir seu cabeçalho no início do *editor.cpp*:

#include <qstring.h>

E vamos declarar uma variável *filename* que armazenará o nome do arquivo selecionado:

QString filename;

getSaveFileName() retorna o nome do arquivo se foi clicado em *Ok*, ou um QString nulo se a janela foi fechada ou *Cancelar* foi clicado. Podemos incluir esse teste em nosso exemplo:

```
void Editor::gravarArquivo()
{
   QFileDialog *dlg;
   QString filename;
   QFile txt_file;
   QTextStream txt_stream;

   dlg=new QFileDialog(this, 0, TRUE);
   filename=dlg->getSaveFileName("new.txt", "Arquivo texto (*.txt)", this);

   if (filename.isNull()) {
```

```
      qWarning("Assim não vale, né... É só eu aparecer e você me cancela...");
   } else {
      qWarning("Valeu... O nome do arquivo é: %s.", filename.latin1());
   }
}
```

latin1() é necessário para exibição com *qWarning()*.

Compile novamente e faça o teste. Tente as duas possibilidades. Note que quando se seleciona um arquivo ele retorna o nome completo, incluindo o caminho.

Agora precisamos gravar o arquivo. Para isso vamos utilizar outras duas classes: QFile e QTextStream. QFile é o que vai fazer todo o tratamento de arquivo e QTextStream vai jogar o texto no arquivo. É mais fácil mostrar o código a explicar. Vaja o código (estou incluindo todo o código do editor.cpp, para finalizar):

```
 1: #include <qmultilineedit.h>
2: #include <qfiledialog.h>
3: #include <qstring.h>
4: #include <qfile.h>
5: #include <qtextstream.h>
6: #include "editor.h"
7:
8: Editor::Editor(QWidget *parent, const char *name, WFlags fl)
9: : Editor_I(parent, name, fl)
10: {
11: }
12:
13: void Editor::gravarArquivo()
14: {
15: QFileDialog *dlg;
16: QString filename;
17: QFile txt_file;
18: QTextStream txt_stream;
19:
20: dlg=new QFileDialog(this, 0, TRUE);
21: filename=dlg->getSaveFileName("new.txt", "Arquivo texto (*.txt)", this);
22: if (filename.isNull()) {
23: return;
24: }
25:
26: txt_file.setName(filename);
27: txt_file.open(IO_WriteOnly);
28:
29: txt_stream.setDevice(&txt_file);
30:
31: txt_stream << texto->text();
32:
33: txt_file.flush();
34: txt_file.close();
35: }
36:
37: void Editor::fecharJanela()
38: {
39: this->close();
40: }
```

Incluimos dois novoc cabeçalhos:

4: #include <qfile.h>
5: #include <qtextstream.h>

E duas variáveis em *gravarArquivo()*:

17: QFile txt_file;
18: QTextStream txt_stream;

Também tiramos aqueles *qWarning()* e trocamos por uma funcionalidade mais lógica: quando clicar em cancelar, ele cancela (return):

22: if (filename.isNull()) {
23: return;
24: }

A partir daí começa o tratamento de arquivo:

26: txt_file.setName(filename);
27: txt_file.open(IO_WriteOnly);

Aqui informamos o nome do arquivo (linha 26) e abrimos com permissão apenas para escrita (linha 27). Não haveria problema algum em abrir com permissão de leitura e escrita. Desde que tenha escrita, está tudo bem.

Agora, anexamos o arquivo que abrimos ao QTextStream:

29: txt_stream.setDevice(&txt_file);

E jogamos o conteúdo do texto (o nosso QMultiLineEdit) para o arquivo, através do QTextStream txt_stream:

31: txt_stream << texto->text();

Depois disso, basta fechar o arquivo (acrescentei um *flush()* apenas por acrescentar. Não costuma ser necessário, mas faz parte de alguns livros didáticos):

33: txt_file.flush();
34: txt_file.close();

Pronto! Terminamos o que foi proposto. Obviamente o nosso editor não abre arquivo, não confere se o arquivo já existe, mas isso você pode facilmente descobrir lendo a documentação da Qt se baseando no que viu nesses 5 artigos sobre a Qt. Mas para encerrar, compile primeiro e execute para testar.

Para mais informações veja o Código do editor que é referido nessa quinta parte na categoria "Programação", em Código do editor.

Esses artigos foram baseados na versão 2.2 da Qt. Porém, tudo o que vimos nesses artigos servem também para a QT 3.X.

Provavelmente alguns se questionarão: "Por que usar esse tal QFile se eu resolvo bem o problema com C ou C++?". A Qt é multiplataforma. Você pode compilar um mesmo código para vários UNIX que supor-

tem X (incluindo o Linux), Windows e Mac OS X (segue abaixo uma tela do nosso editor compilado em Windows usando a Qt 3.0.4). O uso dos componentes da Qt ajuda você a se desvincular do sistema operacional caso deseje portar a sua aplicação.

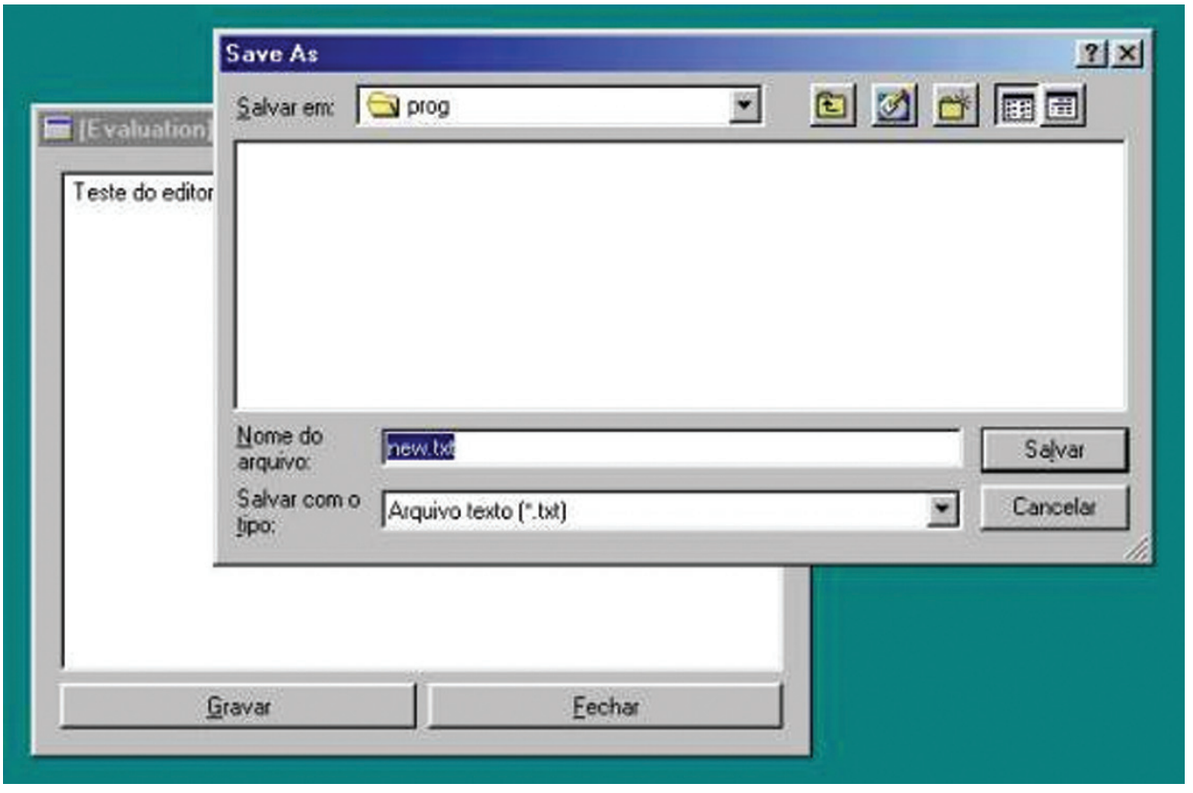

Figura 2