Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

PLAYLIST MANAGER

Simon Karlsson

46:2016

Datum för publicering: 12.12.2016
Handledare: Agneta Eriksson-Granskog

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Simon Karlsson
Arbetets namn:	Playlist Manager
Handledare:	Agneta Eriksson-Granskog
Uppdragsgivare:	Simon Karlsson

Abstrakt

Syftet med examensarbetet är att skapa ett program som kan kopiera mediafiler från många mappar till en mapp. Programmet ska också enkelt kunna byta namn på filer innan de kopieras.

Jag döpte projektet till Playlist Manager för att jag ville ha ett program som kunde förenkla kopiering av musik från mappar som representerar spelningslistor till en enda mapp med all musik.

För att uppnå syftet har jag använt programmeringsspråket C++ och utvecklingsmiljön Qt.

Nyckelord (sökord)

Qt, C++, Mediafiler

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
46:2016	1458-1531	Svenska	19 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
12.12.2016	02.12.2016	12.12.2016

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Information Technology
Author:	Simon Karlsson
Title:	Playlist Manager
Academic Supervisor:	Agneta Eriksson-Granskog
Technical Supervisor:	Simon Karlsson

Abstract

The aim of the thesis is to create a program that can copy media files from several folders to a single folder. The program will also be able to rename files before copying easily.

I named the project Playlist Manager because I wanted a program that could simplify the copying of music from folders representing playlists to a single folder with all the music.

To achieve the purpose I have used the programming language C ++ and the environment for program development Qt.

Keywords

Qt, C++, Media files

Serial number:	ISSN:	Language:	Number of pages:
46:2016	1458-1531	Swedish	19 pages

Handed in:	Date of presentation:	Approved on:
12.12.2016	02.12.2016	12.12.2016

INNEHÅLL

1 INLEDNING

1.1 Bakgrund

Jag kom på idén för programmet när jag kopierade musik från mappar som representerade spelningslistor till en mapp som skulle innehålla all musik. Jag började undersöka om det fanns något enkelt sätt att automatisera kopieringen av musik från många mappar till en mapp. Jag hittade en utvecklingsmiljö för C++ programmering med namnet Qt som innehöll klasser som kunde användas för att få den funktionalitet jag var ute efter.

1.2 Syfte

Syftet med projektet är att få lätt hantering av många mediamappar. Man ska även få en bra överblick över vilka filer som är giltiga mediafiler och vilka som är ogiltiga. Filer som är giltiga ska användaren kunna byta namn på. Programet ska även kunna kopiera innehåll från ett flertal mediamappar in i en mapp. I arbetet undersöks även möjligheten att kopiera ett flertal utvalda valda mediafiler från en mapp till en annan. Med mediafil menar jag en fil med ändelsen .mp3, .mp4, .flv eller .avi.

1.3 Metod

Jag använde mig av C++ som programmeringsspråk och Qt som *integrated development environment* (IDE). Uppgiften delades i hög- och lågnivå. Den låga nivån ska inte innehålla regler eller logik och ska ha lite till ingen felhantering på lågnivå. Inledningsvis sammanställsdes listor med avgränsningar, krav och saker som är bra att ha. Kunskaper i programmering med C++ har jag delvis fått från min utbildning vid Högskolan på Åland och delvis från självstudier. Kunskap hämtades även vid behov från Stack Overflow, Qt Center och Qts egen dokumentation, samtliga på nätet.

1.4 Avgränsningar

När användaren använder programmet för kopiering, namnbyte och uppspelning finns dessa avgränsningar:

- Användaren kan bara välja en fil åt gången för uppspelning.
- Man ska inte kunna byta namn på ogiltiga filer.
- Mappar kan bara användas på en plats i programmet.
- Programmet kommer bara att stöda mediafiler som har ändelserna .mp3, .mp4, .flv samt .avi.
- Programmet kommer inte stöda mer än två språk.
- Programmet använder inte trådar och är inte trådsäkert.

2 QT

Qt är en utvecklingsmiljö för C++. Q:et i namnet kommer från att Haavard Nord gillade hur det såg ut i Haavard Emacsfont. T:et kommer ifrån att de blev inspirerade av et *toolkit* som heter Xt (Wikipedia, 2015).

2.1 Bakgrund

Qt utvecklades av Haavard Nord och Eirik Chambe-Eng runt år 1991. Haavard Nord var verkställande direktör och Eirik Chambe-Eng var direktör för företeget *Quasar Technologies* som låg i Norge. Bytte namn först till Troll Tech och sedan till Trolltech (Wikipedia, 2016).
I början fungerade Qt bara på Linux och på Windows. År 2001 började stöd läggas till för OSX men var inte tillgängligt för alla licenstyper förrän år 2003 (Wikipedia, 2016).

Trolltech blev uppköpt av Nokia år 2008. Nokia bytte namn på företaget till Qt *Software* och senare till Qt *Development Frameworks*. År 2011 sålde Nokia den kommersiella delen av Qt till Digai (Wikipedia, 2016).

2.2 Varför Qt?

Jag valde att använda QT 5.7 på grund av dess nätverksgränssnitt. Qt är även väldigt flexibelt och gör det lätt att få program att fungera på flera plattformar. Qt innehåller strukturer för att gå igenom en mapp och få all information om innehållet i mappen som fungerar på flera plattformar. Det innehåller även struktur för mediaspelning. Qts *debugger* är också väldigt lätt att använda och gör därför det lättare att hitta eventuella fel. Om man vill kontrollera var ett fel ligger så lägger man till brytpunkter. Sedan startar man Qts *debugger* som försöker gå till bryt punkten om den kraschar före brytpunkten så kan man sätta in flera tills en brytpunkt kan nås. När Qts *debugger* har kommit till en brytpunkt så kan man stega sig framåt tills man hittar kraschen/felet.

2.3 QObject

QObject är basen till alla objekt och är basen för Qts objektmodell. Den centrala delen för modellen är en kraftfull mekanism som kallas signaler och *slots*. Dessa medför smidig kommunikation mellan objekt. Programmeraren kan koppla en signal till en *slot* med funktionen *connect*() och ta bort kopplingen med funktionen *disconnect()* . När ett *QObject* tas bort avger den signalen *destroyed()*. Denna signal kan fångas upp för att undvika att objektet lämnar *”dangling pointers”*.

En signal har ingen kod i klassen som den deklareras i. Koden för en signal finns inuti den *slot* som den kopplas till. För att aktivera en *solt* som är kopplat till en signal så använder man *emit* till exempel *emit mySignal()*. När en *emit* används så kallas alla *slots* som är kopplade till signalen och koden exekveras.

Alla *QObject* stöder Qts översättningsfunktionalitet vilket gör det lättare för utvecklare att översätta en applikation och samtidigt ha en struktur över den översatta texten. (The Qt Company , 2016). Figur 1 visar hur filen ser ut när man skapar den och säger att filen ska ärva från *QObject*.

```
#ifndef EXAMPLE_H
#define EXAMPLE_H

#include <QObject>

class Example : public QObject
{
   Q_OBJECT
public:
   explicit Example(QObject *parent = 0);

signals:

public slots:

};

#endif // EXAMPLE_H
```

Figur 1 Hur .h filen kan se ut när man vill ärva egenskaper från QObject

3 KRAVSPECIFIKATION

3.1 Krav

Mina krav för programmet är:

1. Ska veta hur många filer i en mapp som inte var av tillåtet format.
2. Ska veta hur många filer i en mapp som var av tillåtet format.
3. Ska kunna visa en lista på alla giltiga media filer i en mapp.
4. Kunna visa en lista med ogiltiga media filer i en mapp.
5. Låta användaren byta namn på giltiga media filer.
6. Kunna kopiera filer från en mapp till en annan.

3.2 Bra att ha

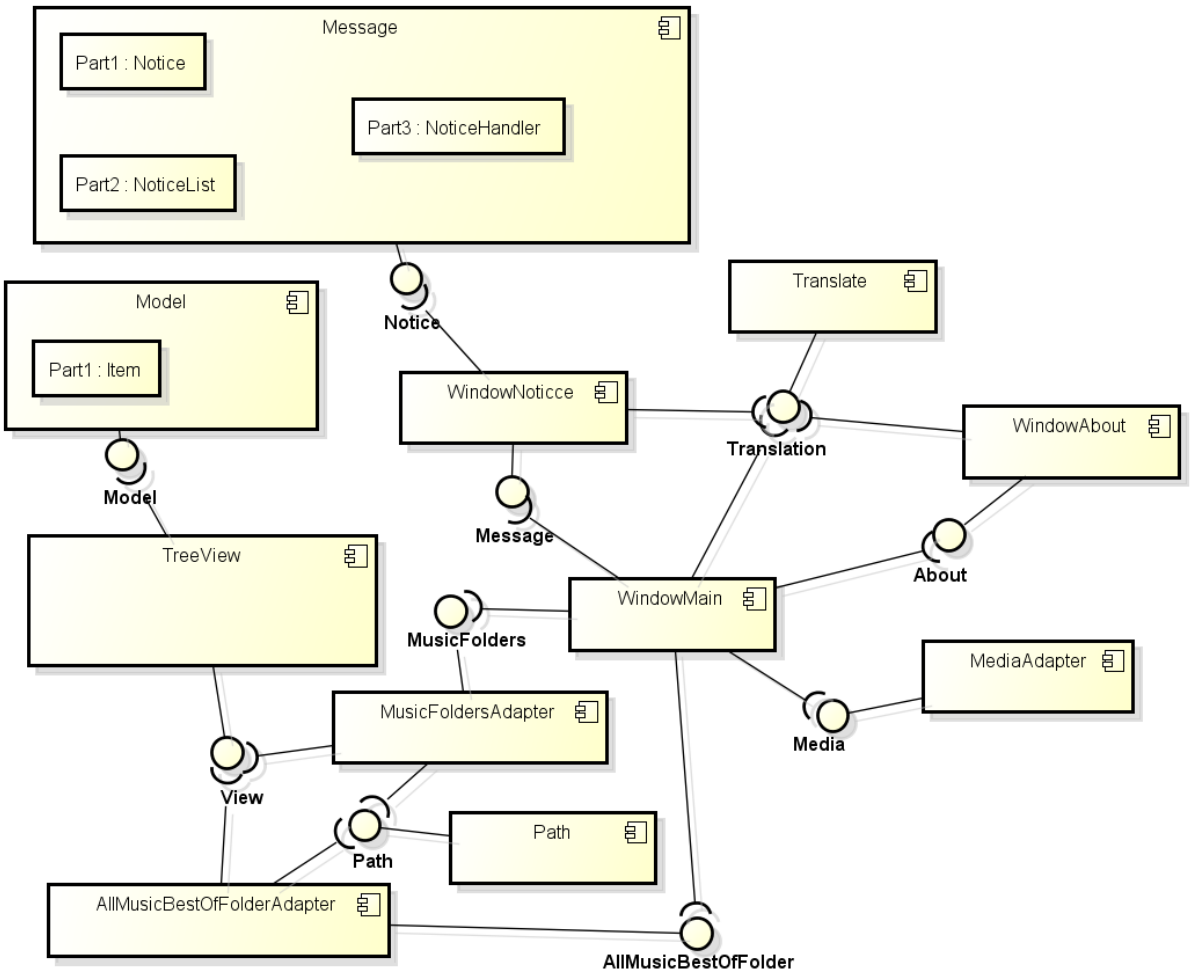
Funktioner som kan vara var att ha:

1. Kunna spela upp en fil.
2. Ha flera språk.
3. Ha trådar.
4. Komma ihåg inställningar.
5. Hållbarheten av media filer som spelas upp.
6. Kunna välja ut specifika filer från mappar och enbart kopiera dem.

4 PLAYLIST MANAGER

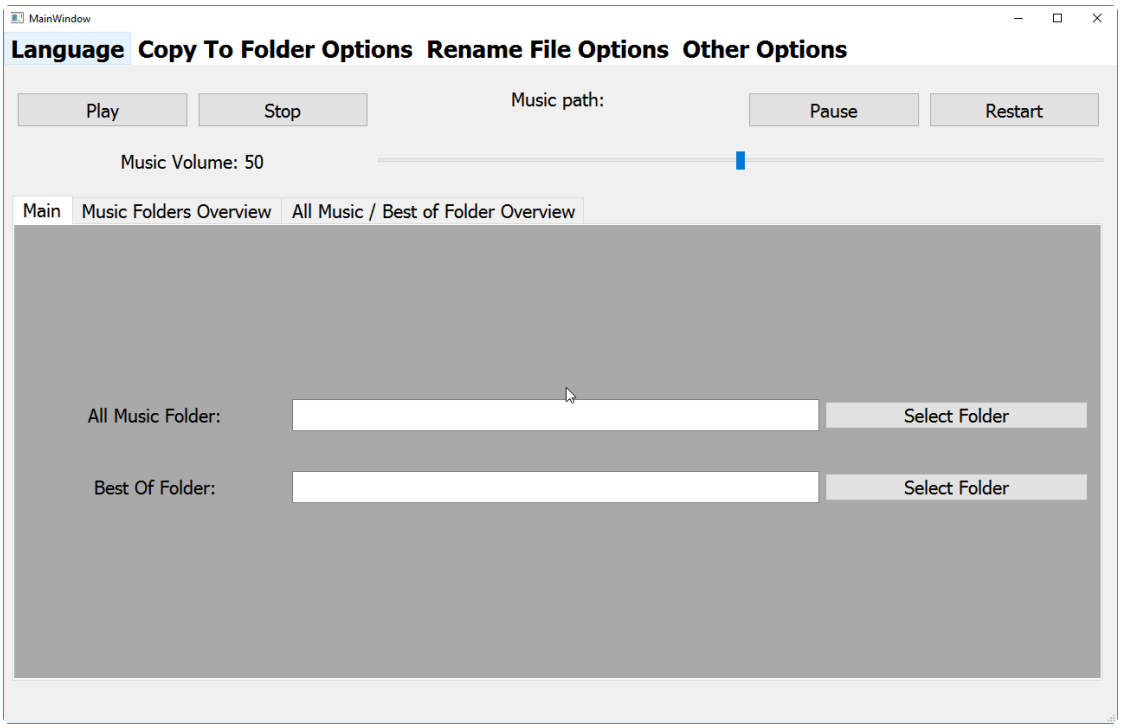
4.1 Funktionalitet

Programet använder engelska som standardspråk och låter användaren bytta till ett annat. Det hjälper användaren att samla ihop alla mediafiler på ett ställe. Programet hjälper även användaren att enbart flytta specifika filer från mappen som har specificerats som huvudmediamapp till en separat mapp. Figur 2 visar komponentdiagrammet för programmet.



Figur 2 Komponentdiagram

Figur 3 visar hur programmet ser ut när det startas.



Figur 3 Programstart

Språkmenyn är den enda menyn som inte är hårdkodad. Den genereras när programmet startas. Figur 4 visar hur språkmenyn genereras.

När språkmenyn genereras så kontrolleras att *QMenu* objektet är initialiserat. Sedan renas *QMenu* på eventuella *QAction* (underrubriker till menyn). För att radioknapparna ska fungera skapas en *QActionGroup* som ser till att bara en av knapparna kan vara markerad åt gången. Efter detta kontrolleras om språkfiler kunde hittas. Om inte, så kommer det bara finnas engelska som språkalternativ. Nästa steg är att koppla *QActionGroups* signal *triggered(QAction*)* med klassen Translators slot *languageChanged(QAction*)* som gör att så fort en ny radioknapp blir markerad så aktiveras *languageChanged(QAction*).*

```
void Translate::createLanguageMenu(QMenu *m) {
   assert(m != NULL);

   foreach(QAction* action,m->actions() ) { m->removeAction(action); }

   QActionGroup* langGroup = new QActionGroup(m);
   QStringList fileNames;

   langGroup->setExclusive(true);

   if(this->v_found == false) {
      addActionToMenu("English",m,langGroup);
      return;
   }

   connect(langGroup,
                SIGNAL(triggered(QAction*) ),
                this,SLOT(languageChanged(QAction*) ) );

   QDir dir(this->v_langPath);
   fileNames = dir.entryList(QStringList("Trans_*.qm") );

   for(auto i = 0; i < fileNames.size(); i++) {
         QString locale = fileNames[i];

         QString lang =  QLocale::languageToString(
                              QLocale(getLocaleName(locale) ).language()  );

         addActionToMenu(lang,m,langGroup);
   }

}
```

Figur 4 Generering av språkmeny

När användaren väljer en mapp som för innehåll av all musik/*best of* skannar programmet mappens innehåll för att ge användaren en överblick på vad mappen innehåller. Programmet skapar då en lista med filer av giltigt format och en med filer av ogiltigt format. En mediafil är av giltigt format om den innehåller en av de ändelser som programmet stöder (Se kapitel 1.4 för giltiga ändelser).

Figur 5 visar hur en mapp skannas. Först sätts sökvägen in i klassen *QDir* som gör det lättare att få information om en mapp. Sedan kontrolleras att sökvägen existerar. Efter detta förstörs de gamla fillistorna. När listorna har förstörts uppdateras sökvägen. Sedan går man igenom

alla poster i mappen och kontrollerar att basnamnet inte är den nuvarande sökvägen (.) eller förälderns sökväg (..). Efter detta kontrolleras om posten är en fil. Om posten är en fil slås extensionen ihop med en punkt.

När detta är gjort kontrolleras om listan med giltiga filändelser är tom. Om listan är tom blir filen automatiskt giltig. Om den inte är tom och den innehåller den nuvarande filens extension är filen giltig. *Folder* mall-klassen kan vara av *Folder*<*FileMusic*,*FileOther*> eller *Folder*<*FileMusic*,*FileMusic*>.

```
template <class T1,class T2> void Folder<T1,T2>::scanFolder(QStringList valid)
{
    QDir d(fullPath() );

    if(d.exists() == false) {
        qDebug()<<"Folder could not be found";
        exit(1);
    }
    this->destroyLists();

    d.refresh();

    foreach(QFileInfo i,d.entryInfoList() ) {
      QString name = i.baseName();
      //skip . & ..
      if(name.compare(".") == 0 || name.compare("..") == 0) { continue; }
      //Folder Found
      if(i.isFile() == false) { continue; }

      QString p = "."+i.suffix();

      if(valid.isEmpty() == true) {
            this->v_validFiles.push_back(QSharedPointer<T1>(new T1(i,this-
            >v_parent) ) );
      }
      else if(valid.contains(p,Qt::CaseInsensitive) == true) {
            this->v_validFiles.push_back(QSharedPointer<T1>(new T1(i,this-
            >v_parent) ) );
       }
       else {
            this->v_inValidFiles.push_back(QSharedPointer<T2>(new T2(i,this-
            >v_parent) ) );
       }
    }
}
```

Figur 5 Hur en mapp gås igenom

Användaren kan spela upp giltiga mediafiler. I figur 6 visas förloppet för en fils uppspelning. Först kontrolleras om filen kan spelas upp. Sedan spelas filen upp och texten som innehåller musiksökvägen uppdateras i det grafiska gränssnittet. *MEDIA_STR* är en egen gjord struktur som innehåller som innehåller två *QString* som representerar filens sökväg & filens namn.

```
void MediaAdapter::playSelectedFile(Ui::WindowMain *ui,QStringList pathInfo)
{
   QSharedPointer<NoticeList> nL = QSharedPointer<NoticeList>(new
NoticeList() );

   try{
      MEDIA_STR mStr = this->canPlayFile(pathInfo);
      this->playFile(ui,mStr);
   }catch(QPair<QSharedPointer<Notice>,noticeTypeFlags> n) {
      nL->add(n.first,n.second);
      throw nL;
   }
}
```

Figur 6 Visar händelse förloppet för att spela upp en fil

4.2 Minneshantering

I C++ har man möjligheten att välja mellan smarta pekare och dumma pekare. En smart pekare lämnar inga ”*dangling pointers*” när referens objekt förstörs. En normal pekare kan lämna *”dangling pointers”* när referensobjekt förstörs (Wikipedia, 2016b). Som smart pekare har jag bestämt mig för att använda Qts *QSharedPointer*. *QSharedPointer* är en mall-klass som enbart kan erbjudas till klasser som är subklass till *QObject*. Den frigör det den pekar på när den går utanför sitt tillämpningsområde, men enbart om andra *QSharedPointer*-objekt inte har en referens till objektet. De enda som kommer att använda vanliga pekare är UI & template-objekt. Orsaken till mall-objekt inte kommer att använda *QSharedPointer* är att de inte kan bli subklass till QObject (The Qt Company , 2016b).

4.3 Dokumentation

Dokumentationen har gjorts i Doxygenformat. *Doxygen* är ett verktyg som används för att generera dokumentation. Det används i t.ex. C, Objektiv-C, C++, C#, Java, PHP, mf. Verktyget kan hjälpa en utvecklare genom att man kan generera en onlinedokumentation för webbläsare(HTML). Den kan även generera en offline referensmanual (LATEX). *Doxygen* kan även generera data i RTF(MS-Word), Postscript, hypertext PDF, komprimerad HTML och Unix manualsidor (Doxygen, 2016). Figur 7 visar normal *Doxygen*-notering och Figur 8 visar *Doxygen*-notering som modifierats för bruk med Qt.

```
/** Saves a BestOf file to path
 * @param errorLog - Choise to Save invalid files in seperate file [Default: true]
 * @throws QSharedPointer<Notice>
*/
```

Figur 7 Doxygen notering

```
/*! Saves a BestOf file to path
 * @param errorLog - Choise to Save invalid files in seperate file [Default: true]
 * @throws QSharedPointer<Notice>
*/
```

Figur 8 Doxygen notering för Qt

4.4 Språkstöd

Jag har valt att göra så att programmet stöder flera språk. För att kunna översätta text måste man först ladda en *QTranslator*. Figur 9 visar hur en *QTranslator* tas bort och en ny *QTranslator* laddas in. Först tas den gamla *QTranslatorn* bort från programmet med *qApp->removeTranslator()*. Sedan kombineras Trans, filnamnet och .qm för att få vad filen heter. För att kontrollera om filen kan laddas så används *translator.load()* som returnerar *false* om filen inte kan laddas in. Om filen hittas så laddas den in med *qApp->installTranslator()*.

```
void Translate::switchTranslator(QTranslator &translator,const QString
&filename) {
   qApp->removeTranslator(&translator);

   QString tmp = "Trans_"+filename+".qm";

   if(translator.load(tmp,this->v_langPath) == false) {
      qDebug()<<"Could not load "<<this->v_langPath+tmp;
      this->v_found = false;
      return;
   }

   this->v_found = true;

   qApp->installTranslator(&translator);
}
```

Figur 9 Byte av språk översättare

För att översätta en sträng måste man använda *QObjects tr()* funktion, till exempel *QObject::tr*(”Översätt detta”). Alla strängar som har funktionen *tr()* samlas ihop i en fil för varje språk som har angetts i projektets .pro-fil med *TRANSLATIONS*-makron genom att använda QTs externa verktyg *Liguist funktion lupdate*. Detta skapar en .ts-fil där man kan sätta in vad strängen ska översättas till. För att göra så att filen kan användas av Qt vid översättningar använder man verktyget *Linguist funktion lrelese* som skapar en .qm-fil.

5 SLUTSATSER

Projektet har resulterat i ett körbart program med namnet Playlist Manager. För programmering används C++ med Qt som IDE i enighet med den angivna metoden. Avgränsningar som angavs i inledningen.

De resultat som har uppnåtts är:

- Programmet vet hur många filer i en mapp som är av giltigt format och vet även hur många filer som är av ogiltigt format.
- Programmet kan visa lista med giltiga filer och kan även visa en lista med filer som inte är av giltigt format.
- Användaren kan byta namn på filer av giltigt format.
- Programmet kan kopiera filer från en mapp till en annan.
- Giltiga filformat kan spelas upp.
- Användaren kan välja mellan engelska och svenska som språk.
- Programmet har inte trådar och kommer inte ihåg inställningar från session till session.

Man kan inte välja ut specifika filer från en mapp och enbart kopiera den (Se kapitel 5.1 för förklaring).

Det finns ingen hållbarhetskontroll av filer som spelas upp (Se kapitel 5.1 för förklaring).

5.1 Problem

Efter ett tag blev programmet så stort att buggtestning blev en svår och komplicerad process. Detta problem lyckades jag aldrig riktigt lösa. Man skulle kunna separera lågnivå och högnivå i två subprojekt för lättare buggtestning.

Det var även svårt att hålla reda på vad som hade en översättningssträng och vad som inte hade det. Problemet blev aldrig riktigt löst i programmet. En lösning kunde vara att ha en

extern fil med alla strängar som ska användas i programmet och sedan manuellt checka av vilka som har satts in i en översättningsfunktion.

Det var även svårt att försöka komma på hur en *best of-fil* skulle implementeras. Jag kom aldrig på hur den skulle kunna implementeras och därför tog jag bort det från programmet helt och hållet.

Jag märkte även att när man hade bytt namn på en fil och den redan var på i mediaspelaren så agerade mediaspelaren som om filen hade samma namn som förut. Detta kan lösas med att koppla ihop en signal med mediaspelaren som säger till när filen bytt namn.

5.2 Framtid

För vidareutveckling och förbättring av programmet:

Man bör införa bättre hållbarhet för mediafiler som spelas upp. Med bättre hållbarhet menar jag att när en fil byter namn så uppdateras filen som spelas upp också vilket den inte gör nu utan den använder samma sökväg och namn som förut. Det som också skulle var bra att införa är att inställningar koms ihåg mellan programnedstängningar. Ytterligare förslag på vidareutveckling och förbättringar av programmet är att införa flera språk samt att förenkla kodupplägget för programmet. Slutligen vore det bra att byta namn på programmets filer med missvisande namn.

KÄLLFÖRTECKNING

Doxygen. (den 11 September 2016). *About*. Hämtat från Doxygen: http://www.stack.nl/~dimitri/doxygen/ den 20 Juli 2016

Jelsoft Enterprises Ltd. (den 18 November 2016). *Qt Centre*. Hämtat från Qt Centre Community Portal: http://www.qtcentre.org/ den 18 November 2016

Stack Exchange Inc. (den 11 November 2016). *stack overflow*. Hämtat från Top Questions: http://stackoverflow.com/ den 18 November 2016

The Qt Company . (den 20 Juli 2016). *QObject Class*. Hämtat från Qt: http://doc.qt.io/qt-5/qobject.html den 20 Juli 2016

The Qt Company . (den 20 Juli 2016b). *QSharedPointer Class*. Hämtat från Qt: http://doc.qt.io/qt-5/qsharedpointer.html den 20 Juli 2016

The Qt Company . (den 20 Juli 2016c). *QTranslator Class*. Hämtat från Qt: http://doc.qt.io/qt-5/qtranslator.html den 20 Juli 2016

Wikipedia. (den 29 Maj 2015). *Qt*. Hämtat från Wikipedia: https://sv.wikipedia.org/wiki/Qt den 7 November 2016

Wikipedia. (den 3 November 2016). *Qt_(software)*. Hämtat från Wikipedia: https://en.wikipedia.org/wiki/Qt_(software) den 13 April 2016

Wikipedia. (den 26 September 2016b). *Smart pointer*. Hämtat från Wikipedia: https://en.wikipedia.org/wiki/Smart_pointer den 8 November 2016