Expad, Editor Multifile dengan Qt

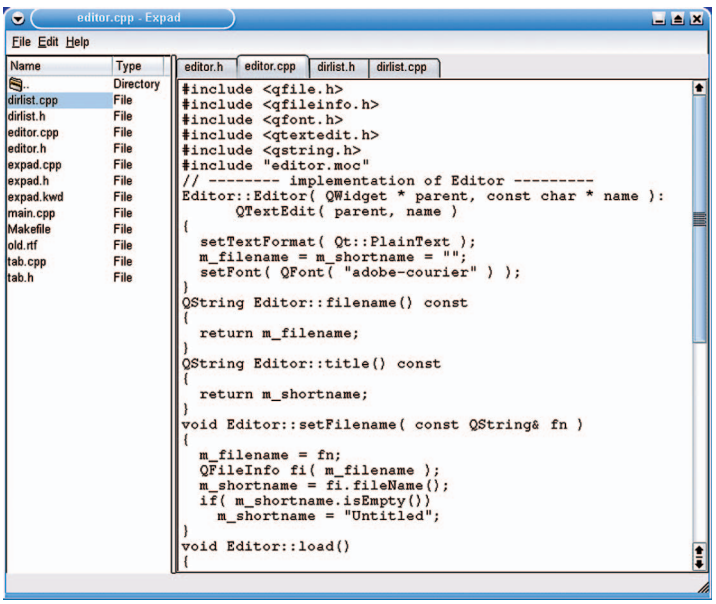
Bagian 1 dari 3 Artikel

Kalau Anda telah sedikit mengenal dasar pemrograman aplikasi grafis berbasis Qt, maka mencoba untuk mengembangkan aplikasi sederhana adalah suatu latihan yang praktis.

Qt adalah sebuah *toolkit* dari Trolltech AS (Norwegia) yang digunakan sebagai kerangka aplikasi yang menggunakan antarmuka grafis (GUI). Penggunaan Qt untuk membangun sebuah program akan banyak menolong sang programer. Selain kaya akan berbagai rutin untuk menyusun antarmuka grafis (GUI), Qt juga menyertakan beragam fungsi-fungsi untuk operasi file, *network*, struktur data, XML, basis data, dan lain sebagainya. Salah satu proyek *open source* yang cukup besar, yaitu KDE, juga mengandalkan Qt sebagai fondasi dasarnya.

Tulisan ini mengasumsikan bahwa pembaca telah memahami konsep pemrograman aplikasi dengan Qt. Jika tidak, Anda bisa menyimak kembali *InfoLINUX* No. 7/2001 hingga 10/2001 yang memuat tutorial bersambung tentang dasar penggunaan Qt. Referensi dan tutorial *online* yang dapat dibaca di *http://doc.trolltech.com* juga akan sangat menolong.

Tampilan Expad yang sudah jadi



Yang akan dijelaskan dalam tutorial ini adalah sebuah studi kasus pengembangan aplikasi contoh yang cukup berguna dalam kehidupan nyata, yaitu sebuah editor multifile bernama **Expad**.

Expad

Sebuah editor teks dikatakan *multifile* jika dapat menyunting beberapa file sekaligus. Lazimnya kemampuan seperti ini dimiliki oleh aplikasi-aplikasi office, misalnya StarOffice atau Microsoft Office (di Windows). Yang akan diulas di sini tentu saja aplikasi yang lebih sederhana karena hanya bertugas mengedit file teks biasa saja, namun punya keunggulan menampung sejumlah file pada satu window aplikasi.

Mula-mula marilah kita definisikan spesifikasi dari editor yang akan dinamakan Expad ini. Syarat utama tentu bahwa editor dapat menampilkan dan menangani file-file teks sekaligus, dengan jumlah yang tidak dibatasi. Untuk mudahnya, sekian banyak file teks ini tidak dipisah-pisah dengan menggunakan window editor sendiri-sendiri. Akan lebih indah jikalau digunakan tab untuk memilih file yang akan diaktifkan. Kira-kira hal demikian serupa dengan fitur *tab browsing* yang ada di browser web. Sebut saja fasilitas Expad ini sebagai *tab editing*.

Untuk memilih-milih file yang akan diedit, Expad diharapkan dapat menampilkan struktur direktori pada sebuah panel tersendiri. Selain mendaftar direktori, panel ini juga menyajikan daftar file-file pada direktori yang aktif. Dengan panel ini, user bisa berpindah direktori dengan mudah dan juga dapat memilih file yang ingin disunting. Jadi begitu menyorot file tertentu dan mengkliknya, maka file ini akan dimuatkan ke editornya dan siap dimodifikasi.

Dari deskripsi sederhana tentang fungsionalitas Expad ini mestinya sudah dapat disketsakan secara kasar apa-apa saja yang perlu diimplementasikan. Yang jelas dibutuhkan adalah window utama aplikasi harus selalu ada dan ini diturunkan dari widget *QMainWindow*. Untuk kasus ini, window utama ini akan disebut sebagai *Expad*.

Dua elemen lain yang akan menjadi bawahan dari Expad adalah editor teks dengan tab dan panel yang menyajikan daftar direktori dan file. Yang pertama akan berupa widget *Tab* dan yang kedua adalah *DirList*. Widget Tab berasal dari widget *QTabWidget* yang sudah disediakan Qt yang memang khusus dipergunakan kalau ingin memanfaatkan tab-tab, lumrahnya dijumpai pada

pembuatan kotak dialog yang kompleks. Sementara itu DirList mewarisi widget QListView, dimodifikasi untuk bisa mendaftar direktori dan file.

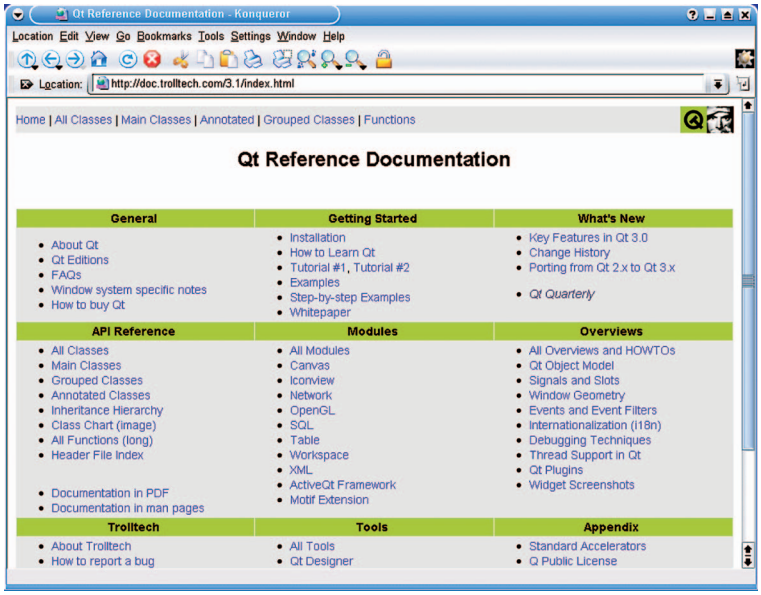
Melongok dokumentasi Qt, telah disediakan widget QTextEdit yang akan berfungsi sebagai ruang untuk menyunting teks. Sayangnya, QTextEdit belum dilengkapi dengan fasilitas untuk menyimpan teks ke file atau mengambil teks dari file. Oleh karenanya, akan dibangun kelas baru bernama *Editor* yang tidak lain adalah QTextEdit dengan tambahan kemampuan beroperasi dengan file.

Satu per satu dari kelas-kelas yang telah disebutkan akan dibahas di bawah ini.

Editor

Kelas Editor mewakili widget utama tempat user mengedit isi dari file. Kelas Editor ini hanya merupakan modifikasi kecil dari widget QTextEdit yaitu dengan penambahan fungsi untuk memudahkan menyimpan dan mengambil isi dari file teks yang sedang disunting. Terdapat juga fungsi *filename()* yang akan memberikan nama file yang bersesuaian dengan Editor tersebut. Bila ini adalah widget Editor yang baru dibuat dan isinya belum pernah disimpan ke file, *filename()* akan mengembalikan string kosong. Berikut adalah isi dari file *editor.h*.

```
#ifndef __EDITOR_H
#define __EDITOR_H

#include <qtextedit.h>
#include <qstring.h>

class QWidget;

class Editor: public QTextEdit
{
   Q_OBJECT

 public:
   Editor( QWidget * parent = 0, const char * name = 0 );
   void load();
   void save();
   QString filename() const;
   QString title() const;
   void setFilename( const QString& fn );

 private:
   QString m_filename;
   QString m_shortname;
};
#endif
```



Dokumentasi Qt di situs web Trolltech

Penyunting teks lumrahnya bekerja dengan font yang sifatnya fixed-width, yakni yang lebar per karakternya sama semua, tidak bergantung dari karakter itu sendiri. Jadi, baik huruf *m* maupun huruf *i* akan menyita ruang yang sama. Font semacam *Courier* tergolong font yang seperti ini. Karenanya, dari awal widget Editor mengatur agar font yang digunakan adalah font *adobe-courier,* yakni salah satu ragam font Courier yang biasanya sudah terinstalasi di sistem Linux. Umpamanya Anda ternyata tidak punya font ini, silakan ganti dengan yang tersedia, misalnya *bitstream-courier* atau malah *Courier New* jika Anda memasang font-font yang TrueType.

Implementasi kelas Editor akan diletakkan pada file *editor.cpp* berikut ini:

```
#include <qfile.h>
#include <qfileinfo.h>
#include <qfont.h>
#include <qtextedit.h>
#include <qstring.h>

#include "editor.moc"

// — implementation of Editor —

Editor::Editor( QWidget * parent, const char * name ):
    QTextEdit( parent, name )
{
 setTextFormat( Qt::PlainText );
 m_filename = m_shortname = "";
 setFont( QFont( "adobe-courier" ) );
}

QString Editor::filename() const
```

```
{
 return m_filename;
}

QString Editor::title() const
{
 return m_shortname;
}

void Editor::setFilename( const QString& fn )
{
 m_filename = fn;
 QFileInfo fi( m_filename );
 m_shortname = fi.fileName();
 if( m_shortname.isEmpty())
  m_shortname = "Untitled";
}

void Editor::load()
{
 if( filename().isEmpty() ) return;

 QFile f( filename() );

 if( f.open( IO_ReadOnly ) )
   {
    clear();
    QTextStream t( &f );
    while ( !t.eof() )
    {
     QString s = t.readLine();
     append( s );
    }
    f.close();
    repaint();
   }
}

void Editor::save()
{
 if( filename().isEmpty() ) return;

 QFile f( filename() );

 if ( f.open( IO_WriteOnly ) )
   {
    QTextStream t( &f );
    t << text();
    f.close();
   }
}
```

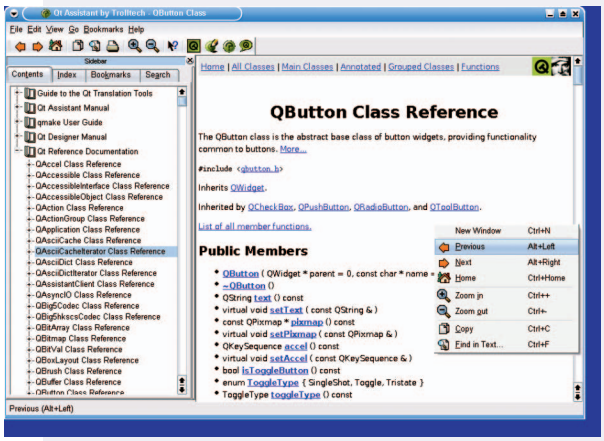
DirListItem dan DirList

Kelas DirListItem akan merepresentasikan informasi mengenai satu item yang ditampilkan di widget DirList, baik untuk direktori maupun untuk file. Sebagai kelas pembantu untuk widget DirList, DirListItem akan memberikan keterangan dalam bentuk dua kolom: yang pertama untuk nama file atau direktori dan yang kedua untuk jenisnya, "Directory" ataukah "File". Guna membedakan antara file dan direktori, kelas DirListItem akan mengatur agar sebuah pixmap kecil bergambar folder dibubuhkan di bagian kiri jika item tersebut merupakan sebuah direktori.

Widget DirList merupakan panel yang mendaftar isi sebuah direktori tertentu. Hal ini sendiri dilakukan dengan menggunakan QFileInfoList sebagaimana bisa dilihat di fungsi *DirList::setDir()*. Fungsi QDir::entryInfoList sendiri akan mengembalikan item-item yang dimiliki oleh direktori yang dimaksud dan kemudian bisa ditelusuri menggunakan iterator bernama QFileInfoListIterator. Dari hasil penjelajahan iterasi ini akan dibentuk sejumlah DirListItem yang mewakili masing-masing item yang ditemukan (quiz untuk pembaca: mengapa entri "." diabaikan?). Perhatikan bahwa karena membuat item baru QListView (dan juga DirList) akan menyebabkan item tersebut ditambahkan di akhir daftar item yang ada sehingga item yang dibuat pertama akan tampil paling bawah. Karenanya, penelusuran iterator harus dilakukan dari akhir ke awal agar nantinya justru item yang harusnya urutannya awal tetap tampil paling atas.

Baik kelas DirListItem maupun DirList dideklarasikan dalam file dirlist.h berikut ini:

Qt Assistant



Salah satu utiliti yang cukup berharga yang disertakan bersama Qt adalah Qt Assistant. Juga masih buatan Trolltech, Qt Assistant dapat berperan sebagai penjelajah dokumentasi, referensi, dan tutorial Qt. Memang semua dokumentasi Qt disimpan dalam format HTML sehingga bisa dibaca dengan menggunakan *browser* web biasa. Akan tetapi, Qt Assistant yang sangat mudah digunakan ini menawarkan ekstra fitur, seperti fasilitas *print*, indeks terstruktur, *full-text search*, dan masih banyak lagi.

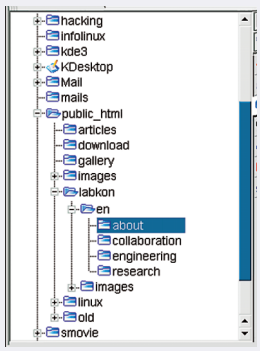
```
#ifndef __DIRLIST_H
#define __DIRLIST_H

#include <qmainwindow.h>
#include <qaction.h>
#include <qwidget.h>
#include <qlist.h>
#include <qstring.h>
#include <qmultilineedit.h>
#include <qtabwidget.h>
#include <qprinter.h>
#include <qsplitter.h>
#include <qlistview.h>

class DirListItem: public QListViewItem
{
  protected:
    bool m_isDir;
    bool m_isReadable;
    QString m_name;

  public:
    DirListItem( QListView*, const QString&, bool, bool );
    QString text( int ) const;
    bool isDir(){ return m_isDir; }
    bool isReadable(){ return m_isReadable; }
    QString shortName(){ return m_name; }
};

class DirList: public QListView
{
    Q_OBJECT

  public:
    DirList( QWidget* parent=0, const char* name=0 );
    void setDir( const QString& );
    QString dir(){ return m_dir; }

  protected slots:
    void slotDoubleClicked( QListViewItem* item );

  signals:
    void selected( const QString& filename );

  protected:
    QString m_dir;

};

#endif
```

Penting pula untuk diterangkan bagaimana DirList menangani klik ganda dari user ketika menyoroti sebuah item, yang diimpelemen-tasikan dalam *slotDoubleCllicked()*. Fungsi yang berupa slot ini dikoneksikan dengan signal bernama *doubleClick()*. Bila item yang diklik adalah sebuah file, maka DirList akan membangkitkan signal bernama *selected(const QString&)*. Kelak signal ini akan dihubungkan dengan slot yang tepat sehingga menghasilkan aksi memuat file yang diklik ke editor. Sementara itu kejadiannya akan berbeda jika item yang diklik adalah direktori. DirList segera beralih ke direktori yang dimaksud, tentu apabila pengaturan *permission* mengizinkan user masuk ke dalamnya. Dalam kasus direktori ini, tidak perlu ada signal yang dipicu.

File dirlist.cpp di bawah ini mengimplemen-tasikan hal-hal yang telah dikemukakan di atas.

```
#include "dirlist.moc"

#include <qdir.h>
#include <qfile.h>
#include <qfileinfo.h>
#include <qlistview.h>
#include <qstring.h>

#include "editor.h"

static const char* icon_folder_xpm[]={
"16 16 17 1",
" c None",
".  c #020202",
"+ c #A1741D",
"@ c #CC943A",
"# c #8D6819",
"$ c #C5C2BE",
"% c #2B2B2B",
"& c #E2DCD5",
"* c #484848",
"= c #ACA69C",
"-  c #737373",
";  c #7A5A12",
"> c #6A6A6A",
",  c #EEEEEE",
"'  c #CEA975",
")  c #563A06",
"!  c #AD6A0E",
"     .*          ",
"   .$,**%        ",
"  *#)-=$,*.*-.   ",
"  %&,'#)-=$,,,.  ",
"  %'@'&,'+)-=$,*",
"  %@!@'''&,'#)-*",
"  %@++!!@@'$&,'*",
"  %+;###+!!@@'&*",
"  *,,=#;##++!!'*",
```

List vs Tree



Implementasikan DirList yang dibahas di tutorial ini hanya menghasilkan daftar direktori dan file yang sederhana. Jika Anda bandingkan dengan struktur direktori yang ditampilkan di *file manager* seperti Konqueror, tentu saja yang demikian terlihat sederhana sekali. Akan tetapi, sesungguhnya membuat *tree* tidaklah susah. Jika Anda telah mengerti dan memahmi kelas DirList dengan baik, tidaklah sulit untuk mengembangkannya untuk menghasilkan *directory tree*, tidak lain karena QListView sebagai kelas induk dari DirList telah dilengkapi dengan kemampuan tersebut. Untuk lebih jelasnya, bacalah dokumentasi kelas QListView dan QListViewItem.

```
"  *,,,,&=#;##+@*",
"  *,,,&&&&$=#;#*",
"  %$&&&&&$$$$$>*",
"   .%>$$$$$$$=>*",
"     .%>=$$==>*",
"        .%>===>*",
"           .%% "};

// — implementation of DirListItem —

DirListItem::DirListItem( QListView* parent, const
QString& name,
        bool dir, bool readable ):
 QListViewItem( parent, name )
{
 m_name = name;
 m_isDir = dir;
 m_isReadable = readable;

 if( m_isDir ) setPixmap( 0, icon_folder_xpm );

}

QString DirListItem::text( int column ) const
{
 if( column == 0 ) return m_name;
 else if( column == 1 ) return m_isDir ? "Directory" :
"File";
 else return QString::null;
}

// — implementation of DirList —

DirList::DirList( QWidget* parent, const char* name ):
 QListView( parent, name )
{
 m_dir = "";
 addColumn( "Name" );
 addColumn( "Type" );

 setDir( "/" );
 setMinimumSize( 100, 50 );
 setSorting( -1 );

 connect( this, SIGNAL( doubleClicked(
QListViewItem* ) ),
    this, SLOT( slotDoubleClicked( QListViewItem* ) ) );
}

void DirList::setDir( const QString& dir )
{
 QDir d( dir );

 if( !d.exists() ) return;

 m_dir = dir;

 d.setSorting( QDir::Name | QDir::DirsFirst |
QDir::IgnoreCase );
 const QFileInfoList *list = d.entryInfoList();
 QFileInfoListIterator it( *list );
 QFileInfo *fi;

 clear();
 it.toLast();

 while ( ( fi = it.current() ) )
   {
    if( fi->fileName() != "." )
     {
      QListViewItem* item;
     item = new DirListItem( this, fi->fileName(),
       fi->isDir(), fi->isReadable() );
     }
    --it;
   }
}

void DirList::slotDoubleClicked( QListViewItem* i )
{
 DirListItem* item = (DirListItem*) i;
 if( item )
  if( item->isReadable() )
   {
    QString fullName = m_dir + "/" + item-
>shortName();
    if( item->isDir() ) setDir( fullName );
    else emit selected( fullName );
   }
}
```

Yang menarik untuk disimak adalah array yang bernama *icon_folder_xpm*. Sebenarnya ini adalah ikon yang akan digunakan untuk mencirikan sebuah direktori, untuk membedakanya dengan file biasa. Lihat di konstruktor DirListItem, terdapat perintah *setPixmap* jika item tersebut merupakan direktori. Format ikon yang dipakai adalah X Pixmap (xpm). Anda bisa saja mengganti ikon tersebut sesuai dengan yang Anda minati. Jalankan Gimp, buka file ikon yang dikehendaki (bisa PNG atau GIF atau format lain yang didukung Gimp) kemudian simpan dalam format XPM). File xpm hasil dari Gimp tadi mestinya bisa dibuka dengan editor teks biasa, kemudian salinkan saja untuk mengganti *icon_folder_xpm* di atas. Mudah bukan?

Ariya Hidayat *(ariya@infolinux.co.id)*

Edisi berikutnya:

Masih ada dua kelas yang akan diulas, yakni kelas Tab dan kelas Expad. Jangan lewatkan bagaimana kesemuanya akan dirangkai membentuk satu aplikasi utuh!

Expad, Editor Multifile dengan Qt

Bagian 2 dari 3 Artikel

Sebagaimana disinggung di bagian sebelumnya, Expad akan merupakan sebuah editor teks dengan fitur *tab editing*. Bagian kedua ini akan mengupas mengenai teknik implementasi fitur tersebut.

Apakah susah mewujudkan sebuah sistem penyunting teks dengan kemampuan *tab editing*? Ternyata tidak. Triknya adalah dengan menampung si editor (lihat kelas *Editor* di bagian pertama) dalam tab-tab sendiri. Qt telah menyediakan *widget* bernama QTabWidget. Untuk membubuhkan sedikit perluasan kemampuannya, QTabWidget akan dimodifikasi menjadi kelas baru yang bernama Tab.

Tab

Kelas *Tab* dideklarasikan pada file *tab.h* di bawah ini:

```
#ifndef __TAB_H
#define __TAB_H

#include <qtabwidget.h>

class Editor;

class Tab: public QTabWidget
{
   Q_OBJECT

  public:
   Tab( QWidget * parent = 0, const char * name = 0, WFlags f = 0 ):
     QTabWidget( parent, name, f ) {};

   int count();

   void addEditor( Editor* e, const QString& t );
   Editor* currentEditor();
   void removeEditor( Editor* e );
};
#endif
```

Melihat definisi widget *Tab* di file header tab.h jelaslah bahwa widget ini persis seperti induknya, *QTabWidget*, dengan hanya fungsi-fungsi ekstra untuk menambahkan sebuah widget Editor baru menjadi salah satu tabnya atau menghapus tab yang sudah ada. Tab-tab di QTabWidget biasa disebut *page* dan inilah yang akan di-*cast* ke widget Editor. Sederhana saja, bukan?

Implementasi kelas Tab cukup pendek dan dituangkan dalam file *tab.cpp* sebagaimana ditunjukkan di bawah ini:

```
#include "tab.moc"
#include "editor.h"

#include <qtabwidget.h>
#include <qtabbar.h>

// — implementation of Tab —-

int Tab::count()
{
 return tabBar()->count();
}

void Tab::addEditor( Editor* e, const QString& t )
{
 if( !e ) return;
 addTab( e, t );
 setCurrentPage( count() - 1 );
 e->show();
 show();
}

void Tab::removeEditor( Editor* e )
{
 if( !e ) return;
 removePage( e );
 delete e;
 show();
}

Editor* Tab::currentEditor()
{
 return dynamic_cast<Editor*>(
currentPage() );
}
```

Fungsi anggota *addEditor* dan *removeEditor*, sesuai namanya, akan menyisipkan sebuah editor baru atau menghapus editor yang dimaksud. Tentu saja, editor tersebut harus dibuat terlebih dahulu karena bukan merupakan tanggung jawab si kelas *Tab* ini.

Karena user dapat beralih-alih tab, maka sebuah fungsi tambahan *current-Editor* diperlukan untuk mengetahui editor mana yang kini tengah aktif.

Window utama: Expad

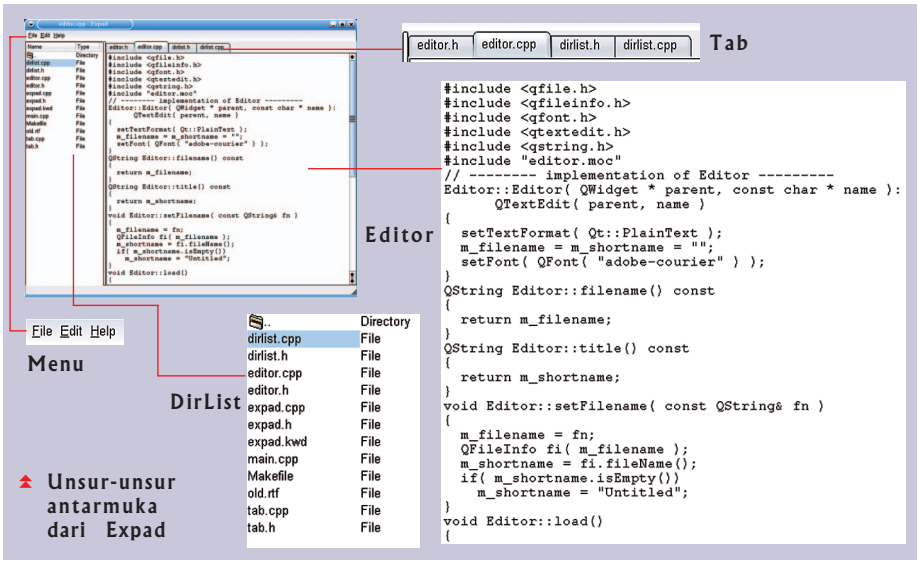
Fungsionalitas paling berat ada pada kelas *Expad* sebagai widget yang merupakan window utama. Expad punya sejumlah slot yang berfungsi melakukan operasi file, penyuntingan, dan lain-lain. Lebih jelasnya, lihat file header *expad.h* berikut ini.

```
#ifndef __EXPAD_H
#define __EXPAD_H
```

dynamic_cast ?

Ya, *dynamic_cast* adalah proses *casting* yang murni dialek C++. Anda tidak akan menemukannya di program C biasa. Berbeda dengan casting ala C yang hanya memaksakan satu tipe variabel ke tipe lainnya, dynamic_cast juga mengikutsertakan faktor mungkin tidaknya variabel tersebut di-casting. Contohnya dalam kasus untuk kelas Tab, variabel yang tipenya QWidget bisa di-casting ke tipe Editor, asal variabel tersebut memang merupakan sebuah Editor. Ini hanya mungkin karena Editor adalah turunan dari QTextEdit yang juga turunan dari QWidget.

```
#include <qmainwindow.h>

class Tab;
class DirList;
class QAction;
class QWidget;
class QSplitter;

class Expad: public QMainWindow
{
   Q_OBJECT

public:
   Expad();

protected:

   void initActions();
   void initMenu();

private slots:

   void fileNew();
   void fileOpen();
   void fileSave();
   void fileSaveAs();
   void fileClose();
   void fileCloseAll();
   void fileQuit();

   void editUndo();
   void editRedo();
   void editCut();
   void editCopy();
   void editPaste();
   void editSelectAll();

   void about();

   void tabChanged( QWidget* );
   void openFile( const QString& filename );

private:

   // main widgets
   QSplitter* splitter;
    Tab *tabs;
   DirList* dirlist;

   // actions
   QAction* ma_FileNew;
   QAction* ma_FileOpen;
   QAction* ma_FileSave;
   QAction* ma_FileSaveAs;
   QAction* ma_FileClose;
   QAction* ma_FileCloseAll;
   QAction* ma_FileQuit;

   QAction* ma_EditUndo;
   QAction* ma_EditRedo;
   QAction* ma_EditCut;
   QAction* ma_EditCopy;
   QAction* ma_EditPaste;
   QAction* ma_EditSelectAll;

   QAction* ma_About;
};

#endif
```



Begitu banyak *QAction* yang ada di Expad mencerminkan pendekatannya yang sedikit berbeda soal penanganan input dari user. Sebuah QAction sendiri adalah abstraksi dari suatu tindakan tertentu yang berasal dari user. Contohnya, *ma_FileNew* adalah action untuk membuat dokumen baru. Selain melalui menu *File, New*, user dapat juga memicu action ini menggunakan keyboard dengan shortcut tertentu, dalam contoh ini adalah *Ctrl+N*. Keuntungannya adalah bahwa satu action saja bisa digunakan sebagai representasi item di menu program dan sebagai tombol shortcut sehingga terasa cukup mudah dan praktis.

Implementasi kelas Expad diletakkan pada file tersendiri bernama *expad.cpp*. Karena cukup panjang, maka isi file tersebut akan dibahas secara tahap demi tahap sehingga tiap bagian pentingnya dapat teranotasi dengan jelas.

Mula-mula, simaklah prolog file expad.cpp berikut konstruktor Expad berikut:

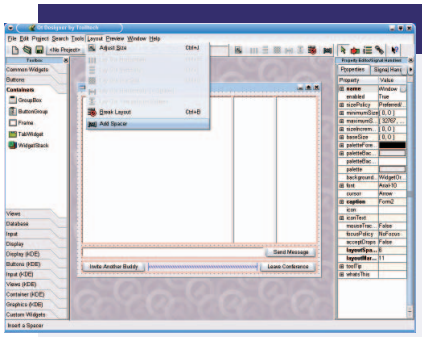
```
#include "expad.moc"
#include "editor.h"
#include "dirlist.h"
#include "tab.h"

#include <qaccel.h>
#include <qaction.h>
#include <qfiledialog.h>
#include <qkeycode.h>
#include <qmenubar.h>
#include <qmessagebox.h>
#include <qpopupmenu.h>
#include <qsplitter.h>
#include <qstatusbar.h>

// — implementation of Expad —-

const char* app = "Expad";

Expad::Expad()
   : QMainWindow( 0, "Expad",
WDestructiveClose )
{
  initActions();
  initMenu();

  splitter = new QSplitter( this );
  setCentralWidget( splitter );
```

```
  dirlist = new DirList( splitter);
  splitter->setResizeMode( dirlist,
QSplitter::KeepSize );
  connect( dirlist, SIGNAL( selected( const
QString& ) ),
     this, SLOT( openFile( const QString& ) ) );

  tabs = new Tab( splitter );
  connect( tabs, SIGNAL( currentChanged(
QWidget* ) ),
     this, SLOT( tabChanged( QWidget* ) ) );

  fileNew();

  statusBar()->message( "Ready", 2000 );
  resize( 450, 400 );
}
```

Tanggung jawab terbesar si konstruktor adalah menyusun *layout* dari tampilan program. Sebagaimana bisa dilihat pada gambar sebelumnya, ada dua elemen utama dalam satu window, yaitu panel direktori (*DirList*) dan editor (*Tab*). Sudah sewajarnya kedua elemen ini tampil saling bersisian, namun dengan sebuah pembatas (*splitter*) yang dapat digerakkan untuk mengatur pembagian ruangan bagi keduanya. Dengan Qt, hal demikian menjadi mudah berkat widget *QSplitter*. Dua widget lain yang ingin dipisahkan tinggal menjadikan si splitter sebagai *parent*-nya.

Sepintas, barangkali Anda bisa lihat bahwa urusan inisialisai tampilan lainnya dilakukan di fungsi *initActions* dan *initMenu* berikut:

```
void Expad::initActions()
{
  ma_FileNew = new QAction( this );
  ma_FileNew->setText( "New" );
  ma_FileNew->setAccel( CTRL + Key_N );
  connect( ma_FileNew, SIGNAL( activated() ),
     this, SLOT( fileNew() ) );

  ma_FileOpen = new QAction( this );
  ma_FileOpen->setText( "Open" );
  ma_FileOpen->setAccel( CTRL + Key_O );
  connect( ma_FileOpen, SIGNAL( activated() ),
     this, SLOT( fileOpen() ) );

  ma_FileSave = new QAction( this );
  ma_FileSave->setText( "Save" );
  ma_FileSave->setAccel( CTRL + Key_S );
  connect( ma_FileSave, SIGNAL( activated() ),
     this, SLOT( fileSave() ) );
ma_FileSaveAs = new QAction( this );
  ma_FileSaveAs->setText( "Save As" );
  connect( ma_FileSaveAs, SIGNAL(
activated() ),
     this, SLOT( fileSaveAs() ) );

  ma_FileClose = new QAction( this );
  ma_FileClose->setText( "Close" );
  ma_FileClose->setAccel( CTRL + Key_W );
  connect( ma_FileClose, SIGNAL( acti
vated() ),
     this, SLOT( fileClose() ) );

  ma_FileCloseAll = new QAction( this );
  ma_FileCloseAll->setText( "Close All" );
  connect( ma_FileCloseAll, SIGNAL(
activated() ),
     this, SLOT( fileCloseAll() ) );

  ma_FileQuit = new QAction( this );
  ma_FileQuit->setText( "Quit" );
  ma_FileQuit->setAccel( CTRL + Key_X );
  connect( ma_FileQuit, SIGNAL( activated() ),
     this, SLOT( fileQuit() ) );

  ma_EditUndo = new QAction( this );
  ma_EditUndo->setText( "Undo");
  ma_EditUndo->setAccel( CTRL + Key_Z );
  connect( ma_EditUndo, SIGNAL( activated() ),
     this, SLOT( editUndo() ) );

  ma_EditRedo = new QAction( this );
  ma_EditRedo->setText( "Redo");
  connect( ma_EditRedo, SIGNAL( activated() ),
     this, SLOT( editRedo() ) );

  ma_EditCut = new QAction( this );
  ma_EditCut->setText( "Cut");
  ma_EditCut->setAccel( CTRL + Key_X );
  connect( ma_EditCut, SIGNAL( activated() ),
     this, SLOT( editCut() ) );

  ma_EditCopy = new QAction( this );
  ma_EditCopy->setText( "Copy");
  ma_EditCopy->setAccel( CTRL + Key_C );
  connect( ma_EditCopy, SIGNAL( activated() ),
     this, SLOT( editCopy() ) );

  ma_EditPaste = new QAction( this );
  ma_EditPaste->setText( "Paste");
  ma_EditPaste->setAccel( CTRL + Key_V );
  connect( ma_EditPaste, SIGNAL( acti
vated() ),
     this, SLOT( editPaste() ) );

  ma_EditSelectAll = new QAction( this );
  ma_EditSelectAll->setText( "Select All");
  ma_EditSelectAll->setAccel( CTRL +
Key_A );
  connect( ma_EditSelectAll, SIGNAL(
activated() ),
     this, SLOT( editSelectAll() ) );

  ma_About = new QAction( this );
  ma_About->setText( "About...");
  connect( ma_About, SIGNAL( activated() ),
     this, SLOT( about() ) );
}

// initialize menu
void Expad::initMenu()
{
```



Manajemen layout

Jika Anda programer Java, pastilah tidak asing dengan teknik Qt dalam melakukan pengaturan *layout*. Tetapi, bila Anda lebih familiar dengan kakas pengembangan di Windows seperti Borland Delphi atau Microsoft Visual Basic, layout bisa jadi membuat kepala pening. Memang, widget-widget di Qt tidak memiliki posisi yang ditentukan oleh lokasi pikselnya. Masing-masing akan menempatkan dirinya sendiri sesuai tempat yang disediakan, dalam hal ini menjadi urusan si pengatur layout. Ada beberapa pilihan layout, seperti vertikal, horizontal, ataupun grid. Anda bahkan bisa memasang *nested layout* atau layout dalam layout. Cara termudah untuk mendesain layout yang tepat adalah dengan menggunakan aplikasi yang cocok untuk kebutuhan ini, yaitu Qt Designer. Dengan modal *point-and-click*, layout serumit apapun jadi mudah. Coba dan rasakan sendiri!

```
  QPopupMenu *file_menu = new
QPopupMenu( this );
  menuBar()->insertItem( "&File", file_menu );
  ma_FileNew->addTo( file_menu );
  ma_FileOpen->addTo( file_menu );
  file_menu->insertSeparator();
  ma_FileSave->addTo( file_menu );
  ma_FileSaveAs->addTo( file_menu );
  file_menu->insertSeparator();
  ma_FileClose->addTo( file_menu );
  ma_FileCloseAll->addTo( file_menu );
  file_menu->insertSeparator();
  ma_FileQuit->addTo( file_menu );

  QPopupMenu *edit_menu = new
QPopupMenu( this );
  menuBar()->insertSeparator();
  menuBar()->insertItem( "&Edit", edit_menu );
  ma_EditUndo->addTo( edit_menu );
  ma_EditRedo->addTo( edit_menu );
  edit_menu->insertSeparator();
  ma_EditCut->addTo( edit_menu );
  ma_EditCopy->addTo( edit_menu );
  ma_EditPaste->addTo( edit_menu );
  edit_menu->insertSeparator();
  ma_EditSelectAll->addTo( edit_menu );

  QPopupMenu *help_menu = new
QPopupMenu( this );
  menuBar()->insertSeparator();
  menuBar()->insertItem( "&Help",
help_menu );
  ma_About->addTo( help_menu );
}
```

Dalam fungsi *Expad::initMenu()* , sejumlah action yang diciptakan di *Expad::initActions()* disisipkan ke menu utama dan ke toolbar (jika memang ingin ada toolbar) hanya dengan memanggil fungsi *addTo* dari masing-masing action. Anda tidak perlu lagi membuat item menu khusus sebagaimana yang sudah-sudah (lihat lagi misalnya program *QSimpleApp* di *InfoLINUX* No. 8/2001). Selain menyederhanakan, teknik ini akan mengurangi duplikasi kode program yang tidak perlu.

Pada slot-slot yang mengurusi file, yakni *fileNew(), fileOpen(), fileSave(), fileSaveAs(), fileClose()*, serta *fileCloseAll()*, sebagian besar hanya bertindak sebagai jembatan untuk memanggil fungsi-fungsi yang tepat dari Editor ataupun Tab.

Perhatikan fungsi *Expad::openFile()*. Dengan memberikan argumen berupa nama file yang ingin dibuka, fungsi tersebut akan mendelegasikan tugas pembukaan file tersebut ke *Editor::load()*. Barangkali Anda perlu melihat ulang bagian pertama tulisan ini untuk mencermati implementasi *Editor::load()*.

```
// opens the given file as new document
void Expad::openFile( const QString& filename )
{
  if( !filename.isEmpty() )
    {
     Editor* editor = new Editor( tabs );
     editor->setFilename( filename );
     editor->load();
     tabs->addEditor( editor, editor->title() );
    }
}
```

Fungsi *Expad::fileNew()* lebih mudah lagi. Cukup buat saja editor baru dan sisipkan ke dalam kumpulan tab yang sudah ada.

```
// creates a new blank document
void Expad::fileNew()
{
  Editor* editor = new Editor( tabs );
  editor->setFilename( QString::null );
  tabs->addEditor( editor, editor->title() );
}
```

Beberapa fungsi lain untuk urusan file merupakan slot. Jadi bisa saja slot ini diaktifkan dari menu (via QAction yang tepat). Untuk itu, perlu juga ditanyakan kepada user nama file yang akan dibuka atau disimpan. Untunglah Qt menyediakan fasilitas kotak dialog yang sudah *built-in*, yaitu *QFileDialog*.

```
// opens and loads a document
void Expad::fileOpen()
{
  QString filename;
  filename  = QFileDialog::getOpenFileName(
dirlist->dir(),
              QString::null, this);
  openFile( filename );
}

// saves active document
void Expad::fileSave()
{
  Editor* editor = tabs->currentEditor();
  if( !editor) return;

  if( editor->filename().isEmpty() )
fileSaveAs();
  editor->save();
}

// saves document to a different name
void Expad::fileSaveAs()
{
  Editor* editor = tabs->currentEditor();
  if( !editor) return;

  QString filename;
  filename  = QFileDialog::getSaveFileName(
dirlist->dir(),
              QString::null, this);
  if( !filename.isEmpty() )
    {
     editor->setFilename( filename );
     editor->save();
     tabs->changeTab( editor, editor->title() );
    }
}

// closes active document
void Expad::fileClose()
{
  Editor* editor = tabs->currentEditor();
  if( !editor ) return;
  tabs->removeEditor( editor );
}

// closes all document
void Expad::fileCloseAll()
{
  while( tabs->currentEditor() )
    {
     Editor* editor = tabs->currentEditor();
     tabs->removeEditor( editor );
    }
}
```

Ariya Hidayat *(ariya@infolinux.co.id)*

Edisi berikutnya:

Bagian terakhir akan mengupas mengenai mekanisme kerja dari operasi penyuntingan seperti *copy, cut, paste*, sekaligus merangkaikan kesemua kelas yang diperlukan menjadi satu aplikasi utuh. Jangan lewatkan pula unjuk gigi kemampuan Qt dengan ilustrasi untuk menghasilkan versi Windows dari Expad.

Expad, Editor Multifile dengan Qt

Bagian 3 dari 3 Artikel

Tibalah saatnya merangkaikan kepingan-kepingan yang sebagian besar telah ditunjukkan pada dua bagian sebelumnya, demi menghasilkan sebuah aplikasi utuh, sang editor.

Terakhir sudah dibahas mengenai fungsi-fungsi dari kelas Expad yang berurusan dengan file. Kini, silakan simak fungsi-fungsi lainnya yang berkenaan dengan operasi penyuntingan, yang dilakukan dengan memanggil fasilitas yang disedikan kelas *Editor* (lihat bagian sebelumnya). Ikhwal mana editor yang aktif dideteksi dengan *currentEditor()* yang sudah disediakan oleh kelas *Tab*.

Fungsi-fungsi dasar untuk melakukan *cut*, *copy*, *paste* yang juga merupakan slot dari kelas Expad bisa dilihat di bawah ini:

```
// cuts selected text
void Expad::editCut()
{
  Editor *e = tabs->currentEditor();
  if( e ) e->cut();
}

// copies selected text
void Expad::editCopy()
{
  Editor *e = tabs->currentEditor();
  if( e ) e->copy();
}

// pastes from clipboard
void Expad::editPaste()
{
  Editor *e = tabs->currentEditor();
  if( e ) e->paste();
}
```

Zaman sekarang, editor teks yang tidak dilengkapi dengan fitur *undo* barangkali akan segera ditinggalkan penggunanya. Melihat tren ini, Expad juga tidak ketinggalan melengkapi diri dengan *undo*, sekaligus juga *redo*.

```
// reverts last action
void Expad::editUndo()
{
  Editor *e = tabs->currentEditor();
  if( e ) e->undo();
}

// redoes last action
void Expad::editRedo()
{
  Editor *e = tabs->currentEditor();
  if( e ) e->redo();
}
```

Masih demi kenyamanan *user*, fasilitas untuk mengaktifkan *selection* pada seluruh isi dokumen ditunjukkan oleh slot *selectAll* di bawah ini.

```
// selects all text
void Expad::editSelectAll()
{
  Editor *e = tabs->currentEditor();
  if( e ) e->selectAll();
}
```

Apa yang terjadi jika ada tab yang diklik oleh user? Ini menandakan bahwa user tersebut ingin berpindah menyunting teks di dokumen yang terasosiasi dengan tab. Melihat koneksi antara signal dan slot yang dilakukan di awal inisialisasi kelas Expad, jelas bahwa yang terjadi adalah pemicuan slot bernama tabChanged(). Apa yang harus dilakukan oleh slot ini? Cukup mengganti *caption* dari program yang akan tampil di *titlebar window*-nya. Dengan cara ini, tatkala file bernama linux.txt sedang aktif diedit, maka judul window aplikasi berubah menjadi "linux.txt - Expad". Cukup profesional, bukan?

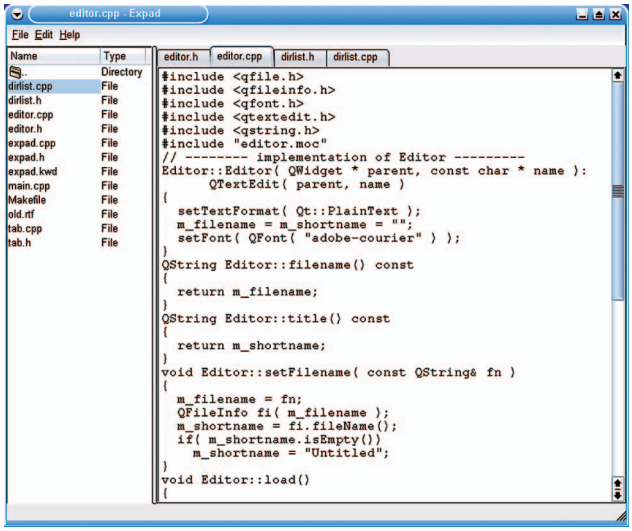
```
void Expad::tabChanged( QWidget* )
{
  Editor* editor = tabs->currentEditor();
  if( !editor ) return;
  setCaption( editor->title() + " - " + app );
}
```

Terakhir, barangkali menampilkan *about box* juga jadi hal yang wajib, baik untuk sekadar numpang nampang ataupun untuk memberikan informasi ringkas lainnya untuk si pemakai program. Memanfaatkan kelas *QMessageBox* yang telah disediakan oleh Qt, menongolkan kotak dialog *About* semacam ini bisa jadi hanya butuh beberapa baris.

```
// displays program information
void Expad::about()
{
  QMessageBox::about( this,
QString("About
%1").arg( app ),
    "<b>" + QString( app ) + "</b>" +
    "<p>Multiple-file text editor</p>"
    "<p>Programmed by Ariya
Hidayat<br>"
    "ariya@infolinux.co.id</p>" );
}
```

Perhatikan bagaimana digunakan tag-tag HTML seperti dan pada keterangan yang ditampilkan dalam kotak dialog tersebut. Hal ini ditujukan agar teksnya dicetak tebal (atau bold). Inilah salah keuntungan Qt, menyediakan dukungan *rich text* (sering disebut juga sebagai *formatted text*) yang dengan mudah dimanfaatkan hanya dengan menyisipkan *tag-tag* yang tepat.

Masih ada yang kurang? Ya, yaitu slot untuk *quit* alias keluar dari aplikasi. Sebuah editor yang baik akan menutup terlebih dahulu dokumen-dokumen yang tengah disunting, tidak tiba-tiba usai



Expad: in action

begitu saja. Berikut adalah implementasi slot tersebut, yang tentu saja akan cukup pendek.

```
// exits program
void Expad::fileQuit()
{
  fileCloseAll();
  close();
}
```

Kalaulah Anda masih kebingungan dengan beberapa kelas Qt yang dipergunakan, dokumentasi referensi Qt memberikan keterangan panjang lebar mengenai semuanya.Jangan segan-segan untuk senantiasa melongoknya!

main.cpp dan Makefile

Sebagaimana yang sudah-sudah, agar implementasi window utama *Expad* beserta elemen-elemennya dapat digunakan, maka harus dibuat program utama dalam sebuah file *main.cpp* seperti yang ditunjukkan berikut:

```
#include <qapplication.h>
#include "expad.h"

int main( int argc, char ** argv )
{
  QApplication a( argc, argv );

  QMainWindow * w = new Expad();
  w->show();

  a.connect( &a, SIGNAL(lastWindow
Closed()), &a, SLOT(quit()) );
  return a.exec();
}
```

Berikutnya, inilah Makefile yang digunakan untuk memudahkan proses *compile-and-link*. Mudah-mudahan Anda tidak jenuh karena Makefile ini pada pokoknya selalu saja dengan tutorial Qt yang dulu, dengan penyesuaian bagian TARGET, OBJECTS, dan MOCS.

```
QTDIR    = /usr/lib/qt3
QTLIB    = $(QTDIR)/lib
QTINCLUDE = $(QTDIR)/include
QTMOC    = $(QTDIR)/bin/moc

TARGET   = expad
OBJECTS  = main.o editor.o dirlist.o tab.o
expad.o
MOCS     = editor.moc dirlist.moc tab.moc
expad.moc

CC       = g++
CFLAGS   = -Wall -O2
LIBS     = -lqt-mt

$(TARGET): $(MOCS) $(OBJECTS)
  $(CC) $(LFLAGS) -o $(TARGET) $(OB
JECTS) $(LIBS) -L$(QTLIB) -I$(QTINCLUDE)

%.o: %.cpp
  $(CC) $(CFLAGS) -I$(QTINCLUDE) -c $< -
o $@

%.moc: %.h
  $(QTMOC) $< -o $@

clean:
  rm $(OBJECTS) $(TARGET) $(MOCS)
```

Begitu Anda sudah mendapatkan semua file *source code* yang dibutuhkan, yaitu *expad.h, expad.cpp, editor.h, editor.cpp, dirlist.h, dirlist.cpp, tab.h, tab.cpp, main.cpp,* dan *Makefile*, maka kompilasi bisa dilakukan dengan perintah:

```
make
```

Bila semuanya lancar dan tidak ada sesuatu yang salah, Anda akan mendapatkan file executable *expad* yang siap dijalankan dengan perintah.

```
./expad
```

Selamat karena baru saja Anda menghasilkan editor yang sudah lebih baik dari Notepad-nya Windows!

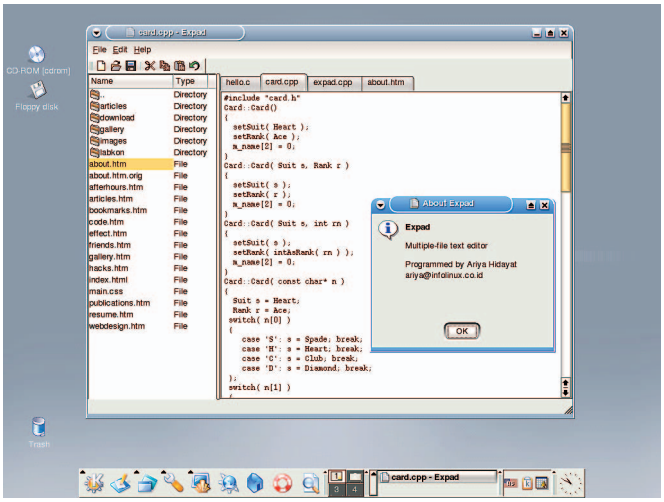
Berbagai penyempurnaan

Sejenak menggunakan dan mengamati kerja Expad ini, bisa disimpulkan bahwa spesifikasi seperti yang dibahas di awal

Gaya pemrograman

Bukan cuma seorang model yang bisa bergaya. Programer pun juga punya gaya, yakni gaya pemrograman. Bila Anda cermati, Qt menggunakan gaya yang mirip dengan Java. Sebuah *identifier* selalu dituliskan dengan huruf kecil, tidak menggunakan *underscore*, tetapi karakter pertama dari tiap kata adalah huruf kapital, kecuali karakter paling pertama sekali. Bisa dilihat dari nama-nama fungsi seperti *cut()* dan *copy()*. Untuk nama yang terdiri dari dua kata atau lebih, jadinya adalah seperti *selectAll()*, *setItemEnabled()*, dan seterusnya. Buat Anda yang familiar dengan Java, gaya semacam ini tidak akan asing lagi. Bahkan juga terhitung kode sumber KDE menggunakan gaya yang sama.

***Nah*, gaya seperti apakah yang harus Anda adopsi? Itu semua bebas, sesuai dengan selera dan barangkali karakter masing-masing. Hanya saja, penting untuk ditekankan mengenai konsistensi dari gaya tersebut. Kalau di satu saat Anda menuliskan *openFile()*, tetapi di tempat lain tiba-tiba menjadi *OpenFile()*, dan secara ajaib ganti lagi menjadi *open_file()*, maka Anda sendiri kelak yang akan dibingungkan oleh hal tersebut. Belum lagi jika Anda bekerja dalam sebuah tim yang terdiri atas beberapa orang. Untuk kasus ini, akan lebih baik jika konsistensi dan gaya yang dipilih ditekankan dari awal, demi menghindari bentrok atau konflik yang tidak beralasan. Selamat bergaya!**



Expad yang dilengkapi toolbar

tulisan ini sudah tercapai. Namun demikian, tentunya ada beberapa penyempurnaan yang masih bisa dilakukan. Anda dapat menganggapnya sebagai latihan kecil guna lebih mengenal seluk-beluk Qt.

Expad punya fasilitas *Close All*. Dengan teknik implementasi yang sama, mestinya juga tidak susah untuk mewujudkan fasilitas *Save All*. Anda tinggal melakukan iterasi tab-tab demi tab, menemukan editor yang berasosiasi dengan tab tersebut, dan memanggil fungsi yang tepat dari editor tersebut agar dokumennya tersimpan.

Sementara itu, bisa Anda saksikan bahwa font yang digunakan di widget Editor selalu sama, yakni adobe-courier. Pilihan yang lebih baik adalah menyiapkan kotak dialog konfigurasi yang salah satunya memungkinkan user mengganti font yang digunakan, bahkan juga tata warnanya. Anda bisa menyimak lagi dokumentasi QTextEdit untuk mendapatkan gambaran dukungan font dan pewarnaan yang telah disediakan.

Di sisi lain, DirList bisa mendapatkan sedikit perbaikan. Kalau untuk direktori ada ikon bergambar folder, mengapa tidak ada ikon serupa untuk item yang berupa file? Anda bahkan bisa membedakan ikon yang digunakan untuk file teks (berekstensi txt), file README, file gambar, dan jenis-jenis lainnya. Sebuah direktori yang tidak bisa dibaca (alias tidak *readable*) seharusnya juga mendapatkan ikon yang berbeda, misalnya gambar folder yang terkunci. Modal untuk ini sederhana: siapkan ikon yang tepat (lihat bagian 1 untuk mengetahui cara menghasilkan ikon XPM dengan GIMP), deteksi jenis file yang diproses (bisa menggunakan kelas QFileInfo yang juga sudah disediakan Qt), dan aktifkan ikon yang tepat untuk jenis file tertentu. Kelihatannya barangkali memang susah, tetapi mengikut contoh untuk Direktori vs File mestinya Anda bisa menurunkannya secara perlahan-lahan.

Modifikasi-modifikasi kecil di atas bisa dilakukan dengan hanya menambah beberapa baris program di sana-sini tanpa harus merombak strukturnya secara keseluruhan.

Toolbar

Hampir semua aplikasi modern dilengkapi dengan toolbar, tujuannya adalah menyediakan akses cepat bagi fungsi-fungsi yang sering digunakan. Expad yang dikupas hingga titik ini belum memiliki toolbar sama sekali. Pertanyaannya adalah: sulitkan membuat toolbar?

Mari kita sorot lagi pembuatan menu Expad. Yang dilakukan adalah menyisipkan masing-masing *action* yang sudah dibuat ke dalam menu. Lihat kembali fungsi Expad::initMenu() pada bagian ke-2 tulisan ini. Untuk toolbar, Anda cukup menyisipkan action tersebut ke dalam toolbar.

Sebagai latihan, beginilah langkah-langkah yang harus Anda lakukan. Mula-mula, siapkan fungsi *initToolbar()* yang analog dengan *initMenu()*. Jangan lupa juga untuk memanggilnya dari konstruktor si kelas Expad. Yang dilakukan oleh initToolbar() adalah membuat toolbar dari kelas QToolBar dan membubuhkan action yang diinginkan (misalnya *editCut* atau *editCopy* atau yang lainnya) dengan menggunakan *QAction::addTo()*. Hal ini serupa sekali dengan initMenu(). Kompilasi ulang Expad dan beres sudah.

Hingga tahap ini Anda akan mendapatkan toolbar, tepat di bawah menu, tetapi sialnya tanpa ikon sama sekali. Walaupun demikian, fungsinya tetap berjalan. Cobalah klik tombol yang tidak berikon itu dan Anda akan memicu *action* tersebut untuk beraksi.

Menambahkan ikon harus dilakukan dari pihak si action. QAction punya fungsi khusus setIconSet yang mengizinkan Anda menentukan ikon yang akan digunakan oleh action tersebut. Formatnya ikonnya masih sama, Anda bisa menggunakan XPM yang dapat dibuat dengan teknik yang sama juga.

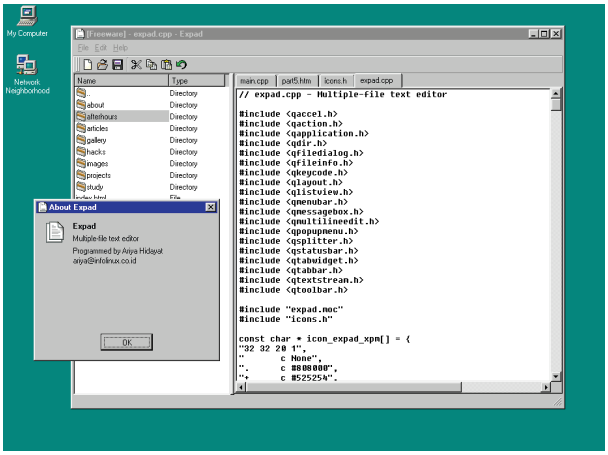
Demi kejelasannya, coba baca dan cermati dulu dokumentasi Qt untuk kelas QToolBar, QAction, QIconSet, dan QPixmap. Bila sukses, Anda akan mendapatkan Expad ber-toolbar sebagaimana bisa dilihat di *screenshot* yang ditunjukkan di halaman ini.

Expad for Windows?

Barangkali Anda pernah mendengar bahwa Qt disebut sebagai toolkit yang *multi-platform.* Maknanya adalah bahwa aplikasi-aplikasi yang dibuat sepenuhnya dengan kelas-kelas yang disediakan oleh Qt benar-benar bersifat lintas *platform*, bisa dikompilasi di sistem operasi yang didukung Qt, yaitu aneka varian Unix (termasuk Linux), Microsoft Windows, dan Mac OS. Karakter seperti ini menyebabkan source code program sama sekali tidak perlu diubah manakala dilakukan *porting* ke sistem operasi lain. Dengan demikian, Anda bisa mendesain program tertentu di Linux, membawa source code-nya ke Windows, kemudian melakukan kompilasi ulang dan menghasilkan program baru yang sudah for Windows.

Sayangnya, Qt versi Windows tidaklah diberikan secara cuma-cuma dan juga tidak open-source. Anda harus membayar untuk mendapatkan lisensi pengembangan aplikasi Windows dengan Qt. Berita baiknya adalah bahwa Trolltech—si pembuat Qt—membuat perkecualian bahwa Qt/Windows ini tersedia juga dalam edisi nonkomersialnya. Dengan menggunakan library Qt yang disediakan dalam edisi ini, Anda bisa menghasilkan aplikasi Windows sepanjang aplikasi tersebut juga sifatnya *free software.*

Memanfaatkan edisi nonkomersial atau evaluasi dari Qt/Windows (yang bisa di-*download* dari situs web *www.trolltech.com*), Expad juga dapat dipindahkan ke Windows. Karena *library* yang disediakan hanya cocok untuk Microsoft Visual C++, mau tidak mau Anda harus menggunakan Visual C++ jika ingin mengompilasi Expad di Windows. Dengan portabilitas Qt, sama sekali tidak ada file source code yang perlu diubah, cukup sesuaikan proses kompilasinya (menggunakan *Project* di Visual Studio). Hasilnya adalah Expad for Windows, berpenampilan dan berperilaku benar-benar serupa dengan versi Linuxnya.



Versi Windows dari Expad

Dari studi kasus pengembangan editor multifile ini, bisa Anda saksikan sendiri bagaimana kekayaan API dari Qt menjadi sangat menolong. Berbekal desain dan rancang bangun untuk aplikasi yang akan dikembangkan, pekerjaan selanjutnya nyaris seperti merangkaikan blok-blok siap pakai yang sudah disediakan Qt. Cepat dan nyaman. Kalau Anda sempat mengenal toolkit yang lain, misalnya GTK atau wxWindows, mengonstruksi editor seperti Expad (dengan fitur-fitur yang setara) akan menjadi berbeda dan susah untuk tidak melampaui batas 800 baris program. Begitu program selesai, dengan sifatnya yang portabel, Anda bisa mengkompilasinya di Windows dan/atau Mac untuk mendapatkan versi Windows dan/atau Mac dari program tersebut dengan mudah dan cepat.

Ariya Hidayat *(ariya@infolinux.co.id)*

TANYA JAWAB

Tanya: *Proses compile berjalan lancar. Namun, pada saat Expad dijalankan muncul pesan kesalahan "Segmentation Fault". Mengapa?*

Jawab: Penyebab paling utama biasanya adalah ketidaksinkronan antara file hasil MOC dengan file header. Solusi: jalankan *make clean* dahulu sebelum *make*.

Tanya: *Saat Expad dijalankan, saya ingin struktur direktori yang disajikan dimulai dari home directory, bukan dari root directory. Bagaimana mengubahnya?*

Jawab: Lihatlah konstruktor untuk kelas DirList. Perintah setDir("/") harus diubah, tidak lagi menunjuk ke / (root directory), tetapi ke *home directory.* Bagaimana mengambil home directory? Anda bisa menggunakan fungsi statik *QDir::homeDirPath()*. Lihat juga referensi Qt tentang kelas QDir untuk lebih jelasnya.

Tanya: *Bisakah saya mengedit file teks yang berukuran besar, misalnya beberapa MB, dengan Expad?*

Jawab: Berbeda dengan Notepad-nya Windows, widget QTextEdit bisa menangani teks sebanyak-banyaknya, hanya dibatasi oleh kapasitas memori yang ada. Karena Expad menggunakan widget ini, maka dengan sendirinya Expad akan sanggup mengurusi file yang ukurannya besar. Hanya saja kemungkinan Anda akan mengalami gangguan performa seperti waktu *loading* yang lambat, pergerakan kursor yang patah-patah, dan lain-lain.

Tanya: *Bagaimana membubuhkan kemampuan color syntax highlight seperti yang lazim dijumpai di editor yang canggih?*

Jawab: Untuk ini, Anda bisa memanfaat kelas QSyntaxHighlight (baru ada di Qt 3.1). Walaupun ini pekerjaan berat, namun bukan berarti mustahil untuk dilakukan. Anda juga dapat mengambil contoh dari program lain yang punya fasilitas serupa seperti Freddy (*http://freddy.sf.net*) atau Kate (*http://kate.sf.net*).

Tanya: *Apakah source code untuk Expad juga diedit dengan Expad?*

Jawab: Tentu saja tidak. Hal ini serupa dengan lingkaran ayam dan telur. Bagaimana Expad bisa digunakan kalau *source code*-nya sedang dibuat?

Tanya: *Saya sudah berhasil pula melakukan kompilasi Expad di Windows akan tetapi mengapa di titlebar selalu ada imbuhan [Freeware]?*

Jawab: Hal ini sebuah keterbatasan karena digunakan Qt Windows edisi non-komersial (QT-NC). Jika Anda ingin menghilangkannya, silakan gunakan Qt Windows yang lisensi penuh (dan membayar lebih dari US$ 1500 untuk itu). Untuk lebih rincinya, lihatlah FAQ mengenai hal ini di situs web Trolltech.