Chapter 12 – File, Stream, dan XML

Agenda

Pada chapter ini kita akan membahas tentang beberapa class khusus pada Qt Framework yang digunakan untuk bekerja dengan File dan dokumen XML. Adapun materi yang akan dibahas pada HOL ini adalah:

- Bekerja dengan QDir dan QFileInfo.
- Bekerja dengan QFile dan QTextStream.
- Menggunakan QDataStream
- Manipulasi dokumen XML dengan DOM
- QXMLStream Reader untuk membaca dokumen XML

Bekerja dengan Paths

QDir digunakan untuk bekerja dengan paths dan drives pada aplikasi Qt. QDir memiliki beberapa static method yang memudahkan anda bekerja dengan file sistem. Misal QDir::current() dapat digunakan untuk mengembalikan QDir dari direktori kerja anda, QDir::home() akan mengembalikan QDir dari home direktori pengguna, QDir::root() akan mengembalikan root direktori, dan QDir::drives() akan mengembalikan objek QList<QFileInfo> yang mewakili root dari semua drive yang ada.
Objek QFileInfo menyimpan informasi tentang file dan direktori, ada beberapa method penting yang sering digunakan yaitu:

- isDir(), isFile(), dan isSymLink() akan mengembalikan nilai true jika objek yang dicek berupa direktori, file, atau symbolic link (shortcut pada window).
- dir() dan absoluteDir() akan mengembalikan QDir yang mengandung informasi dari objek file. Method dir() akan mengembalikan direktori relatif dari direktori aktif, dan method absoluteDir() mengembalikan path direktori yang dimulai dari root.
- exists() akan menghasilkan nilai true jika objek tersebut ada.
- isHidden(), isReadable(), isWritable(), dan isExecutable() mengembalikan status hak akses dari file.
- fileName() akan mengembalikan QString berupa nama file tanpa path.
- filePath() akan mengembalikan QString berupa nama file beserta path, path dapat bersifat relatif terhadap direktori aktif.
- absoluteFilePath() akan mengembalikan QString berupa nama file beserta path, path diawali dari drive root.
- completeBaseName() and completeSuffix() akan mengembalikan QString berupa nama file dan ekstensi (suffix).

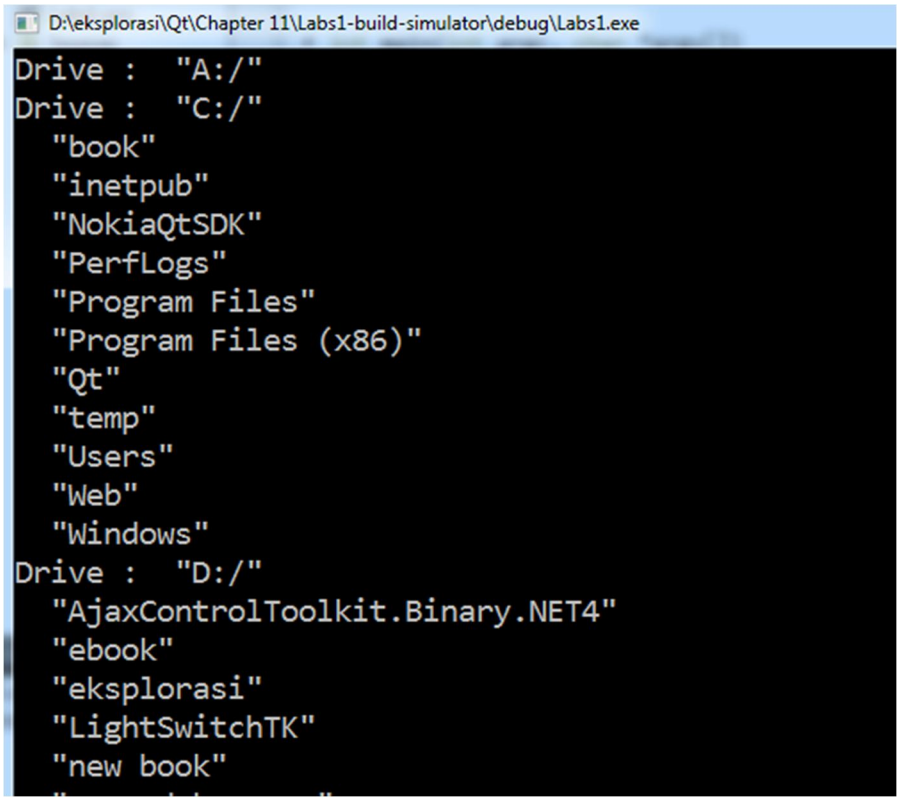
Program dibawah ini akan menunjukan cara penggunaan QDir untuk mengambil informasi direktori yang ada di komputer anda.

Labs.1 Menampilkan daftar drives dari root directories

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs1**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QDir>
#include <QFileInfo>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    foreach (QFileInfo drive, QDir::drives()) {
        qDebug() << "Drive : " << drive.absolutePath();
        QDir dir = drive.dir();
        dir.setFilter(QDir::Dirs);
        foreach (QFileInfo rootDirs, dir.entryInfoList()) {
            qDebug() << " " << rootDirs.fileName() ;
        }
    }
    return a.exec();
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



Analisa:

- Static method QDir::drives() akan mengembalikan collection berupa QList<FileInfo> yang berisi semua drive yang ada pada komputer.
- Untuk membaca semua drive beserta semua folder /direktori didalamnya anda dapat menggunakan foreach.
- Method setFilter(QDir::Dirs) artinya yang akan ditampilkan hanya direktori saja, tidak file atau symbolic link, untuk menampilkan semua (drive, direktori, file) anda dapat menggunakan QDir::AllEntries.
- Output dari program tersebut adalah daftar drive pada komputer anda beserta dengan folder yang ada didalamnya.

Dibawah ini adalah daftar kriteria yang dapat digunakan untuk melakukan filter.

- QDir::Dirs: Lists directories.
- QDir::AllDirs: Lists all directories.
- QDir::Files: Lists files.
- QDir::Drives: Lists drives.
- QDir::NoSymLinks: tidak menampilkan symbolic links.
- QDir::NoDotAndDotDot: tidak menampilkan special entries .
- QDir::AllEntries: Lists directories, files, drives, dan symbolic links.
- QDir::Readable: Lists readable files, harus dikombinasikan dengan Files atau Dirs.
- QDir::Writeable: Lists writable files. harus dikombinasikan dengan Files atau Dirs.
- QDir::Executable: Lists executable files. harus dikombinasikan dengan Files atau Dirs.
- QDir::Modified: Lists files yang sudah dimodifikasi.
- QDir::Hidden: Lists files yang sifatnya hidden.
- QDir::System: Lists system files.
- QDir::CaseSensitive: filter name harus case sensitive jika file sistem case sensitive.

Bekerja dengan Files

Anda dapat menggunakan QDir untuk mengambil informasi file dan QFileInfo untuk mengambil informasi file yang lebih lengkap, untuk proses manipulasi yang lebih jauh lagi seperti membuka, membaca, dan memodifikasi file anda harus menggunakan class QFile.

Agar lebih jelas bagaimana cara menggunakan QFile untuk membuka file, anda dapat mengerjakan labs dibawah ini.

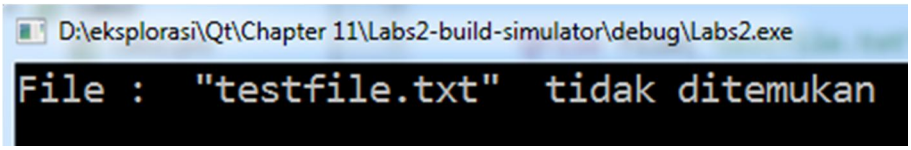
Labs.2 Memeriksa apakah file ada dan bisa diakses

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs2**, kemudian tulis kode berikut.

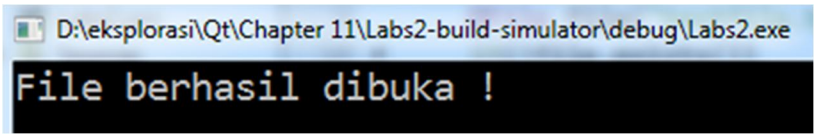
```
#include <QtCore/QCoreApplication>
```

```
#include <QFile>
#include <QDebug>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QFile file("testfile.txt");
    if(!file.exists())
    {
        qDebug() << "File : " << file.fileName() << " tidak ditemukan";
        return a.exec();
    }
    if(!file.open(QIODevice::WriteOnly))
    {
        qDebug() << "Tidak dapat membuka file " << file.fileName() << " untuk
ditulis";
        return a.exec();
    }
    qDebug("File berhasil dibuka !");
    file.close();
    return a.exec();
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



3. Output diatas mempunyai arti bahwa anda belum membuat file dengan nama “testfile.txt”, agar program diatas berhasil dijalankan buat file “testfile.txt” didalam folder Labs2-build-simulator. Kemudian jalankan kembali programnya, maka akan tampil output berikut.



Analisa

- Method exists() digunakan untuk mengecek apakah file yang ada atau tidak. Jika file tidak ditemukan maka program akan keluar dan menampilkan output program tidak ditemukan.
- Method open() digunakan untuk membuka file, permision yang digunakan pada program diatas adalah WriteOnly. Jika file tidak dapat dibuka maka akan keluar pesan tidak dapat membuka file.

Bekerja dengan Stream

Setelah anda membuka file akan lebih mudah untuk mengakses file tersebut menggunakan stream class. Qt hadir dengan 2 macam stream class, satu untuk teks file dan satu lagi untuk binary file. Dengan menggunakan stream untuk mengakses file anda dapat menggunakan operator << dan >>untuk menulis dan membaca data dari file.

Text Stream

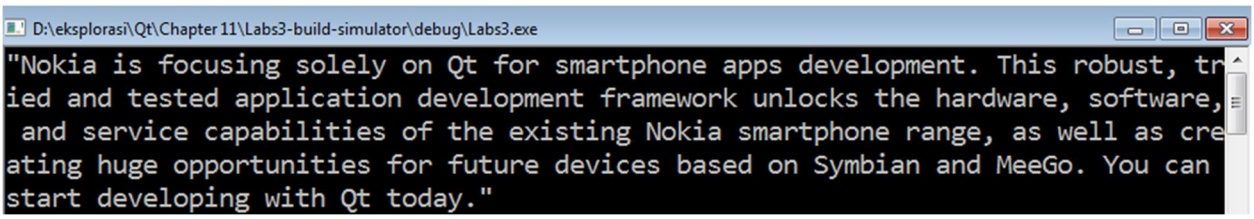
Untuk membuat text stream pada file, buat objek QFile dan buka file seperti biasa, disarankan jika anda menggunakan parameter QIODevice::Text dan hak akses QIODevice::ReadOnly. Agar lebih jelas bagaimana penggunaan stream buatlah contoh program pada **Labs3** dibawah ini.

Labs.3 Menggunakan Stream untuk membaca file

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs3**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QFile>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QFile file("D:\\sample.txt");
    if(!file.exists())
    {
        qDebug() << "File " << file.fileName() << " tidak ditemukan !";
        return a.exec();
    }
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug() << "File " << file.fileName() << " tidak dapat diakses !";
        return a.exec();
    }
    QTextStream stream(&file);
    //membaca semua teks yang ada dalam sample.txt
    QString teks = stream.readAll();
    qDebug() << teks;
    //membaca teks per line
    while(!stream.atEnd())
    {
        QString line = stream.readLine();
        qDebug() << line;
    }

    file.close();

    return a.exec();
}
```

2. Buat file teks pada alamat tertentu (pada contoh diatas di drive D:\sample.txt), masukan sembarang teks kedalam file tersebut.
3. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



Analisa

- Objek QTextStream digunakan jika anda ingin menggunakan stream untuk mengakses file text.
- Untuk membaca semua data yang ada pada file text gunakan method readAll().
- Untuk membaca file baris demi baris dapat digunakan method readLine() yang dijalankan didalam loop, method atEnd() digunakan untuk memeriksa apakah sudah sampai akhir file.

Data Stream

Pada beberapa kasus tertentu mungkin anda tidak dapat menggunakan file text untuk menyimpan data. Misalnya anda harus bekerja dengan file yang mempunyai format bukan teks, atau anda membutuhkan ukuran penyimpanan yang lebih kecil daripada menggunakan file teks. Dengan menyimpan data kedalam machine-readable seperti file biner ukuran file bisa menjadi lebih kecil dibandingkan dengan menyimpan data kedalam human-readable format seperti file teks.
Untuk membaca file biner anda dapat menggunakan objek QDataStream. Program dibawah ini menunjukan penggunaan objek QDataStream untuk mengakses file biner.

Labs.4 Menggunakan Data Stream

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs4**
2. Buka file Labs4.pro untuk menambahkan library GUI karena pada controh program ini digunakan class QColor.

```
#-------------------------------------------------
#
# Project created by QtCreator 2011-01-03T19:58:33
#
#-------------------------------------------------
QT       += core
QT       += gui
TARGET = Labs4
CONFIG   += console
CONFIG   -= app_bundle
TEMPLATE = app
SOURCES += main.cpp
```

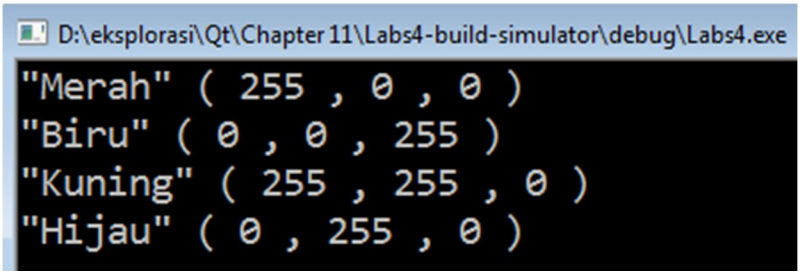
3. Kemudian tambahkan kode berikut pada file main.cpp.

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QDataStream>
#include <QList>
#include <QColor>
```

```
#include <QFile>

struct Warna
{
    QString text;
    QColor color;
};

QDataStream &operator << (QDataStream &stream, const Warna &data)
{
    stream << data.text << data.color;
    return stream;
}

QDataStream &operator >>(QDataStream &stream, Warna &data)
{
    stream >> data.text;
    stream >> data.color;
    return stream;
}

void saveList()
{
    QList<Warna> list;
    Warna data;
    data.text = "Merah";
    data.color = Qt::red;
    list << data;
    data.text = "Biru";
    data.color = Qt::blue;
    list << data;
    data.text = "Kuning";
    data.color = Qt::yellow;
    list << data;
    data.text = "Hijau";
    data.color = Qt::green;
    list << data;
    QFile file( "datastream.dat" );
    if( !file.open( QIODevice::WriteOnly ) )
        return;
    QDataStream stream( &file );
    stream.setVersion( QDataStream::Qt_4_7);
    stream << list;
    file.close();
}

void loadList()
{
    QList<Warna> list;
    QFile file( "datastream.dat" );
    if( !file.open( QIODevice::ReadOnly ) )
        return;
    QDataStream stream(&file);
    stream.setVersion(QDataStream::Qt_4_7);
```

```
    stream >> list;
    file.close();
    foreach( Warna data, list )
      qDebug() << data.text << "("
               << data.color.red() << ","
               << data.color.green() << ","
               << data.color.blue() << ")";
}

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    saveList();
    loadList();
    return a.exec();
}
```

4. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



Analisa

- Pada program diatas struct Warna adalah *user define type* yang dibuat sendiri, struct Warna memiliki dua member variabel yaitu text yang bertipe QString dan color yang bertipe QColor.
- Untuk memasukan data biner (dengan tipe data Warna) kedalam stream buat operator <<
- Untuk mengambil data biner dari stream buat operator >>
- Method saveList() digunakan untuk membuat objek warna , memasukan objek tersebut kedalam list dan menuliskannya kedalam file datastream.dat.
- Method loadList() digunakan untuk mengambil data stream dari file datastream.dat dan menampilkan data tersebut dengan cara membaca dari list.

XML

XML adalah meta-language yang dapat digunakan untuk menyimpan data terstruktur berupa string atau teks file. Komponen dasar penyusun dokumen XML adalah tag, attribute, dan teks. Contoh dokumen XML yang sederhana ditunjukan pada listing dibawah ini.

```
<document name="DocName">
      <author name="AuthorName" />
      Some text
</document>
```

Pada dokumen XML diatas document tag mengandung author tag dan teks . Tag document diawali dengan <document> dan diakhiri dengan closing tag </document>. Kedua tag document dan author memiliki attribute yang sama yaitu name.

Author tag tidak memiliki tag penutup karena tidak memiliki elemen lain didalamnya, cara penulisannya adalah <author />, ini sama dengan menuliskan <author></author>.

Qt mendukung tiga cara untuk memanipulasi dokumen XML yaitu QStreamReader, DOM, dan SAX. Untuk menggunakan library XML pada Qt anda harus menambahkan library XML pada Qt project file.

```
#-------------------------------------------------
#
# Project created by QtCreator 2011-01-03T21:43:04
#
#-------------------------------------------------
QT       += core
QT       -= gui
QT += xml
TARGET = Labs5
CONFIG   += console
CONFIG   -= app_bundle
TEMPLATE = app
SOURCES += main.cpp
```

DOM

DOM (Document Object Model) bekerja dengan cara merepresentasikan semua dokumen XML menjadi bentuk tree yang mempunyai node dan menyimpanya dalam memory.

Membuat File XML

Pertama kita akan mulai dengan membuat file XML menggunakan DOM, adapun tahapan yang akan kita kerjakan adalah membuat node, menyambungkan node, dan terakhir menulis dokumen XML. Agar lebih jelas mari kita coba buat program dibawah ini.

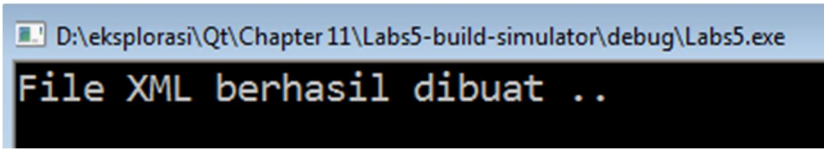
Labs.5 Membuat Nodes untuk membuat simple XML Document

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs5**, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <QDomDocument>
#include <QDomElement>
#include <QDomText>

int main(int argc, char *argv[])
```

```
{
    QCoreApplication a(argc, argv);

    QDomDocument dokumen;
    QDomElement mhs = dokumen.createElement("Mahasiswa");
    mhs.setAttribute("Jurusan","TI");

    QDomElement nim = dokumen.createElement("Nim");
    QDomElement ipk = dokumen.createElement("Ipk");
    QDomText nimtext = dokumen.createTextNode("22002321");
    QDomText ipktext = dokumen.createTextNode("3.5");

    dokumen.appendChild(mhs);
    mhs.appendChild(nim);
    nim.appendChild(nimtext);
    mhs.appendChild(ipk);
    ipk.appendChild(ipktext);

    QFile file("simple.xml");
    if(!file.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        qDebug() << "File tidak ditemukan !";
        a.exit(-1);
        return a.exec();
    }

    QTextStream stream(&file);
    stream << dokumen.toString();
    qDebug() << "File XML berhasil dibuat ..";

    file.close();
    return a.exec();
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



Hasil dari file XML “simple.xml” yang berhasil dibuat adalah :

```
<Mahasiswa Jurusan="TI">

 <Nim>22002321</Nim>

 <Ipk>3.5</Ipk>

</Mahasiswa>
```

Analisa

- Untuk membuat dokumen XML menggunakan DOM, pertama buat objek QDomDocument .
- Langkah selanjutnya adalah membuat objek QDomElement untuk membuat element Nim dan Ipk.
- Untuk menambahkan text pada element tambahkan objek QDomText.
- Setelah element dan text selesai dibuat, anda dapat memasangkan element dan teks menjadi child node dengan menggunakan method appendChild().
- Setelah dokumen XML selesai dibuat, anda dapat menuliskan dokumen tersebut ke file teks dengan menggunakan objek QTextStream, pada contoh program diatas dokumen XML disimpan dalam file "**simple.xml**"

Membaca XML file

Pada contoh sebelumnya anda telah bisa membuat dokumen XML dengan DOM, sekarang kita akan mencoba membaca dokumen XML yang sebelumnya sudah dibuat.
Pada contoh dibawah ini kita akan membaca dan memasukan file kedalam QDomDocument dan kita juga akan mempelajari bagaimana cara menemukan elemen dan teks yang terkandung dalam dokumen. Agar dokumen XML pada file dapat diload kedalam QDomDocument maka dokumen XML tersebut harus valid.

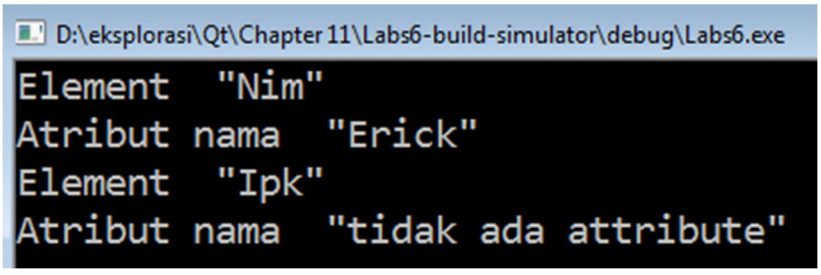
Labs.6 Membaca DOM dari dokumen XML

1. Untuk dokumen XML yang akan digunakan pada contoh program ini adalah dokumen XML yang sudah kita buat sebelumnya yaitu "simple.xml".
2. Buka Qt Creator dan buat project Qt Console Application baru dengan nama **Labs5**.
3. Jalankan dahulu program tersebut dengan menekan tombol Ctrl + R, agar folder simulator dengan nama Labs6-build-simulator dibuat.
4. Kopikan file "simple.xml" yang akan dibaca kedalam folder Labs6-build-simulator.
5. Kemudian buka file main.cp, tulis kode untuk membaca file XML berikut:

```
#include <QtCore/QCoreApplication>
#include <QFile>
#include <QTextStream>
#include <QDomDocument>
#include <QDomElement>
#include <QDomText>
#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QFile file("simple.xml");
```

```
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug() << "File " << file.fileName() << " tidak ditemukan";
        return a.exec();
    }

    QDomDocument dokumen;
    if(!dokumen.setContent(&file))
    {
        qDebug() << "Gagal untuk parsing ke DOM tree";
        file.close();
        return a.exec();
    }

    QDomElement dokumenElemen = dokumen.documentElement();
    QDomNode node = dokumenElemen.firstChild();
    while(!node.isNull())
    {
        if(node.isElement())
        {
            QDomElement element = node.toElement();
            qDebug() << "Element " << element.tagName();
            qDebug() << "Atribut nama " << element.attribute("nama","tidak ada
attribute");
        }

        if(node.isText())
        {
            QDomText teks = node.toText();
            qDebug() << teks.data();
        }
        node = node.nextSibling();
    }
    return a.exec();
}
```

6. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



Analisa

- Untuk membaca data dari file, seperti biasa gunakan objek QFile dan jalankan method open().

- Untuk mengambil data yang ada file untuk dimasukan kedalam objek QDomDocument gunakan method setContent().
- Untuk mengakses data elemen pada QDomDocument, buat objek QDomElement.
- Untuk mengambil node dari QDomElement, buat objek QDomNode. Method firstChild() digunakan untuk mengambil node awal pada QDomElement.
- Kemudian jika node yang dibaca tidak null, cek apakah node tersebut berupa elemen, jika node tersebut element lakukan konversi ke objek QDomElement dan gunakan method tagName() untuk menampilkan nama element dan method attribute() untuk menampilkan atribut jika ada.
- Cek juga apakah node bertipe teks dengan method isText(), jika teks konversi node kedalam objek QDomText kemudian tampilkan isi dari teks dengan menggunakan method toText().
- Untuk berpindah ke node selanjutnya gunakan method nextSibling().

Memodifikasi File XML

Pada topik sebelumnya kita sudah membahas bagaimana cara menulis dan membaca data dari dokumen XML. Pada topik kali ini akan dibahas bagaimana cara memodifikasi dokumen XML yang sudah anda load kedalam objek QDomDocument.

Labs.7 Modifikasi data dokumen XML

1. Buat project console aplication baru dengan nama Labs7, pilih Qt Simulator sebagai simulator yang digunakan. Jalankan program sehingga akan dibuat folder Labs7-build-simulator.
2. Pada folder Labs7-build-simulator buat file dengan nama “simple.xml”, kemudian tulis dokumen XML berikut.

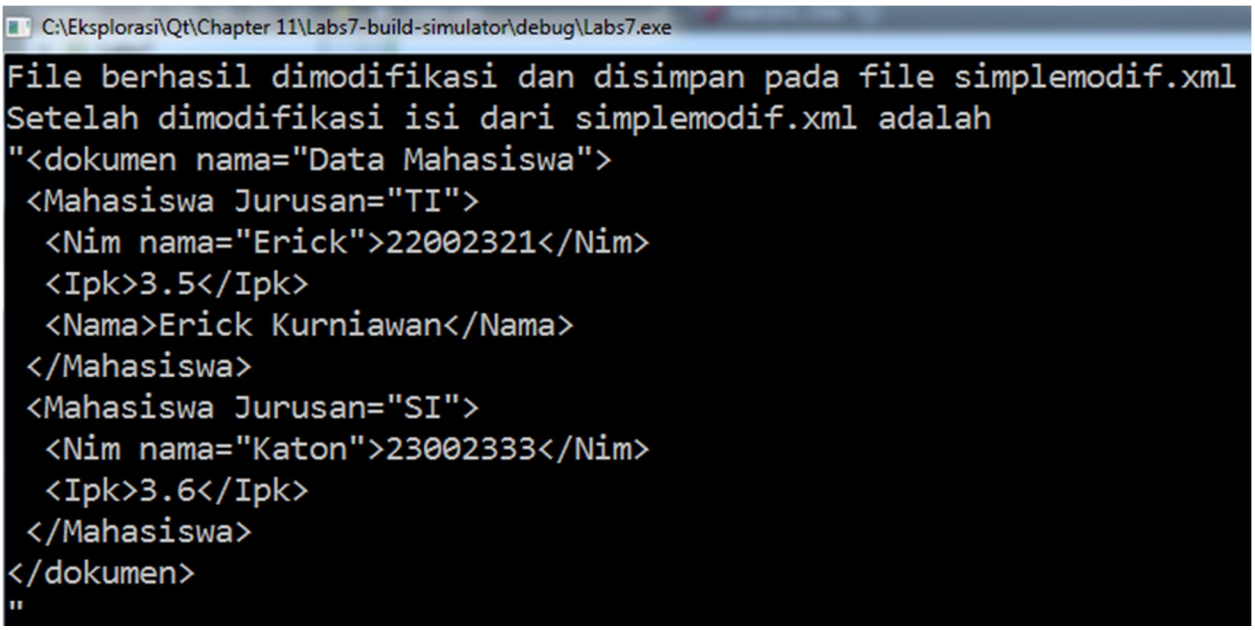
```
<dokumen nama="Data Mahasiswa">
    <Mahasiswa Jurusan="TI">
        <Nim nama="Erick">22002321</Nim>
        <Ipk>3.5</Ipk>
    </Mahasiswa>
    <Mahasiswa Jurusan="SI">
        <Nim nama="Katon">23002333</Nim>
        <Ipk>3.6</Ipk>
    </Mahasiswa>
</dokumen>
```

3. Kemudian pada file main.cpp tambahkan kode berikut untuk melakukan modifikasi file xml yang sudah kita buat sebelumnya.

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QFile>
#include <QTextStream>
#include <QDomDocument>
#include <QDomElement>
#include <QDomText>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QFile fileAsli("simple.xml");
    if(!fileAsli.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug() << "File " << fileAsli.fileName() << " tidak ditemukan";
        return a.exec();
    }

    QDomDocument dokumen;
    if(!dokumen.setContent(&fileAsli))
    {
        qDebug() << "Gagal parsing file ke DOM tree";
        fileAsli.close();
        return a.exec();
    }
    fileAsli.close();

    QDomElement elemenDokumen = dokumen.documentElement();
    QDomNodeList elemen = elemenDokumen.elementsByTagName("Mahasiswa");
    if(elemen.size() == 0)
    {
        QDomElement mhs = dokumen.createElement("Mahasiswa");
        elemenDokumen.insertBefore(mhs,QDomNode());
    }
    else
    {
        QDomElement mhs = elemen.at(0).toElement();
        QDomElement nama = dokumen.createElement("Nama");
        QDomText textNama = dokumen.createTextNode("Erick Kurniawan");
        nama.appendChild(textNama);
        mhs.appendChild(nama);
    }

    QFile fileModif("simplemodif.xml");
    if(!fileModif.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        qDebug() << "Gagal untuk membaca file xml";
        return a.exec();
    }
    QTextStream stream(&fileModif);
```

```
    stream << dokumen.toString();
    qDebug() << "File berhasil dimodifikasi dan disimpan pada file simplemodif.xml";
    fileModif.close();

    //membaca isi dari file simplemodif.xml
    qDebug() << "Setelah dimodifikasi isi dari simplemodif.xml adalah";
    if(!fileModif.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug() << "Gagal membaca file xml";
        return a.exec();
    }
    qDebug() << fileModif.readAll();
    fileModif.close();

    return a.exec();
}
```

4. Kemudian jalankan program diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



5. Untuk melihat hasil dari file xml yang sudah dimodifikasi, anda dapat membuka file "simplemodif.xml" yang ada pada folder **Labs7-build-simulator**.
6. Isi dari file "simplemodif.xml" adalah sebagai berikut

```
<dokumen nama="Data Mahasiswa">
 <Mahasiswa Jurusan="TI">
  <Nim nama="Erick">22002321</Nim>
  <Ipk>3.5</Ipk>
  <Nama>Erick Kurniawan</Nama>
 </Mahasiswa>
 <Mahasiswa Jurusan="SI">
```

```
  <Nim nama="Katon">23002333</Nim>
  <Ipk>3.6</Ipk>
 </Mahasiswa>
</dokumen>
```

Analisa

- Langkah pertama baca file simple.xml yang akan dimodifikasi, kemudian masukan isinya kedalam QDomDocument.
- Ambil root element dari dokumen menggunakan method documentElement() dan masukan kedalam objek bertipe QDomElement.
- Kode QDomNodeList elemen = elemenDokumen.elementsByTagName("Mahasiswa"); digunakan untuk mengambil semua element yg mempunyai tag <Mahasiswa> dan memasukannya kedalam variabel elemen yang bertipe QDomNodeList.
- Kode if(elemen.size() == 0) digunakan untuk memeriksa apakah elemen dengan tag <Mahasiswa> ditemukan, jika tidak ditemukan maka buat elemen <Mahasiswa>, jika ditemukan ambil elemen <Mahasiswa> pertama kemudian tambahkan elemen baru <Nama> beserta teks di dalam elemen <Mahasiswa> tersebut.
- Langkah terakhir adalah membuat QTextStream dan menyimpan dokumen XML yang sudah dimodifikasi kedalam file simplemodif.xml.

QXMLStream Reader

Selain menggunakan DOM untuk membaca dokumen XML, anda juga dapat menggunakan objek QXMLStreamReader. QXMLStreamReader adalah class parser XML tercepat dan termudah untuk digunakan, karena QXMLStreamReader parser bekerja secara incremental sehingga mempermudah pembacaan tag. QXMLStreamReader juga cocok digunakan untuk membaca file yang berukuran besar yang tidak cocok jika disimpan di memory. Beberapa token yang digunakan adalah sebagai berikut:

Token Type	Example	Getter Functions
`StartDocument`	N/A	`isStandaloneDocument()`
`EndDocument`	N/A	`isStandaloneDocument()`
`StartElement`	`<item>`	`namespaceUri()`, `name()`, `attributes()`, `namespaceDeclarations()`
`EndElement`	`</item>`	`namespaceUri()`, `name()`
`Characters`	`AT&amp;T`	`text()`, `isWhitespace()`, `isCDATA()`
`Comment`	`<!-- fix -->`	`text()`
`DTD`	`<!DOCTYPE ...>`	`text()`, `notationDeclarations()`, `entityDeclarations()`
`EntityReference`	`&trade;`	`name()`, `text()`
`ProcessingInstruction`	`<?alert?>`	`processingInstructionTarget()`,

Token Type	Example	Getter Functions
		`processingInstructionData()`
`Invalid`	`>&<!`	`error()`, `errorString()`

Misal anda memiliki dokumen XML sebagai berikut
<doc>

<quote>Einmal ist keinmal</quote>

</doc>

Setiap anda menggunakan method readNext() maka akan dibaca satu token. Untuk kode diatas kita dapat membaca pertoken.

StartDocument

StartElement (name() == "doc")

StartElement (name() == "quote")

Characters (text() == "Einmal ist keinmal")

EndElement (name() == "quote")

EndElement (name() == "doc")

EndDocument

Setelah memanggil method readNext(), anda dapat mengecek token yang sedang aktif dengan menggunakan isStartElement(), isCharacter(), atau menggunakan fungsi yang mirip seperti state().

Labs.8 Menggunakan QXMLStream Reader untuk membaca XML

1. Buat aplikasi console baru dengan nama Labs8, pilih Qt Simulator untuk menampilkan outputnya.
2. Buat file simple.xml dan buat dokumen XML sebagai berikut

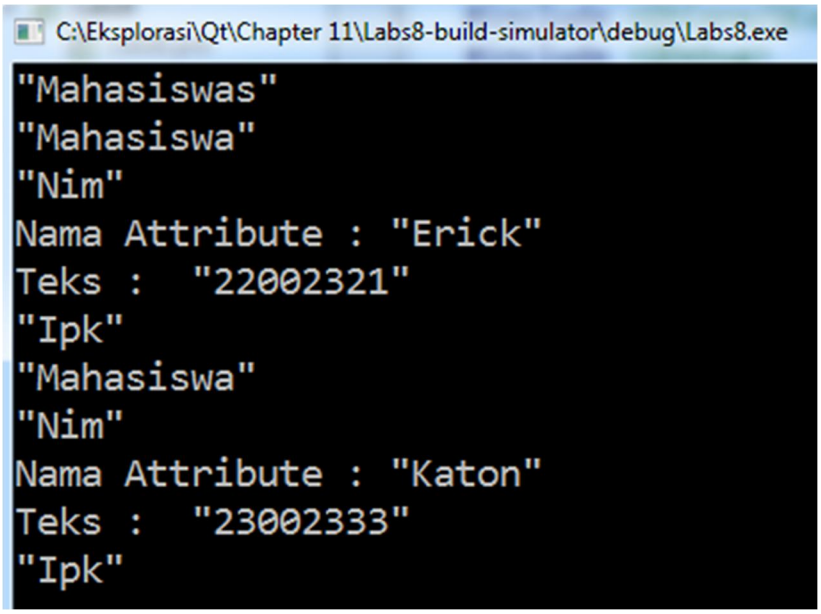
```
<Mahasiswas>
	<Mahasiswa>
		<Nim nama="Erick">22002321</Nim>
		<Ipk>3.5</Ipk>
	</Mahasiswa>
	<Mahasiswa>
		<Nim nama="Katon">23002333</Nim>
		<Ipk>3.6</Ipk>
```

```
        </Mahasiswa>
</Mahasiswas>
```

3. Kemudian ketikan kode berikut pada main.cpp untuk membaca data dari dokumen XML.

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QFile>
#include <QXmlStreamReader>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QFile file("simple.xml");
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qDebug() << "File tidak ditemukan ";
        return a.exec();
    }
    QXmlStreamReader reader;
    reader.setDevice(&file);
    while(!reader.atEnd())
    {
       reader.readNext();
       if(reader.isStartElement())
       {
           qDebug() << reader.name();
           if(reader.name() == "Nim")
           {
                qDebug() << "Nama Attribute :" << reader.attributes().value("nama");
                qDebug() << "Teks : " << reader.readElementText();
           }
       }
    }
    file.close();
    return a.exec();
}
```

4. Jalankan aplikasi tersebut, maka akan ditampikan output sebagai berikut.



Analisa

- Untuk membaca dokumen XML menggunakan **`QXMLStreamReader`** anda dapat melakukan looping dengan memeriksa apakah sudah sampai pada akhir dokumen **`while(!reader.atEnd())`**.
- Method **`reader.readNext();`** digunakan untuk berpindah token.
- Method **`reader.isStartElement()`** digunakan untuk mengecek apakah token yg sekarang aktif adalah elemen awal.
- Kode **`if(reader.name() == "Nim")`** digunakan untuk mengecek apakah nama elemen adalah "Nim" jika ya baca attribute dan teks pada elemen tersebut untuk ditampilkan.

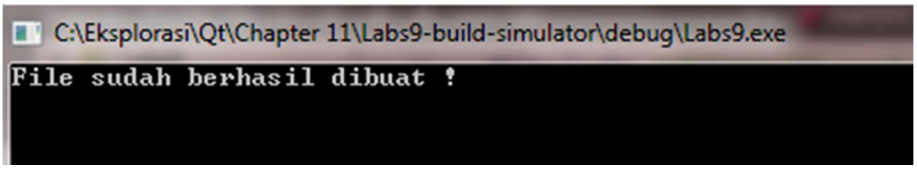
Labs.9 Membuat dokumen XML dengan QXMLStreamWriter

1. Buat aplikasi console dengan nama Labs9, kemudian tulis kode berikut

```
#include <QtCore/QCoreApplication>
#include <QDebug>
#include <QFile>
#include <QXmlStreamWriter>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    QFile file("simple.xml");
    if(!file.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        qDebug() << "File tidak ditemukan..";
        return a.exec();
    }

    QXmlStreamWriter writer(&file);
    writer.setAutoFormatting(true);
```

```
    writer.writeStartDocument();
    writer.writeStartElement("Books");
    writer.writeStartElement("Book");
    writer.writeStartElement("Author");
    writer.writeAttribute("Name","Erick Kurniawan");
    writer.writeAttribute("Title","Qt Programming");
    writer.writeEndDocument();
    file.close();
    qDebug() << "File sudah berhasil dibuat !";
    return a.exec();
}
```

2. Jalankan program diatas untuk menggenerate dokumen XML dengan nama simple.xml.



3. Isi dari dokumen XML yang barusan anda buat adalah sebagai berikut.

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
    <Book>
        <Author name="Erick Kurniawan" Title="ASP.NET 3.5"/>
    </Book>
</Books>
```

Analisa

- Anda dapat menggunakan **`QXMLStreamWriter`** untuk membuat dokumen XML.
- Kode **`writer.setAutoFormatting(true);`** digunakan untuk memformat secara otomatis dokumen XML yang akan dibuat, misal menambahkan line break dan indentation pada bagian yang kosong pada elemen. Tujuan utamanya adalah memisahkan data menjadi beberapa baris sehingga membantu dalam pembacaan dokumen.
- Kode **`writer.writeStartDocument();`** digunakan pada saat pertama kali dokumen akan dibuat.
- Kode **`writer.writeStartElement("Books");`** digunakan untuk menuliskan elemen Books.
- Kode **`writer.writeAttribute("Name","Erick Kurniawan");`** digunakan untuk menambahkan attribute pada elemen tertentu.
- Kode **`writer.writeEndDocument();`** digunakan untuk menutup semua start element yang sebelumnya dibuat dengan method **`writeStartElement()`**.