Lecture Qt010
Files

Instructor: David J. Coe

CPE 353 – Software Design and Engineering

Department of Electrical and Computer Engineering

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Outline

- **QIODevice** Class
- **QFile** Class
- **QDir** Class
- Opening Files
- Files as Text Streams
- Files as Data Streams
- Lessons Learned
- Key Points

QIODevice Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Abstract class that provides interface to classes which must read or write blocks of data
- **QIODevice**
  - **QAbstractSocket**
    - **QUdpSocket**
    - **QTcpSocket**
  - **QBuffer**
  - **QFile**
    - **QTemporaryFile**
  - **QProcess**

QIODevice Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Use **open()** to open an object from a class derived from **QIODevice**
  - **open()** is defined as virtual within **QIODevice**
  - Returns **true** if successful, **false** otherwise
- Parameters of open() include
  - **QIODevice::ReadOnly**
  - **QIODevice::WriteOnly**
  - **QIODevice::ReadWrite**
  - **QIODevice::Truncate**
    - Delete previous contents
  - **QIODevice::Append**
    - Add to end of existing content
  - **QIODevice::Text**
    - Use OS specific EOL

QFile Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Derived from **QIODevice**
- For reading and writing text and binary files
  - Use constructor to create a **QFile** object associated with a particular file

```
QFile myData( "data.txt" );

myData.open( QIODevice::ReadOnly );
```

- Can also create the **QFile** object and associate a file with it later (only if file has not been opened)

```
QFile myData;

myData.setFileName( "data.txt" );

myData.open( QIODevice::ReadOnly );
```

QFile Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Other useful **QFile** methods
  - `size()`
  - `setPermissions(...)`
  - `permissions()`
  - `rename(...)`
  - `remove(...)`
  - `exists(...)` returns `true` if file exists, otherwise `false`
  - `copy(...)` copies file as specified
  - `atEnd()` `true` if `EOF` reached; `false` otherwise

QDir Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Goal is to give access to directory structure and its contents
  - Can use absolute or relative path names
  - Absolute paths begin with directory separator "/" regardless of platform
  - If on a Windows machine, "/" will be translated to "\"

```
QDir("C:/Documents and Settings");

QDir("/usr/local/bin");
```

  - Relative paths lack an initial directory separator

```
QDir("debug/example.exe");
```

- Methods **isAbsolute()** and **isRelative()** provide indication of which type of path is currently in use
- **makeAbsolute()** forces conversion to absolute path

QDir Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Other useful QDir methods
  - **current()** returns current working directory
  - **home()** returns user home directory
  - **exists()** true if object exists; false otherwise
  - **filePath()** returns filename including path
  - **absoluteFilePath()** returns absolute path
  - **root()** returns root directory
  - **cd(...)** change to specified directory
  - **cdUp()** move up to parent directory

QDir Class

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Example

```
QDir d = QDir::root();
...
if ( !d.isRoot() )
{
 qDebug() << "Error";
}
```

Example01: QFile

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
// File Example 01
#include <QFile>
#include <QtDebug>
int main(int argc, char* argv[])
{
  QFile someFile( "sample.txt" );
  if ( !someFile.exists() )
  {
    qDebug() << "Error -- file does not exist";
  }
  else if ( !someFile.open( QIODevice::ReadOnly ) )
  {
    qDebug() << "Error -- unable to open file for input";
  }
  else
  {
    qDebug() << "File opened for input successfully";
  }
  someFile.close();

  return 0;
} // End main()
```

Sample Output

```
$
$ ./FileExample01
File opened for input successfully
$
$
```

Files as Text Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Specify **IODevice::Text** with **open()**
- Associate a **QTextStream** object with the file
- Use input functions such as **>>**, **readLine()**, or **readAll()**
- Use output functions such as **<<** or **write()**

Example02: Files as Text Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
// File Example 02
#include <QFile>
#include <QtDebug>
int main(int argc, char* argv[])
{
  QFile someFile( "sample.txt" );
  if ( !someFile.exists() )
  {
    qDebug() << "Error -- file does not exist";
  }
  else if ( !someFile.open( QIODevice::ReadOnly |
            QIODevice::Text ) )  // Open for text input
  {
    qDebug() << "Error -- unable to open file for input";
  }
  else
  {
    qDebug() << "File opened for input successfully";
    QTextStream  someStream(&someFile);
    QString      someValue;
    while ( !someStream.atEnd() )
    {
      someStream >> someValue;
      qDebug() << someValue;
    }
    someFile.close();
  }
  return 0;
} // End main()
```

Sample Output

```
$ cat sample.txt
Hello_World
Goodbye_World
$

$
$ ./FileExample02
File opened for input successfully
"Hello_World"
"Goodbye_World"
""
$
```

Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Binary format can reduce file size
- Match data types exactly when using the **<<** and **>>** operators
  - Qt equivalents (such as **qint16**) are safe choices
- Be sure to read/write the same binary format as it has changed as Qt continues to evolve
  - Specify the version
- Ex. **QDataStream::Qt_4_1**
  - Use the most recent version

Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Three strategies for dealing with version issue (Blanchette and Summerfield)
  - Hard code of version number
  - Embed version number within file
  - Embed logic to force selection from a handful of hard coded versions
  - Embed version number within file
- Reading can then adjust version to exactly match
  - See **Qt Assistant** for sample code

Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Need to turn data into sequence of binary values
  - Most C++ and Qt types already "serialization ready" (i.e. **>>** and **<<** operations previously defined)
- Two approaches to serialization of data
  - Manual serialization
- Data consists of several different values which may or not be bundled within an object
- Explicitly use **<<** and or **>>** to write or read each component
  - Overload **<<** and **>>** for the data object
- Define how you want the serialization to occur for object

Example03: Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
// File Example 03
#include <QFile>
#include <QtDebug>
#include <QString>
#include <QChar>
#include <QDataStream>
#include <QList>
class Student
{
  private:
    QString name;
    quint16  age;
    QChar   grade;
  public:
    // Constructors
    Student() {  }
    Student(QString n, quint16 a, QChar g) { name = n;  age = a;  grade = g; }
    // Getters
    QString getName() { return name; }
    quint16 getAge() { return age; }
    QChar getGrade() { return grade; }
    // Setters
    void setName(QString n) { name = n; }
    void setAge(quint16 a) { age = a; }
    void setGrade(QChar g) { grade = g; }
};
```

Example03: Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
QDataStream& operator<<(QDataStream& ds, Student& s)
{
  ds << s.getName() << s.getAge() << s.getGrade();
  return ds;
}


QDataStream& operator>>(QDataStream& ds, Student& s)
{
  QString name;
  quint16 age;
  QChar grade;

  ds >> name >> age >> grade;
  s.setName(name);
  s.setAge(age);
  s.setGrade(grade);

  return ds;
}
```

Example03: Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
int main(int argc, char* argv[])
{
  QFile outFile( "sample.txt" );
  if ( !outFile.open( QIODevice::WriteOnly ) )
  {
    qDebug() << "Error -- unable to open file for output";
  }
  else
  {
    qDebug() << "File opened for output successfully";
    Student student1("Homer Simpson", 50, 'F');
    Student student2("Bart Simpson", 10, 'D');
    Student student3("Lisa Simpson", 8, 'A');

    QDataStream outStream(&outFile);
    outStream.setVersion( QDataStream::Qt_4_1);
    outStream << student1 << student2 << student3;
    outFile.close();
```

Example03: Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
  QFile inFile( "sample.txt" );                  // Associate data stream with input file
  if ( !inFile.open( QIODevice::ReadOnly ) )
  {
    qDebug() << "Error -- unable to open file for input";
  }
  else
  {
    QDataStream inStream(&inFile);
    inStream.setVersion( QDataStream::Qt_4_1);  // Set version to match

    Student s;                                   // Create temporary student variable

    while ( !inFile.atEnd() )                    // While not at EOF
    {
      inStream >> s;                             // Input a student

      // Write to console
      qDebug() << s.getName() << " " << s.getAge() << " " << s.getGrade();
    }
    inFile.close();                              // Close input file
  }

  return 0;
} // End main()
```

Example03: Files as Data Streams

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
$ ./FileExample03
File opened for output successfully
"Homer Simpson"   50   'F'
"Bart Simpson"   10   'D'
"Lisa Simpson"   8   'A'
$
```

Lessons Learned

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- In most cases, treating files as text streams will be the simplest solution
  - Files are not compressed
  - Readable in a text editor
  - Simplifies debugging

Key Points

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- **QDir** and **QFile** provide a convenient means of abstracting directory and file management independent of target platform
- Text stream and data stream objects provide an interface layer that allows use of familiar operators such as **>>** and **<<**