Beatrak

An Application for Automatic Beat Tracking and Cataloguing of Music Collections

Erik Jälevik

Supervised by
Andrew Gartland-Jones

Submitted as part of a
BSc in Computer Science
Department of Informatics
University of Sussex
April 2004

Candidate no: 38209

Statement of Originality

This report is submitted as part requirement for the degree of Computer Science at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

An application to obtain the IP rights to the software described herein has been made, but at the time of writing, the outcome is still unknown.

Erik Jälevik,
29 April 2004

Acknowledgements

Thanks to my supervisor Andrew Gartland-Jones, to all the knowledgeable contributors to the alt.comp.lang.learn.c-c++ newsgroup and the music-dsp mailing list, to the kind people who wrote, and gave away for free, the software libraries needed for the project, and above all, to all the talented producers and performers of all the amazing music out there, without which this project would not even have been conceivable.

Summary

This report describes the design and implementation of a program for analysing music to automatically determine its tempo. Such a program would be useful for DJs and music producers who would like to categorise music collections by tempo.

Chapter 1 gives some background information and establishes aims and objectives. These include making the beat detection as accurate as possible and producing a highly usable GUI. The program should be written in C++ and follow an object-oriented design methodology. The application should primarily be implemented for Windows but written in a platform-independent way to allow for easy porting.

Chapter 2 presents the results of research that was made into possible ways of achieveing the task. It describes two methods for beat tracking, by Scheirer and Dixon, and Scheirer's method is found the most promising. An overview of existing solutions and of tools and libraries needed follows.

Chapter 3 contains the requirements specification, which in more detail specifies what the application needs to do. Some likely scenarios are outlined and risks are identified.

A project plan with phases and tasks along with estimated durations and deadlines follows in Chapter 4.

In Chapter 5, the overall archtecture of the system is described. The package structure and the high-level class hierarchy of the system are presented and explained. As the design makes use of the design patterns Strategy, Singleton and Observer, these are explained and their role in the system described. There is also a section on common tactical policies which contains discussion on the use of pointers, exceptions, dynamic memory allocation and other issues that affect the system as a whole.

Chapter 6 goes into more detail about the design of individual classes, presents their class diagrams and screenshots of their visual appearance in the case of GUI components. An overview of the beat tracking process follows describing the steps and also some improvements made to Scheirer's algorithm such as the addition of phase constancy. Finally, the design and implementation of the audio streaming classes and the database classes are documented.

Chapter 7 presents the results of testing. The program passes almost all conformance tests and successfully determines the tempo of 19 out of a test set of 20 music excerpts of different styles. Finally, the beat-tracking performance of Beatrak is compared to that of the commercial DJ application Traktor and Beatrak is the winner.

Table of Contents

1. Introduction

Today, most music collections are catalogued and indexed on information such as artist, title, genre etc. This is satisfactory for many purposes but in certain situations, it would be useful to also be able to sort and search based on other properties of songs. One of those properties is tempo, and one of the situations in which it would be useful is when DJing by beat mixing tracks. This information can be obtained manually by counting the number of beats in a specific time period but it would be unfeasible, not to mention very tedious, to do this for a large collection of music. The proposed piece of software will perform this beat-tracking task automatically and store the results in a database.

1.1 Beat Mixing

In the early 1970s, a New York DJ called Francis Grasso was the first to start experimenting with synchronising the tempo of two records and playing them back simultaneously (Kempster, 1996). To do this, he needed to find two records with roughly the same tempo, pitch shift one to match the tempo of the other while monitoring it in headphones, and release it at exactly the right moment to achieve a smooth, synced mix. This is the foundation of the technical side of DJing and is known as beat mixing.

Because tempo is the major (but not the only) determining factor in whether two records can be mixed or not, a DJ always needs to be aware of the tempi of particular tracks. The reliance on beat mixing has become so prevalent that in modern dance music, genres are often defined on the grounds of tempo. Therefore, a DJ that sticks to playing a particular style, like house or drum'n'bass, can be fairly certain that most tracks within that genre will be beat-mixable. However, for anyone wanting to be a little more adventurous and beat mix different genres as well as older music where tempo is not such a determining factor, a fully tempo-analysed collection would be an extremely useful thing to have. It would hopefully aid people in being more creative and eclectic in their mixing, as it will be possible to instantly check what other tracks in a collection, regardless of genre, have a similar tempo to the one currently playing.

1.2 Aims and Objectives

The objective is to implement a GUI application which can detect the tempo of pieces of music. This information will, along with other user-specified or automatically downloaded information about a track, be placed in a database for subsequent access via the program's user interface. The program should be able to take audio in various formats and output the tempo or tempo ranges of the music. The application should be able to do batch analysis of a music collection stored in a digital format but I also hope to implement a real-time feature for beat tracking of music from other sources, like vinyl.

My main aims are trying to get the beat detection as accurate as possible, and to create a highly usable and appealing interface. Further aims include a modular and well-designed piece of

software, where the beat-tracking functionality could be easily detached and reused in some other context. Ideally, the program should be implemented in a platform-indepedent way, so that it will run on both Windows and Linux.

1.3 Methodology

The program will be designed and implemented using sound object-oriented software engineering methods. The general process will be iterative in the sense that previous stages might be revisited in the light of new insights found during a subsequent stage. UML will be used where helpful to clarify and specify the workings of the system. For the user interface, usability guidelines and principles will be followed as much as possible and I also hope to be able to carry out some usability testing. Since audio processing is a computationally demanding task, the software will be written in C++ for reasons of efficiency. I also intend to release the code as open source on completion of the project so therefore open-source components and libraries will be favoured over closed-sourced ones as much as possible.

As an aid in documenting the progress of the project, I have set up a web site devoted to it at http://homepage.ntlworld.com/erik.jalevik/proj. All project documentation is available on the site, and a log of work done has been kept. The music excerpts used as a test set (see Section 7.2.1) are also available for download.

1.4 Constraints

As with all projects of this type, the major constraint is time. I only have a limited amount of time to finish the program and therefore, only a limited amount of functionality can be implemented. This is something I have tried to take into account by dividing the requirements in the specification in Section 3 into core features and extended features. The core features are to be seen as the minimal specification, whereas extended features are extras that will be implemented if time permits.

Another obvious constraint is my own knowledge of the subject areas and technologies involved. As digital signal processing and C++ programming are not subjects taught as part of the Computer Science degree, I have needed to acquire much of this knowledge in other ways to be able to undertake this project.

2. Domain and Requirements Analysis

The first phase of the project involved carrying out research into beat tracking and tools needed to do the job. The outcome of this is presented in this chapter.

2.1 Previous Work

The problem of automatically determining the tempo of recorded music, or beat tracking, has been an active area of research in recent years. Beat in this context is defined as the regular pulse to which a human listener would tap along. This might seem a simple problem due to its apparent intuitiveness, but just like other basic human cognitive abilities, it is surprisingly hard to simulate in a computer program.

Most of the early work in the field assumed prior processing of the audio signal into symbolic form, such as MIDI, before beat tracking was attempted. As the current system will need to deal with acoustic audio signals, these approaches will be of little help. The earliest attempt to track tempo from audio signals was a percussion transcription system by Schloss (1985, cited in Dixon, 2001a). Of limited appeal, it was superseded by several systems developed by Goto and Muraoka (1995, 2001). Their first solution required the music to contain drums, but a later version could also be applied to music without drums. However, both of their systems make many assumptions about the music, such as a 4/4 time signature, specific placements of bass and snare drums and the likelihood of chord changes being greater on downbeats. This severely narrows the styles of music they can successfully analyse.

The two most promising approaches I found were by Scheirer (1998) and Dixon (2001a). Scheirer has proposed a system which makes no assumptions about style of music, presence of drums or any other particular feature apart from the presence of a clearly discernible rhythmic pulse. A schematic overview of the system can be found in Figure 2.1. The system works in real-time and is capable of following tempo changes in the signal. The method involves splitting the signal into a number of frequency subbands on which the amplitude envelopes are calculated. The signal is then downsampled since only activity at the frequency of individual beats is of concern. The derivatives, i.e. the slopes, of the envelopes

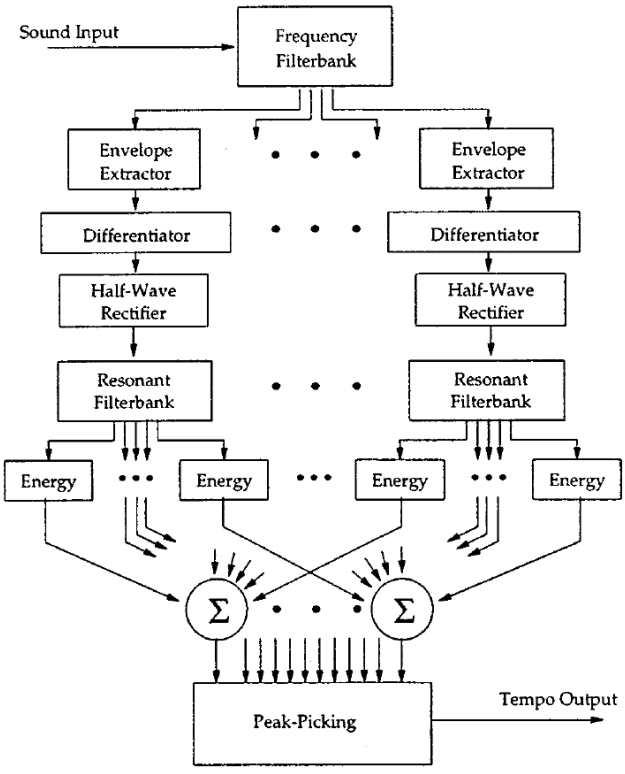

Figure 2.1. Schematic of Scheirer's algorithm (Scheirer, 1998).

are taken and peaks in the resulting signal can be interpreted as beat onsets but aren't explicitly assumed as such by the algorithm. Next, each subband is fed through a bank of resonators, implemented using comb filters, each tuned to a particular delay time. These delay times correspond to the tempi that can be detected, and one resonator is used for each detectable BPM value in a predetermined range. The outputs from these resonators are then summed across frequency subbands. The resonator frequency with the highest output, or energy, is taken as the tempo of the signal. A further step works out the phase of the beat but since this will not be needed for the current system I won't describe the procedure.

Dixon's beat-tracking system uses a different approach and operates in two stages. The first step is called the tempo induction stage and the second the beat-tracking stage. Again, I will only describe the tempo induction step. To detect onsets, the signal is high-pass filtered and smoothed by calculating the average absolute value within a window of the signal. Linear regression is used to determine the slope of the envelope and onsets are deduced by peak-picking in this signal. In the next step, the whole extract is examined for inter-onset intervals of up to around 2 seconds. Intervals with other onsets in-between are also considered. Each distinct interval (give or take about 20 ms) found by the algorithm is assigned to a cluster and in the end, the number of elements in each cluster is used to give the cluster a score. Each cluster score is adjusted by adding scores of clusters related by an integer multiple to the current cluster. This reflects an expectation that for a cluster representing the beat, there will also be clusters for integer fractions and multiples of the tempo. The beat interval of the highest scoring cluster will then be the most likely tempo. The main difference between Scheirer's and Dixon's methods is that the latter needs to read the entire input before detecting the tempo so cannot be used in real-time. Another drawback of Dixon's method is that it is not well equipped to handle tempo changes. A piece of music will need to keep a relatively steady tempo throughout for the clustering algorithm to work.

Both Scheirer's and Dixon's algorithms have produced quite impressive results but I decided on using Scheirer's algorithm since I would like the ability to track the tempo in real-time and also due to its better handling of tempo changes. It might be possible to implement Dixon's algorithm as an extension or maybe try to combine the two. For instance, using Scheirer's envelope extraction with Dixon's clustering algorithm could be worthwhile for batch processing of songs. Further possible tweaks include adding high-level information to Scheirer's algorithm to improve the accuracy and tuning the number of frequency subbands and resonators for faster execution speed.

2.2 Existing Solutions

The initial idea for this project was born when I one day went looking for software that would automatically tempo-analyse my music collection. As I couldn't find any freely available programs capable of this task, I decided to write one. Similar programs are however available. Most of these are of the “digital DJ” type, i.e. an attempt to emulate the classic DJ setup of two turntables and a mixer in software with MP3s instead of records. Some of these provide tempo detection as part of the package.

One example of the above is Native Instruments' Traktor, one of the best-known digital DJ programs, a screenshot of which can be found in Figure 2.2. It's a professional and sleek program which allows you to import and beat track a music collection and then mix tracks using a virtual mixer and decks. The beat tracking works reasonably well but seems to get confused on music without a clearly pronounced beat. Judging by the examples on Scheirer's web site, his algorithm should be able to perform better on music without strong beats. It also has no way of accounting for tempo changes, it simply stores one tempo value as the BPM of a track. Many other examples of this type of software exist but none that I have found are free. It should also be pointed out that the scope of these applications is much greater than the proposed system. A fully featured digital DJ program like Traktor would be much beyond the scope of a 3rd year project. These applications also have a slightly different aim in that they try to automate the mixing process in some way or

Figure 2.2. Screenshot of Native Instruments' Traktor.

other, whereas I'm more interested in providing an aid for DJs who enjoy mixing the old-fashioned way.

Dixon has implemented a version of his beat-tracking method called BeatRoot (Dixon, 2001b), shown in Figure 2.3. This is completely different from the programs mentioned above as its main aim is to help analyse tempo and timing of musical performances. It creates a visualisation of a piece of music with markers at the estimated beat times. Although freely avalable, it is not suited to the current aims as it provides no means for batch analysis or for building up a database.

2.3 Tools and Libraries

A range of different tools and libraries will be needed: audio libraries for handling audio I/O and processing, libraries for decoding MP3 files and reading from CD, a GUI framework for creating the user interface and a file-based embedded database engine for storing and querying the collection.

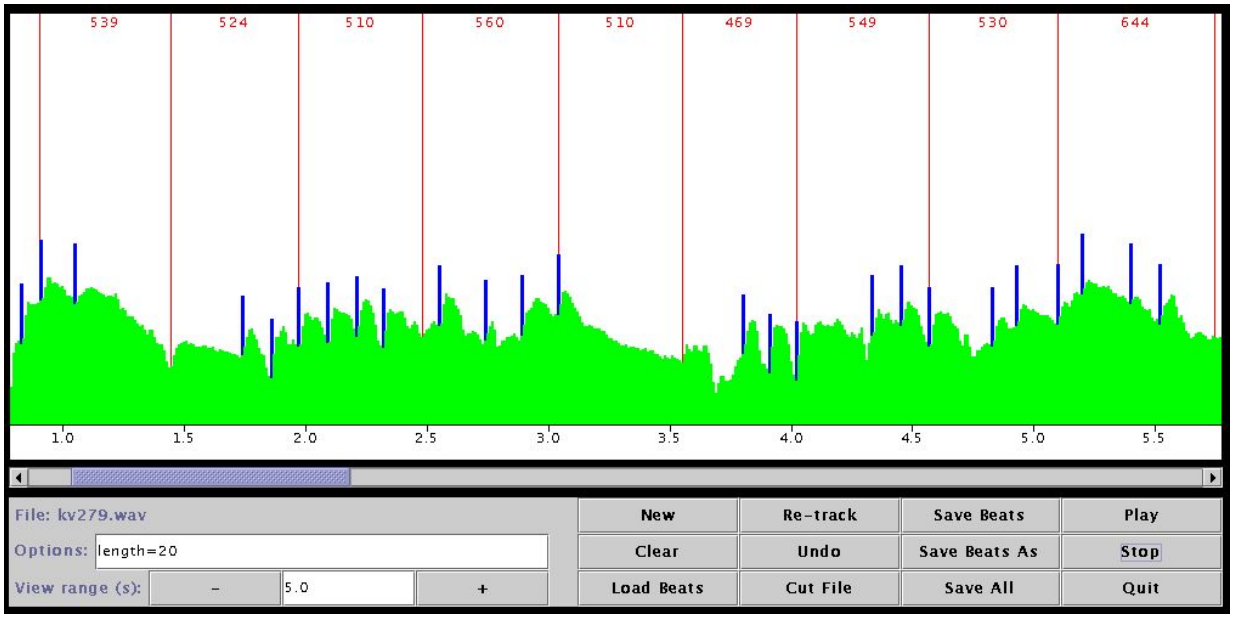

Figure 2.3. Screenshot of Dixon's BeatRoot.

2.3.1 Audio Libraries

There are several open-source and cross-platform libraries available for handling various aspects of audio processing. Annoyingly, there is no single library that includes all the audio features required by the project. The libraries I have looked at include PortAudio, libsndfile, Sig++ and the Synthesis Toolkit in C++ (STK). From browsing their APIs and reading their websites, STK seemed to be the most suitable. It includes a fair amount of the features I need, such as WAV I/O, real-time I/O and filters, is fully cross-platform and is written in an object-oriented manner in C++. Initial tests with WAV file I/O has indicated that the library works well and is easy to use.

Within the audio programming world, reading MP3 and CD audio is considered separate from the areas handled by the above-mentioned libraries, so unfortunately none of the above include this functionality. It has been problematic to find a free MP3 library due to the many patents surrounding the format. One promising candidate does however exist, called MAD, and it has reportedly been used in many software projects. When it comes to CD audio extraction, I have only come across one free library, AKRip, which unfortunately is Windows-only.

2.3.2 GUI Frameworks

As the application needs to be as portable as possible, I needed a GUI framework designed to be cross-platform. This ruled out Microsoft's MFC, which otherwise would have been the obvious choice for Windows. It also ruled out GTK, a popular GUI framework on the Linux platform, because its Windows support is still patchy. Of the remaining ones, I looked more closely at Qt and wxWidgets (which was called wxWindows when I started the project and at the time of writing the interim report, but which has since changed its name to wxWidgets due to pressures from Microsoft), both of which are truly cross-platform. I decided on wxWidgets in the end mainly due to cumbersome licensing restrictions for the Windows version of Qt. Further reasons for choosing wxWidgets include the fact that it uses native widgets regardless of platform and that it has been in development for a long time and thus ought to be reasonably stable.

2.3.3 Databases

Most databases follow a client-server model where the database engine runs in a separate process as a server and requires client programs to connect to it to retrieve data. Well-known database

systems like Oracle, DB2, MySQL and PostgreSQL all follow this model. This sort of database would however not be suitable for the project at hand. What is needed is a compact, local file-based database that can be embedded within the application process but still be powerful enough to provide fast searching and filtering of records. In researching this, I found three systems fitting the description, SQLite, Berkeley DB and Metakit. Of these, SQLite is the only one that provides an SQL interface to the database, the others are based on more unconventional ways of querying the data. For reasons of familiarity with SQL and ease of implementation, I will use SQLite even though it has a fairly clunky and old-fashioned C-style API.

2.4 Target Users

The target users are likely to be either DJs or remixers/producers, hobbyist or otherwise. Although it is likely that most users fitting this description will be reasonably computer-literate, this can't be taken for granted. The best way to address this will be to make the user interface conform as much as possible to current GUI standards. It is also very likely that use of the program will be occasional. Therefore, it will be important to pay attention to memorability, i.e. it should be easy to come back to the program and remember how to use it after a period of not having used it.

3. Requirements Specification

The following list of requirements was produced by carefully considering the intended purpose and scope of the software. It is meant to capture what features would be useful for both building the catalogue and using it to its full potential during mixing. Each class of requirement is divided into a section of core requirements that should be implemented and a section of extended requirements that will only be implemented if time permits.

Due to space constraints, full descriptions of each requirement have been omitted. The full specification including a description and motivation for each requirement is available on the project website.

3.1 Functional Requirements

3.1.1 Core Features

The program shall:

001. Track the tempo of recorded music
002. Track tempo changes
003. Process WAV files
004. Process CD audio
005. Process MP3 audio
006. Store information about analysed songs in an internal database
007. Allow manual data entry of artist
008. Allow manual data entry of title
009. Allow manual data entry of genre
010. Automatically download CD song information where possible
011. Automatically determine MP3 song information where possible
012. Automatically store song source information
013. Allow batch processing of files
014. Display database contents to user
015. Allow sorting of tracks based on tempo and multiples thereof
016. Allow filtering of tracks based on tempo and multiples thereof
017. Allow more complex filtering based on other attributes
018. Allow manual verification of tempo
019. Provide a “tap” function for manual tempo detection
020. Allow customised display of columns in database view
021. Allow resizing of columns in database view
022. Remember personalisation settings between sessions
023. Provide an export to CSV function
024. Allow cancelling of beat-tracking
025. Allow editing of data in collection view
026. Allow deletion of data in collection view

3.1.2 Extended Features

If possible, the program shall also:

050. Detect tempo in real-time
051. Be able to determine success or failure of beat tracking
052. Achieve better accuracy through the use of several algorithms
053. Extract breakdown information
054. Detect starting point of drums
055. Automatically download song information based on fingerprints
056. Build a central Internet database of tempo information

3.2 Data Requirements

At least the following pieces of data about each song will need to be stored in the database:

Artist (string)
Title (string)
Length (time)
Format (string)
Genre (string)
Source (string)
Tempo (array of integers)
Tempo verified (boolean)
Date analysed (date)

Most of the above should be self-explanatory but Source might need further explanation. It's intended to hold the source from which the particular song was analysed. In the case of a CD track, it will hold the name of the album. In the case of an MP3 or a WAV, it will hold a pathname to the file. In the case of real-time audio, this field will be user-defined.

Depending on extensions implemented, more fields might become required.

3.3 Performance Requirements

3.3.1 Core Features

The program shall:

200. Process songs faster than their playing time
201. Detect tempo to within 1 BPM's accuracy
202. Detect tempi between 80 and 200 BPM
203. Hold up to 100,000 songs without significant performance reduction

3.4 Operational Requirements

3.4.1 Core Features

The program shall:

300. Run on Windows
301. Run in a minimum resolution of 640x480

3.4.2 Extended Features

If possible the program shall also:

350. Run on Linux

3.5 Usability Requirements

3.5.1 Core Features

The program shall:

400. Provide contextual help in the form of tooltips
401. Provide keyboard shortcuts for the experienced user
402. Provide feedback about beat-tracking process
403. Use consistent terminology throughout
404. Allow TAB navigation in all dialogs
405. Have a default button in all dialogs
406. Have an online help system

3.6 Look-and-Feel Requirements

3.6.1 Core Features

The program shall:

500. Look modern and sleek
501. Have a native look-and-feel

3.7 Other Requirements

3.7.1 Core Features

The program shall:

600. Be designed so that beat-tracking algorithm(s) can be reused

3.8 Scenarios

3.8.1 Analysing the Tempo of a Single CD Track

Homestar is a bit of an indie fan and would like to find out the tempo of “Girl Afraid” by The Smiths which he has on the “Hatful of Hollow” album. He clicks the button on the main toolbar for adding from CD. He clicks the FreeDB button to automatically download the song titles. As so often is the case, the genre comes back wrong from FreeDB so he selects “Indie” from the Genre drop-down box. He then ticks the box next to track 13 and clicks OK. The program will fill the artist, title, format, length and source fields with the information retrieved from FreeDB and the CD. A progress dialog appears with an indicator of how long the analysis is going to take. A short while later, the progress indicator disappears and the newly analysed track appears in the collection view. Homestar can now see the entry for the song in the collection view highlighted in red as the tempo has not yet been verified. The tempo reads 176 which is roughly in the range he expected so he double-clicks the line to open the track information dialog and verifies the tempo by clicking on the Verify button next to the tempo range. After clicking OK and returning to the collection view, the song is now displayed in black.

3.8.2 Displaying All Songs Between 115 and 125 BPM

Dribcot has already catalogued his collection and is in the middle of an old school house mix. He's just put on Mr Fingers' “Can You Feel It” and would like to mix it into something more interesting than just another Chicago classic. So by using the filtering function available in the collection view, he specifies that he wants to to only show tracks at 120 BPM +/- 5. After clicking Filter, he is presented with a view of all tracks starting at 115 BPM and going up to 125 BPM. By doing this, he discovers a King Tubby track classified as 122 BPM (although its real tempo in fact is 61 BPM but this falls outside the range of the detection algorithm) which he proceeds to mix into the Mr Fingers track. As reggae doesn't in general tend to be the same tempo as house, this particular combination would probably not have crossed Dribcot's mind had he not had the database.

3.8.3 Batch Analysis of All MP3s in a Folder

Hallon has a large MP3 collection on her hard drive and wants to feed it through the program to build up a database. She clicks the button for adding files on the toolbar and is presented with the Add Files dialog. In the tree view of the folder hierarchy she ticks the box next to the root folder of her collection. In the dialog are also options for how to deal with the names and genre of the batch-analysed tracks. She chooses to let the program extract this information from the MP3s' ID3 tags. She is then ready to start so clicks the OK button. The progress window displays a list of all the scheduled tracks and a progress bar for the current track being analysed. The tracks will be added one by one to the database as they are analysed. Once all tracks are analysed, the progress window disappears and Hallon starts browsing through the newly created database.

3.9 Risks

There are three major risk factors greater than any other:

- Successful tempo detection rate is too low
- Accuracy of tempo detection is too low
- Speed performance of algorithm is not good enough

Unfortunately, there is no straight-forward way to assess these without a running implementation of the beat-tracking algorithm. But these risks have been acknowledged within the project plan by

scheduling the writing of the beat-tracking code as early as possible. This way any potential issues will be discovered early and there will be time left to investigate alternative strategies.

Other risk areas are the third party libraries I will be using. They might lack some of the features needed or they might not be implemented well. To address these risks, I planned to do some testing of the libraries during the initial stages of the project. However, due to lack of time, I decided to skip some of this testing to be able to get on with more pressing work. In any case, I don't consider these risks as significant as the ones mentioned above, since libraries mainly deal with well-established and well-understood concepts. Therefore, it should for the most part be possible to find alternatives or workarounds in case of problems. The chosen libraries have also been used by thousands of developers around the world, so should in all likelihood be adequate.

4. Project Plan

This chapter describes the specific tasks that the project will involve.

4.1 Milestones

The major project deadlines are as follows:

- 23 October 2003: Project Proposal
- 11 December 2003: Interim Report
- 11 March 2004: Draft of Final Report
- 29 April 2004: Final Report

4.2 Phases and Tasks

A breakdown of phases and tasks with deadlines and estimates of their duration has been drawn up. It should be pointed out that although the plan is laid out in a sequential "one-task-after-another" way reminiscent of a waterfall design process, the actual process followed will be iterative. This means that there will be considerable overlap between the design, implementation and testing phases. It also means that development during these phases will be cyclic, e.g. it might be necessary to go back and do some more detailed design after some implementation has been carried out.

4.2.1 Research and Analysis

Task	Description	Duration	Deadline	Completed
Research beat tracking, DSP, GUI frameworks and audio libraries	Research methods for beat tracking and how they work. Acquire required knowledge of digital signal processing. Research cross-platform GUI frameworks for C++. Research audio libraries for C++.	several weeks	23/10/03	23/10/03
Write proposal	A document proposing the project needs to be written and handed to supervisor.	3 hours	23/10/03	21/10/03
Write requirements specification	Draw up requirements in more detail.	2 days	30/10/03	30/10/03
Write project plan	Write a project plan including phases and tasks with deadlines and estimated durations.	5 hours	30/10/03	30/10/03
Prototype GUI	Sketch out different versions of main GUI components and work out what components will be needed.	1 day	31/10/03	31/10/03

Test wxWidgets for needed features	Write some test code for wxWidgets to evaluate its support for the kind of components needed for the GUI.	2 days	5/11/03	skipped
Determine needed components for Scheirer's algorithm	Study Scheirer's paper to understand exactly what DSP building blocks will be needed.	2 days	6/11/03	8/11/03
Test STK for needed features	Write test code for the STK audio library to evaluate its support for the components needed for Scheirer's algorithm.	1 day	10/11/03	skipped
Test STK for audio I/O features	Install and test the STK audio library to evaluate its support for general audio I/O like reading from files.	1 day	11/11/03	28/11/03
Research libraries for reading audio from CD and MP3	Need to find libraries for these tasks as they are not supported by STK.	1 day	14/11/03	21/11/03
Research file-based database systems	Find out what file-based databases might be suitable.	5 hours	18/11/03	20/11/03
Test libraries for reading audio from CD and MP3	Briefly test chosen libraries.	5 hours	24/2/04	13/1/04
Test file-based database system	Briefly test chosen database.	5 hours	3/1/04	17/1/04

4.2.2 Design

Task	Description	Duration	Deadline	Completed
Produce high-level class hierarchy	Produce first iteration of the general architecture of the system.	1 day	24/11/03	24/11/03
Analyse class interaction	Produce first iteration of what interactions are needed between classes. Sequence diagrams might be a helpful notation.	1 day	26/11/03	9/12/03
Produce lower-level class hierarchy	Produce first iteration of lower-level design by fleshing out the class hierarchy with important methods and attributes.	2 days	1/12/03	9/12/03
Design data model	Design a data model for the storage of tracks and their associated information.	2 hours	1/12/03	21/12/03
Write interim report	The interim report should be handed in at the end of the autumn term.	2 days	11/12/03	10/12/03

4.2.3 Implementation

Task	Description	Duration	Deadline	Completed
Set up bug database	Before starting to code, I need to have a way of recording any bugs I come across during development. This will probably just be a spreadsheet or something.	1 hour	22/12/03	21/12/03
Implement beat tracker	Write the classes to do with the actual beat tracking. Start by getting WAV support to work and test and tweak algorithm on suitable tracks.	1.5 weeks	2/1/04	2/1/04
Implement database classes	Write the classes to do with representing database records in the program and also all the operations needed on them.	1 week	16/1/04	27/1/04
Implement GUI	This includes writing code for the collection view, toolbar, menus, add single file dialog, add multiple files dialog, add CD dialog, tapper, single file progress window and batch progress window.	3 weeks	20/2/04	24/3/04

Implement tapper	Write the feature that lets you determine tempo by tapping along to a track.	2 days	24/2/04	6/3/04
Implement FreeDB support	Write the code necessary to query the FreeDB service to retrieve artist/title information for CDs.	1.5 days	27/2/04	29/2/04
Implement file name parsing/ID3 support	Write the code necessary to retrieve artist/title information from file names or ID3 tags.	1 day	2/3/04	1/4/04
Implement Export to CSV function	Write code for exporting database contents in CSV format.	5 hours	5/3/04	19/4/04
Design icon	It'd be nice to have a unique logo for the program to use as an icon.	1 day	7/3/04	24/3/04
Write draft report	The draft report should be handed in at the end of the autumn term.	1 week	11/3/04	23/4/04

4.2.4 Documentation

Task	Description	Duration	Deadline	Completed
Write help system	Write user documentation in the form of an online help system.	2 days	24/3/04	18/4/04

4.2.5 Testing

Task	Description	Duration	Deadline	Completed
User testing	Have some potential users test the interface and comment on its usability.	1 day	31/3/04	not done
Functionality testing	Test finished program for conformance to requirements specification and fix bugs.	1 week	1/4/04	20/4/04
Write final report	The final report should be handed in in the second week of the summer term. Hopefully, most of the work should have been done when writing the draft report.	3 days	29/4/04	27/4/04
Prepare presentation	Prepare for presentation that takes place in the summer term.	2 days	10/5/04	

5. System Architecture

Two of the main goals of any object-oriented system are modularity and clarity of design.

Why is modularity desirable? It facilitates reuse by dividing the system into more or less independent subsystems. It will make a complex system easier to understand for the same reason; an ordered, hierarchical structure of parts comprising a whole is much easier to make sense of than a flat and highly tangled structure of intertwined parts.

Why is clarity of design desirable? Although somewhat surprising at first, the fact is that most code is read more than it's written (Eckel, 2004). Not only by other people, but also by yourself at some later point in time. If the system has a clear and well-thought out design and this is reflected in the code, it stands a much greater chance of being maintained and improved upon in the future.

This chapter will describe my efforts to make the current system modular and well designed.

5.1 High-Level Structure

A venerable design strategy for GUI applications is the model-view-controller (MVC) pattern, which made its first appearance in Smalltalk in the early 80s (Burbeck, 1992). The MVC pattern enables you to partition the system into three independent parts: the model, which contains the actual application logic and data, the view, which is the windows and widgets on the computer screen, and the controller, which takes care of the mapping between user actions and model functions.

As it seemed the most sensible way to layout a GUI application, this was the model I based the initial high-level designs on. However, as the design progressed further, it became apparent that wxWidgets, the GUI toolkit, was not particularly suited to the MVC pattern. Its approach couples the view and the controller more directly than pure MVC does. Each GUI widget, like a window, a dialog or a button, is a subclass of wxEvtHandler, the base wxWidgets controller class. This means that each widget can have its own event table, essentially acting as the controller for the functions that can be invoked from it. It can also refrain from defining an event table, in which case the events it generates get propagated up to its parent control/event handler.

The drawback of this is that it makes it harder to replace only the view, or only the controller, with something else. However, as it's highly unlikely that this will ever be necessary for this particular application, I decided to go with the wxWidgets way of doing things instead of forcibly squeezing it into the MVC mould.

5.2 Packages

This leads us to the high-level package diagram (Figure 5.1) showing the top-level package structure of the system and the major classes within each package. The view and controller classes

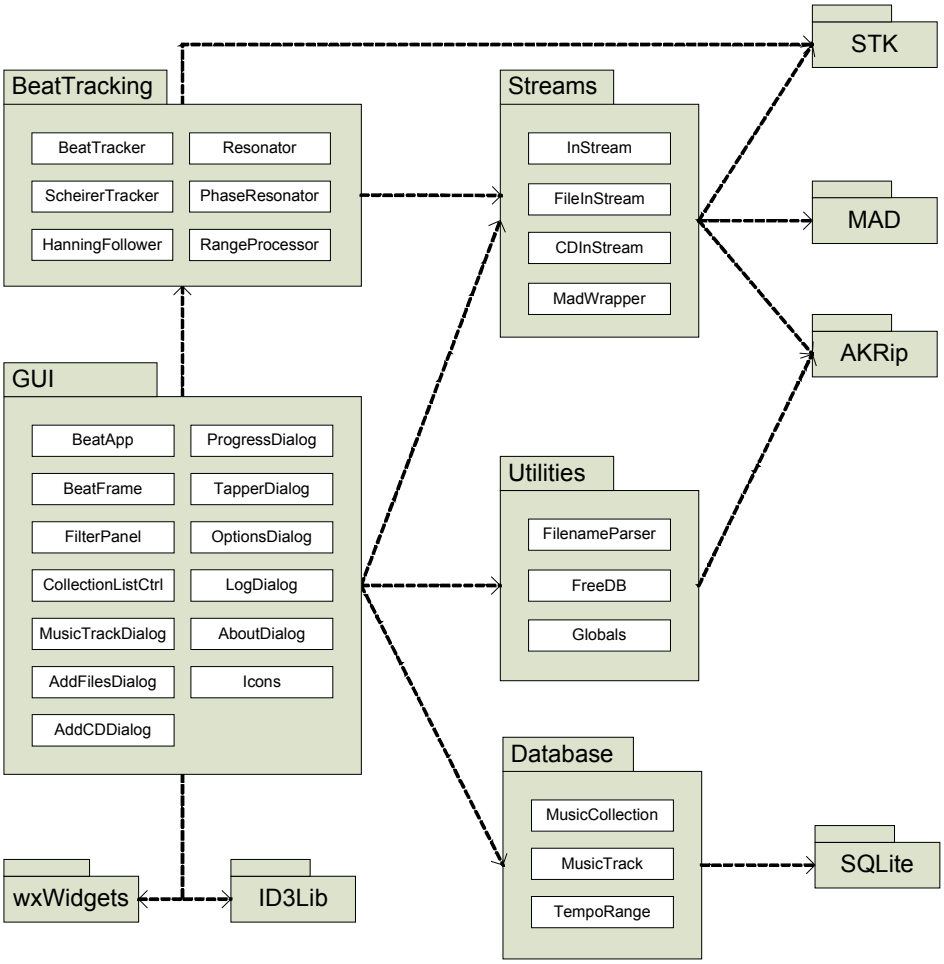

Figure 5.1. Package structure of the system with major classes in each package shown.

belong to the GUI package on the left in the diagram. The model classes are divided into four packages: BeatTracking, Streams, Database and Utilities. External libraries are also shown along with the dependencies on them.

One of the requirements was that the beat-tracking classes should be reusable. This has been achieved by making all the model packages completely independent of wxWidgets. All use of wxWidgets-specific functionality is constrained to classes in the GUI package so that none of the functionality of the model is dependent on wxWidgets. The arrows in the diagram indicate directions of dependency. It's also evident that the model packages are independent of each other, with one exception. The beat tracking package needs the audio streams to be able to work so these will always have to be distributed together.

5.3 Class Model

Figure 5.2 shows a more detailed top-level diagram including all the classes in the system. The heart of the system is the BeatFrame, which is the application's main window frame and also contains the top-level event table for the GUI. The MusicCollection class is the interface to the database through which all adding, deleting and updating of music tracks occur. The different stream classes are responsible for reading different types of audio and the BeatTracker and subclasses take care of the beat tracking. Finally, the utility classes mainly provide functionality for retrieving metadata, such as parsing filenames and accessing FreeDB for CD information.

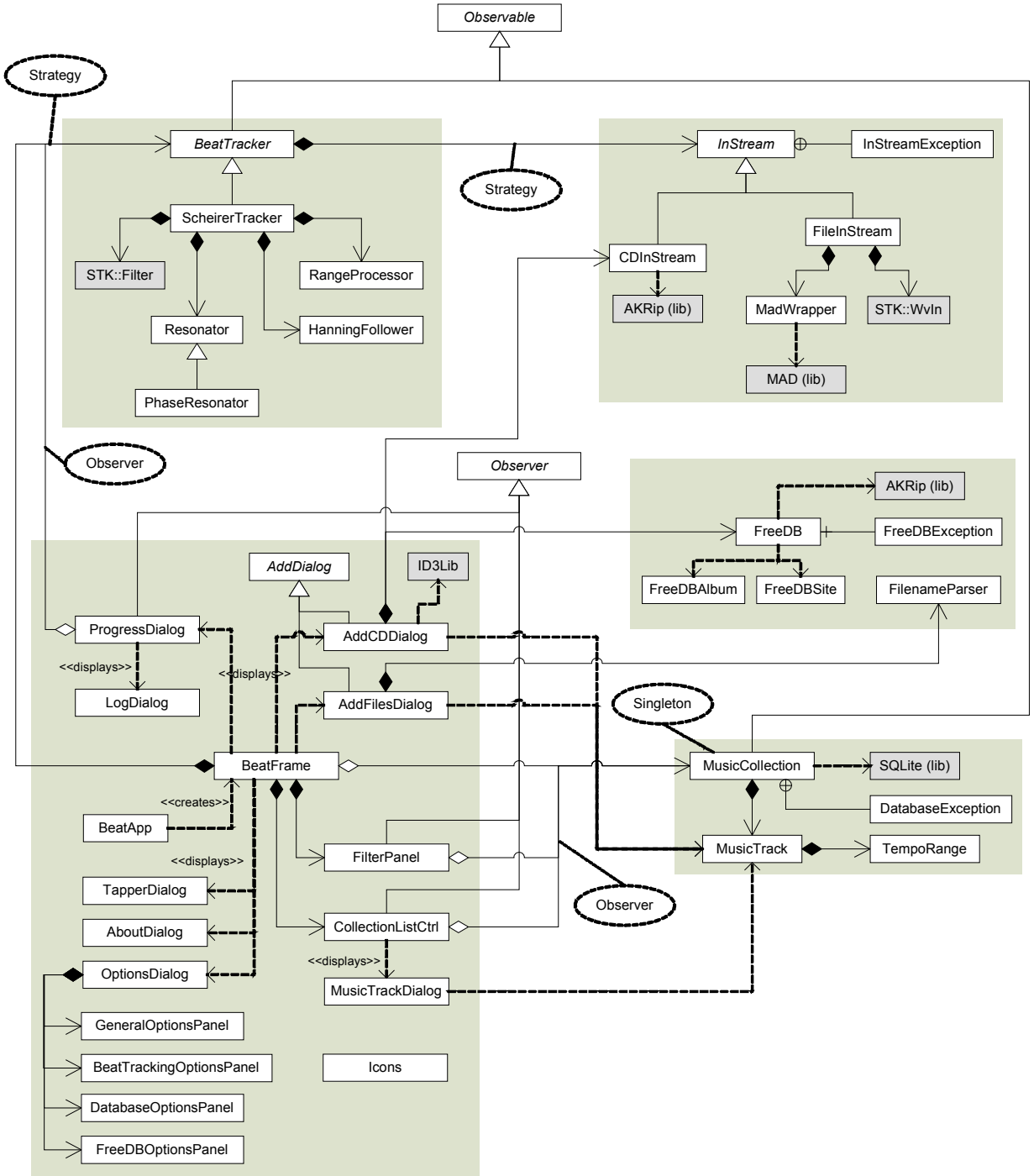

Figure 5.2. Class model.

In designing the class model, an important thing to bear in mind is what's likely to change and where we might want to extend the system in the future. An obvious candidate for extension is the beat-tracking algorithm. We might want to add different beat-tracking algorithms using some different strategy for detecting tempo in the future. For this reason, the abstract BeatTracker class was created to provide a common interface for all future beat-tracking algorithms to inherit from. Any client of the beat-tracking package can then be written to interact with a BeatTracker object and it is guaranteed to work with any future beat trackers who adhere to the interface of the base class.

The same thing applies to the audio streams. The BeatTracker needs an audio in stream to read its audio samples from. In the current system, audio in WAV, MP3 or CD format is catered for, but it's very likely that support for other formats will be added in the future. To this end, the abstract InStream class provides the basic stream interface that all streams will need to adhere to. The subclasses CDInStream and FileInStream work very differently internally, but to a client who only cares about getting streamed audio samples, they can be treated as one and the same.

5.4 Use of Strategy

The design of the beat trackers and the audio streams described above conform to the Strategy design pattern (Gamma et al., 1995). Strategy is a way of defining several different strategies for achieving a task and making them interchangeable. The BeatTracker class hierarchy comprises a family of strategies for beat tracking and the InStream hierarchy comprises a family of strategies for reading audio.

The design includes a double application of Strategy. The BeatTracker, although being an implementation of Strategy itself, is also the client (or Context as the Gamma book calls it) of the Strategy class InStream. This achieves total modularity between the beat-tracking algorithms and their way of receiving audio data. The streams don't need to know about additions of new beat-tracking algorithms and the beat trackers don't need to know about the addition of new streams. The beat tracker does however need to be configured with the correct stream to use by someone, and this role is played by the controller BeatFrame.

5.5 Use of Singleton

The MusicCollection class represents the interface to the database storing all the music tracks. It wouldn't make sense to ever have several objects of this class running within the system as it can only interact with one database file at a time. By using the Singleton design pattern, it's possible to enforce this restriction. The idea behind Singleton is to make the constructor private, and only provide access through a static Instance method that returns a pointer to the only instance of the class. The first time Instance is called, the object is created and returned. On subsequent calls, only a pointer to the already existing object is returned.

In this way, the Singleton itself controls how many instances of it can be created. The use of a static method to access the single instance also solves the problem of many different classes in the GUI needing easy access to the collection.

5.6 Use of Observer

Normal usage of the application will include a lot of viewing, adding, updating and deleting of tracks. The actual data is held in the model class MusicCollection and is displayed in the main window by the view class CollectionListCtrl. New tracks are added by the controller in the ProgressDialog as the result of new beat-tracking operations. Various other classes may sometimes also change the state of the database. One of the main issues when using a view decoupled from the model is how to keep the view updated and consistent with the model data.

Enter the Observer pattern. It describes two abstract base classes called Observer and Observable (Gamma calls it Subject but I find the name Observable much more intuitive). Any class derived from Observable can be "observed" by any class derived from Observer. Observer objects register their interest in an Observable by calling its Attach method passing a reference to themselves as

an argument. All the subclass of the Observable then needs to do is call the base class method Notify whenever its state has changed. The Observable base class then sends a message to its registered Observers by calling Update on each one. The Observers, who know about the Observable, can then in turn call methods on the Observable to find out about its new state.

In Beatrak, MusicCollection is an Observable, and CollectionListCtrl an Observer. This way, the model can stay decoupled from the view and still be synchronised at all times. Another application of the Observer pattern is between the BeatTracker and the ProgressDialog in the GUI. As beat tracking is a time-consuming process, a progress bar is shown when beat tracking is underway. For the progress indicator to be accurate, it needs information about the BeatTracker's current state. To solve this, we make the BeatTracker an Observable and the ProgressDialog an Observer of it.

There is one small problem with Observer, however, and that is that it requires abstract base classes to be inserted into the class hierarchy. What if our classes are already part of a hierarchy? This is the case for the GUI Observer classes described above; they are both part of the wxWidgets framework. CollectionListCtrl, for example, is derived from wxListCtrl and ProgressDialog from wxDialog. Since we can't change the design of the wxWidgets library itself we are forced to use multiple inheritance in order to make use of the Observer pattern.

Most authorities on the subject recommend against using multiple inheritance since it can give rise to ambiguities. This is only the case for implementation inheritance however; in the case of Observer, all we are inheriting is an interface. Observer only contains a pure virtual function, i.e. a function without a body, so it can safely be used in a multiple inheritance scenario. An abstract C++ class with only pure virtual functions is the same thing as an interface in Java, and interfaces never give rise to ambiguities when several are implemented by the same class.

5.7 Common Tactical Policies

As the application spans about 40 classes and pulls together several different libraries including non-object-oriented C ones, it was important to develop a set of policies to ensure consistency across the system.

Exceptions really start to make sense when building GUI applications with several layers, so it seemed like a good idea to use them to deal with error handling. Exceptions are however not very widely used in C++, both because they are a fairly recent addition to the language and also because of their somewhat bolted-on feel. STK does use exceptions but wxWidgets doesn't since it was conceived before exceptions were introduced into C++. For this reason, exceptions are not allowed to propagate through wxWidgets code, so every potential exception has to be caught and dealt with in the event handlers of the application.

The C libraries like SQLite, AKRip and MAD obviously can't use exceptions as there is no support for them in C. I have therefore tried to confine the use of these libraries to certain "wrapper" classes which deal with the library error return codes internally and throw relevant exceptions to the client class on error. To this end, the MusicCollection contains a nested DatabaseException class, InStream defines an InStreamException and so on. This wrapping of C libraries also has the effect of hiding their non-OO interfaces and providing a more convenient object-based method of accessing their features.

One thing to note about the exception policy is that the above-mentioned exceptions should only be used for true exceptional conditions that could not have been prevented at compile time, such as failure to read input or user error. For programmer errors, such as erroneous parameters being passed to a function, the standard C++ exception logic_error should be thrown.

Pointers and dynamic memory allocation can be a source of trouble in C and C++, especially with regards to creating memory leaks. Another policy has therefore been to try and avoid using pointers as much as possible. In many cases, objects can be fully embedded in other classes, which means that their memory will be freed automatically when their containing object is destroyed. In other situations, such as parameter passing, references can be used instead of pointers. I have also favoured classes from the Standard Template Library (STL) such as vector and string over the native arrays and C-strings (character arrays) due to their automatic allocation and deallocation of memory. Where pointers need to be passed around, the STL smart pointer class auto_ptr is used.

As a general rule, I have tried to limit the use of out parameters to a bare minimum. The reason for this being that their semantics seems very unintuitive. If you pass something into a function, it ought to be some information needed by the function to do its job, not just an empty shell for the function to fill. C (and C++ to a certain extent) uses them a lot, because returning large amounts of data by value is inefficient and returning pointers gives rise to complicated issues of memory ownership. In C++, auto_ptrs can be used instead where large objects need to be returned from functions, so they are generally favoured.

Having said the above about pointers, it might come as a surprise to anyone flicking through the code at the back of this report to find such liberal use of the new keyword within all the GUI classes, particularly in their BuildMe methods. This abundance of dynamic memory allocation is due to the fact that wxWidgets stipulates the use of pointers and heap allocation for *all* GUI controls. This might sound like a memory management nightmare at first but it is in fact not, because as long as all these objects are created with a parent (such as the dialog on which a button sits), the parent automatically frees the memory allocated to all its children at destruction.

There is one exception to this requirement. Modal dialog windows, i.e. dialogs that interrupt the normal program flow until dismissed, do not have to be created on the heap with new, but can be temporarily created on the stack and automatically freed at the end of the function scope. Therefore, all modal dialogs in the application are created on the stack and all non-modal ones on the heap.

5.8 Class Interaction

To enable the reader to get a feel for how the classes interact before delving into the details of the low-level design, Figure 5.3 depicts the sequence of interactions needed to carry out the main task of the application: beat-tracking a song (in this example an MP3). It provides a nice overview as it touches on most of the major classes in the system.

The sequence starts by the user pressing the “Add File(s)” button on the toolbar. As the toolbar is owned by BeatFrame, the event is handled by BeatFrame’s OnAddFiles handler. The handler launches the AddFilesDialog from which the user can select files to add and set various options. Upon clicking OK, control returns to the handler which calls GetSelectedTracks on the dialog to get hold of the selected files. Within this method, the AddFilesDialog uses the FilenameParser (and an external ID3 parsing library if the file is an MP3) for retrieving any metadata from the files. The selected files are then returned in the form of MusicTrack objects. Next, the BeatTracker to use is instantiated and configured with the correct stream (in this case the FileInStream) and options. A pointer to the tracker along with the vector of files to analyse is passed to the ProgressDialog which takes care of the scheduling and progress display. It calls TrackTempo on the BeatTracker to analyse the file, and when finished, adds it to the collection with the call to MusicCollection::Add. Once MusicCollection has updated the database, it calls Notify to tell its Observers about the addition of a new track. The CollectionListCtrl is one of the Observers of the collection and upon receiving the notification, it asks MusicCollection for the new information via its GetTrack method. The display is updated and the program returns to the event loop ready for new user actions.

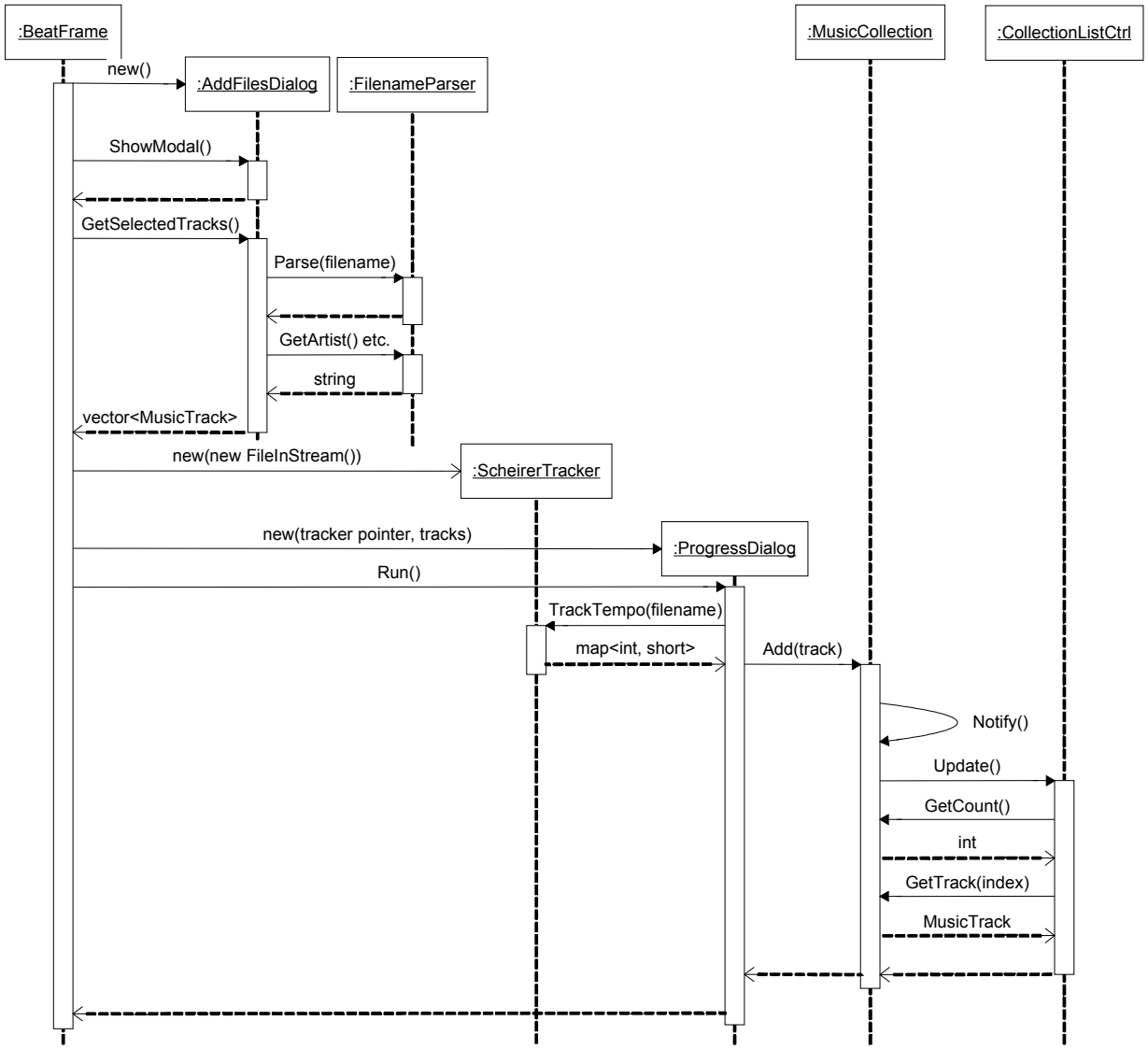

Figure 5.3. Sequence diagram of beat-tracking procedure.

6. Class Design

This chapter provides more detailed descriptions of noteworthy features of the various components in the system.

6.1 GUI

The GUI is the biggest of the packages and it contains all the windows and dialogs along with their event handling code. Figure 6.1 shows the rather crowded static class model. Details about the individual components are given in the subsections further down.

The internals of the GUI classes are largely dictated by wxWidgets. All classes that handle events contain an event table, which is built from special macros defined by wxWidgets. The event table links up events from individual controls with handler functions to be invoked when the event occurs. All GUI classes also have a method called BuildMe, called from the constructor and responsible for creating and laying out all the visual elements of the window. Most of the code in these methods is autogenerated by a tool called wxDesigner, which I used to do the initial visual design of the components. The program is launched by the BeatApp class, which contains the wxWidgets equivalent of the main method, the OnInit handler. From here, the main application frame BeatFrame is created and initialised.

To come up with a suitable visual layout for the main application window (shown in Figure 6.2), it was important to think about what the program will be used for most of the time. This is likely to be browsing, filtering and searching through the collection of tracks. Therefore, the default view when launching the program is the collection view. The most common operations, such as beat tracking and adding files are available from the toolbar.

Since we wish to track tempo changes, some songs might end up with much more tempo information than others. Despite this, tempo information needs to be displayed succinctly within the list in the main view. The chosen solution here is to only display the first and last tempo of a song in the list, and display the full information in a separate window (the MusicTrackDialog, see Section 6.1.4), launched when double-clicking on a list item.

Usability concerns have played a central role throughout the development of the GUI. Nielsen (1994) outlined ten usability heuristics in the early 90s emphasising the importance of “Visibility of system status”, “Match between system and the real world”, “User control and freedom”, “Consistency and standards”, “Error prevention”, “Recognition rather than recall”, “Flexibility and efficiency of use”, “Aesthetic and minimalist design”, “Help users recover from errors” and “Help and documentation”. How these have been realised will be touched upon in the relevant subsections below.

Some of the above are however concerned with application-wide behaviour. For example, to achieve a match between the system and the real world, one aim has been to only use language belonging to the problem domain rather than the system domain. Words like “dialog” or “function” should for example never appear in messages displayed to the user. The error reporting mechanism has been

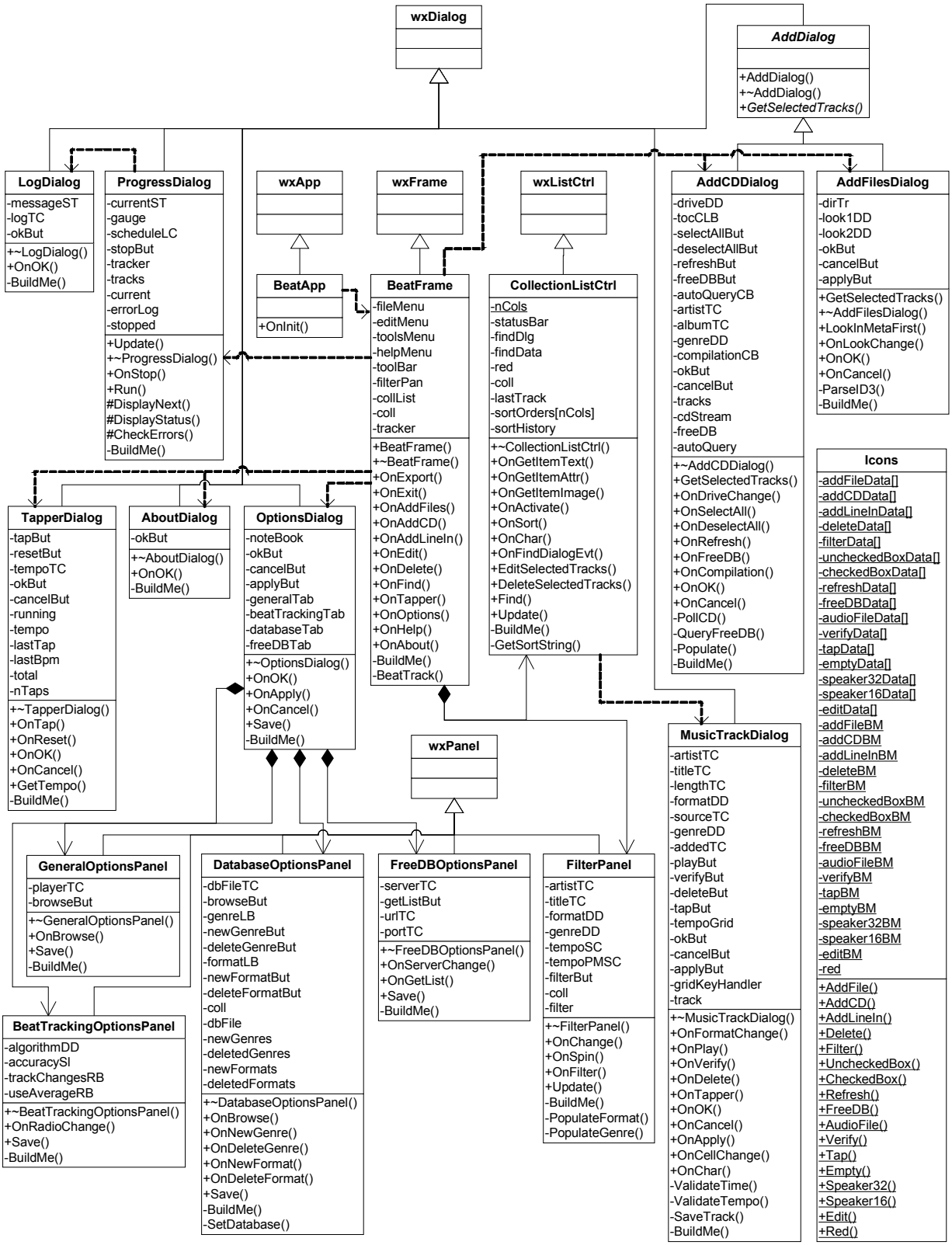

Figure 6.1. GUI classes

designed to only provide technical information such as error codes in debug builds of the program, thus eliminating all jargon in the release builds that end users will see.

User control and freedom can be supported by always making sure that there is a Cancel option on each dialog. Ideally, undo should also be supported for all operations. This is currently not the case

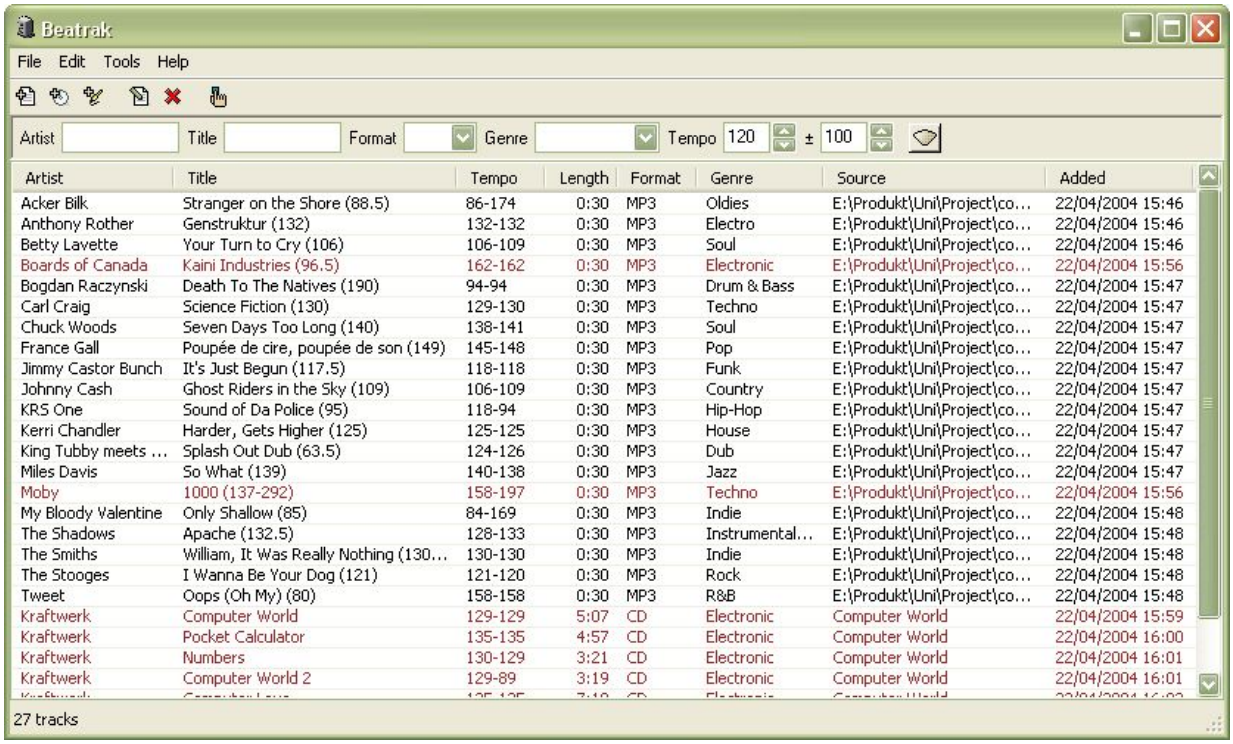

Figure 6.2. Screenshot of BeatFrame, the main application window.

for this application since it requires considerable effort to implement but would be desirable for future versions.

Flexibility and efficiency means providing shortcuts and faster ways of doing things for the experienced user. All the major commands in the program can be invoked by Ctrl-key shortcuts, Tab can be used instead of the mouse for navigating in dialogs etc.

6.1.1 BeatFrame

The BeatFrame is the class responsible for the main application frame shown in Figure 6.2 and is also the main controller in the GUI. It knows about all the other dialogs, and displays them when needed. All events for menu selections and toolbar button clicks are handled by this class and delegated to relevant model or GUI classes for carrying out the requested actions.

More localised behaviour, such as a request to sort the collection by clicking on a column header, is handled directly by the contained CollectionListCtrl, which sends a message to the database to sort its tracks. The general philosophy is that each widget handles and controls the events that are relevant to it. If a task that's specific to a certain component is invoked via a menu in BeatFrame, the task is delegated down to the widget that knows how to handle it.

The BeatFrame class is currently somewhat of a spider in the web as it is associated with most of the other classes in the system. I am not entirely happy with this design as it goes slightly against the OO goal of low coupling but it seemed impossible to tie it all together in any other way.

6.1.2 FilterPanel

The FilterPanel sits along the top of the collection window and can be used to filter the display on various fields. It handles the filter button presses and calls the MusicCollection with the filter settings from its own event handlers. It has also been made an Observer of the MusicCollection so

that the Format and Genre drop-downs are instantly updated if a new format or genre is added to the collection.

With the filter button on, it updates the collection view on each key press, so that on-the-fly searches can be carried out by just typing a few letters of a keyword. While this is very convenient, it can get a little sluggish on slower computers due to the poor performance of the database interface (see Section 6.4.2). This can be overcome by leaving the filter off while typing, and pressing the button once the whole word has been typed.

6.1.3 CollectionListCtrl

The CollectionListCtrl is the control responsible for the list of tracks in the centre of the BeatFrame. It has been implemented in its "virtual" mode, meaning that it doesn't hold any data of its own. This is to enable having a large number of tracks in the collection without consuming too much memory. When scrolled or otherwise changed, the control calls its OnGetItemText method which is overridden to retrieve the corresponding information from the MusicCollection. As a result, the list control only acts as a dumb shell, all the data is handled and kept in the MusicCollection class, described in Section 6.4.2.

Sorting, both ascending and descending, is possible on all columns. When sorting the Tempo column, the track's first tempo is used for comparisons. Three levels of sorting is supported. If we for example, first click on the artist column, and then on the tempo column, tracks will be sorted on tempo, and then on artist where the tempi are the same. Newly added tracks that haven't had their tempo verified yet show up in red in the list as a reminder of this fact.

6.1.4 MusicTrackDialog

The MusicTrackDialog (Figure 6.3) is brought up when a track is double-clicked in the collection view. It displays more detailed tempo information and allows editing of the data. Ideally, it should be possible to edit the data directly in the collection list view, but this is a limitation of the wxListCtrl component that is currently used to display the collection. In the future, this is likely to be replaced by some other list component that allows editing of all the columns in-place.

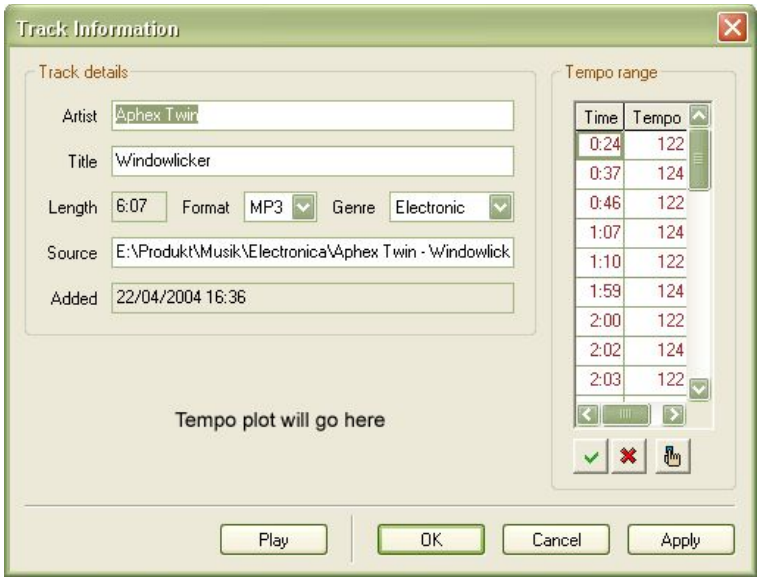

Figure 6.3. Screenshot of MusicTrackDialog.

The buttons along the bottom of the Tempo range subsection can be used to verify the tempi, delete individual entries or launch the Tapper window (see Section 6.1.7). Unfortunately, it was not possible using the currently available features in wxWidgets to get these buttons to be consistent with the native look that the buttons in the main toolbar take on, so they look slightly clunky and unprofessional at the moment.

Figure 6.3 also shows the intended position of a plot of the tempo range at the bottom of the window. This has however not yet been implemented due to time pressures but will be made a priority as it would provide a nice overview of the changing tempo of a piece of music.

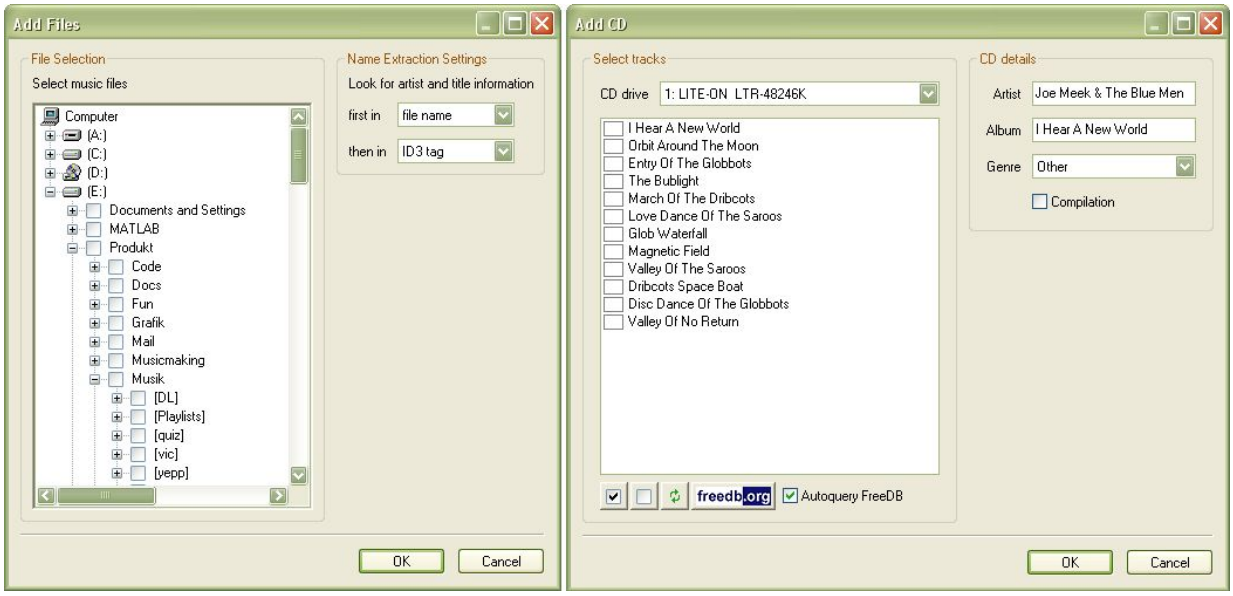

Figure 6.4. Screenshots of AddFilesDialog and AddCDDialog

6.1.5 AddFilesDialog and AddCDDialog

Through these dialogs, shown in Figure 6.4, the user selects files or tracks from a CD to beat track and add to the collection.

For selecting multiple files and folders, I thought the most convenient widget would be a checkable tree control. Unfortunately, wxWidgets doesn't provide one so the standard tree control had to be customised to provide this behaviour. It has been achieved by using the icon next to an item to display an empty box or a checked box depending on whether a mouse click has been detected over it or not. Unfortunately, this meant that the distinction between folder and file normally provided by the folder/file icons was lost, the only way to distinguish between folders and files in the current control is the little + next to all folders.

The AddCDDialog is similar to the file adding one but instead of a tree it uses a flat list to display the tracks on the CD. Buttons for selecting/deselecting all tracks, refreshing and querying FreeDB are available. Again, its visual appearance leaves a lot to be desired as it's using a stock checkable list component from the wxWidgets library. Why anyone would choose to design a rectangular check box is beyond me, but this is another cosmetic feature I intend to modify when time permits.

This dialog needs to use the CDInStream in order to read information such as available CD drives, the number of tracks and their lengths from the CD.

The responsibility for retrieving metadata about the music lies with these two classes. The AddFilesDialog uses the FilenameParser and the ID3Lib library to achieve this, and AddCDDialog uses the FreeDB class to retrieve album information from the online CD database. If a genre not present in the system is encountered in the metadata, it is automatically added to the database at this point.

Figure 6.5. Screenshot of ProgressDialog.

6.1.6 ProgressDialog

As beat tracking currently takes a substantial amount of time, some way of keeping the user informed of what's happening is needed. One of the tenets of usability is visibility of system status at all times. If the program is busy, the user should be informed immediately and not be left wondering whether a button click or a mouse drag had any effect. To this end, the ProgressDialog (Figure 6.5) contains a progress bar indicating how much of the current track has been analysed and also a list of all scheduled tracks and their status.

There are other areas of the system where a user request can sometimes take a second or two to complete. In these cases, I have tried to ensure that an hourglass cursor is always displayed as soon as the user has requested an action.

The ProgressDialog contains a Stop button for cancelling the beat tracking. One way of implementing this would be to run the BeatTracker in a separate thread controllable from the ProgressDialog. I chose not to use any multithreading in this application, however, as threads are often difficult to program properly and can give rise to subtle bugs. Instead, as the ProgressDialog is an Observer of the tracker, control is given to it regularly during a beat tracking operation so that it can update its display. The Stop button is polled during these updates. As long as the updates happen regularly enough, the Stop button can be as responsive as it would be using multiple threads.

6.1.7 TapperDialog

Even if the beat-tracking algorithm used here can detect tempi correctly in the majority of cases, it's nowhere near as good as a human listener. Yet. As a concession, the program includes a manual tempo tapping feature depicted in Figure 6.6. It simply measures the elapsed time between successive presses of the Tap button and averages this over time. As only a few lines of code is needed to do this, it's all been kept in the dialog class itself. It seemed unnecessary to separate this out into a model class.

Figure 6.6. Screenshot of TapperDialog.

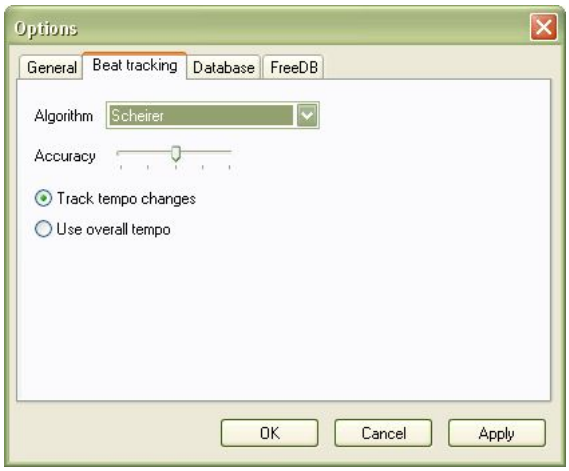

Figure 6.7. Screenshot of OptionsDialog

6.1.8 OptionsDialog

A number of things should be configurable by the user, such as external player program, how tempi should be reported, what database file to use etc. It should also be possible to add new genres and formats to the system. This can be done from the OptionsDialog displayed in Figure 6.7.

Options are saved to a configuration file in the application's current directory. wxWidgets provides a convenient class called wxConfig for this purpose. It provides a static method for getting a config object which then takes care of reading and writing values to the file. Apart from the options specifically set from the OptionsDialog, the config

file also automatically stores things such as size and position of windows, columns and other GUI elements modified by the user.

6.2 Beat Tracking

The beat-tracking classes are shown in Figure 6.8. The main player in the beat-tracking package is the sole implemented tracker, the ScheirerTracker. It makes use of a number of supporting classes that slot in at different stages of the process. These classes are the STK Filter class, a simple digital filter, the HanningFollower, which performs convolution in order to find amplitude envelopes and the Resonator, a resonating comb filter for detecting the frequency of periodic signals. The RangeProcessor is used for post-processing of a tempo range once it's been produced by the tracker.

Scheirer's 1998 paper contains an in-depth description of the algorithm. What follows is an overview of the method focusing on this particular implementation.

6.2.1 Bandpass Filtering

The first step of the algorithm involves filtering the audio signal to separate it into frequency subbands. This is taken care of by the STK Filter class. Digital filter theory is a complex subject of which I've only been able to scratch the surface but the basic operation of a digital filter is quite simple. In the case of a recursive filter like the ones used here, it multiplies a set number of its most recent input and output values by a predetermined set of coefficients and adds these products up to get the new output value. A non-recursive filter uses only previous inputs.

By choosing the coefficients carefully, all kinds of magic can be achieved. I cheated and enlisted Matlab to help me find the right coefficients for a set of six recursive filters splitting the signal into subbands of 0-200, 200-400, 400-800, 800-1600, 1600-3200 Hz and above 3200 Hz. These coefficients are stored as static arrays in the ScheirerTracker class and are passed to the STK Filter class constructor on initialisation of the tracker. A signal can then be filtered by just passing samples into the Filter's tick method.

The motivation for the bandpass filtering is that a musical signal often contains different rhythmic elements at different frequencies. The low frequencies might contain bass drums or bass lines. In the mid-range, vocals and lead instruments are found and in the treble, hihats and other short percussive sounds often reside. By bandpass filtering, we can separate out these different components and take their influence on the overall rhythm into account in a much more accurate way than if considering the entire spectrum at once. It is harder to find a regular rhythmic pulse in the spectrum as a whole as compared to the individual subbands.

6.2.2 Envelope Extraction

The next step of the process involves finding the amplitude envelopes of the subbands as sudden changes in amplitude often signal rhythmically significant events. Scheirer suggests convolution with a half-Hanning window as a good way of finding the envelope. A Hanning window is simply a smooth function shaped like the top of a cosine wave. By halving this so that it starts at its maximum and smoothly slopes away towards 0, we get a half-Hanning window (pictured in Figure 6.9).

The job of convolving the signal with this function is carried out by the HanningFollower class. Since we only need the envelopes sampled at a much lower rate than the incoming audio, the HanningFollower has been designed not to carry out the convolution each time a new input is received, but merely store the value in an internal buffer until GetOutput is called when the

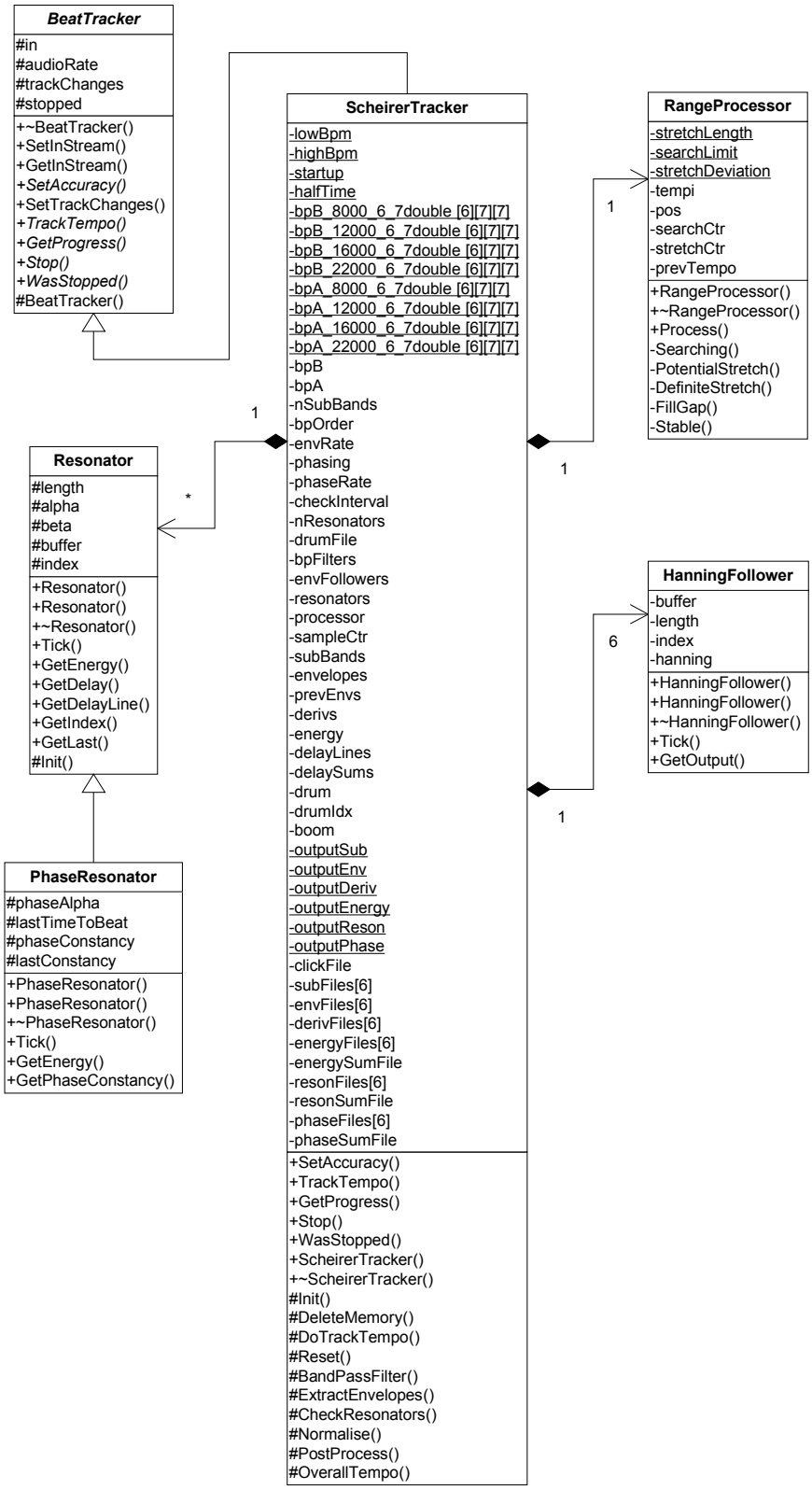

Figure 6.8. Beat-tracking classes.

convolution is performed. This means that we only have to perform the convolution around 200 times per second instead of 44,100 times if we were using audio sampled at that rate.

Still, convolution is a very time-consuming process even with this optimisation as it, unlike the bandpass filters described above, is a non-recursive process. It should be possible to convert the

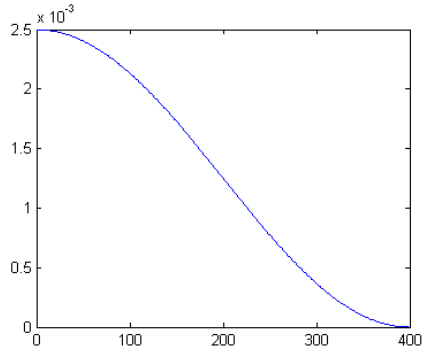

Figure 6.9. A 400-sample half-Hanning window.

HanningFollower into a recursive filter somehow, and gain a considerable boost in execution speed, but I've yet to figure out how it's done. Some experiments were done with a simple recursive envelope follower but it didn't produce results as good as the HanningFollower.

Once we have the envelopes, downsampled to about 200 Hz, we take the derivative by subtracting the previous envelope value from the current. Since we are looking for rhythmic onsets, we are not interested in decaying envelope values so if the derivative is negative, we set it to 0 (known as half-wave rectification). This derivative signal is what is taken to represent the rhythm of the music and is sent to the tempo-detecting resonators.

6.2.3 Tempo Determination

The third helper class is the Resonator. It is a simple recursive comb filter that has the property of resonating when stimulated with input of a certain periodicity. Each filter has a delay parameter which determines the period it resonates at. By setting the delays of these filters to values corresponding to the range of tempi we want to detect, the delay of the resonator corresponding most closely to the tempo of the incoming signal will represent the tempo.

The resonators also have a gain parameter called alpha. This regulates how quickly the resonator adapts to changes in the tempo. A lower value means recent inputs are given more importance and a higher value emphasises the history of previous inputs more. Currently this is set to ensure that the halftime, i.e. the time it takes for the resonator to reach half its energy in response to an impulse response, across all resonators is 2 seconds.

Each of the subbands has a dedicated bank of resonators tuned to a closely spaced range of tempi. The six banks of resonators are identical, each responsible for tracking the rhythm in one subband of the signal. The actual tempo estimate is determined by summing the energy content of these resonators across subbands to get the overall winner. The number of resonators per bank depends on how wide a tempo spectrum we want to track and the envelope sampling rate. For example, if we want tempi between 80 and 200 BPM and use an envelope sampling rate of 200 Hz, the resonators will have delays between 150 (for 80 BPM) and 60 (for 200 BPM) envelope samples.

Since the resonators use integer delays, the tempo resolution of the resonators can become quite coarse in the faster tempi. If 60 samples corresponds to 200 BPM, the next delay up is 61 samples, which corresponds to 196.7 BPM. At the lower end, resolution is better due to the smaller relative differences between longer delays. The next delay down from 150 (80 BPM) is 149 samples which corresponds to 80.5 BPM. To get better accuracy, the envelope sampling rate would need to be increased but this also leads to longer execution time.

This has proved a tricky balance to find as one of the more common reasons for failed tempo detection is that the real tempo of a piece of music lies halfway between two resonators. If the tempo is in the upper region of the range where the resonators are further apart, this could lead to the wrong resonator being picked as the winner. To ameliorate this somewhat, my implementation also takes into account the energy of the two neighbouring resonators when working out the winning resonator. This is based on the assumption that if the tempo lies between two resonators, both of these two resonators will have fairly high outputs, and if taken together, will beat the erroneous resonator.

Currently, the low and high BPM limits are constants set at compile-time and the number of resonators needed to track this range with as much resolution as the envelope sampling rate allows is calculated at runtime.

There is also the possibility of using fractional delay resonators, which would allow for a completely arbitrary spacing of tempi, but this would mean making the resonators slower and more complex, but it is a feature worthy of future investigation.

6.2.4 Phase Constancy

When testing the beat tracker, I noticed that it was often having difficulties with music where the rhythm had a particlar kind of syncopation, illustrated by Figure 6.10. The instrument or sound marking the rhythm does not fall on the downbeat, but instead plays five notes in the space of four downbeats. This can be heard clearly in the track "Science Fiction" by Carl Craig, included in the test set (see Section 7.2.1) and variants of it are quite common in modern music.

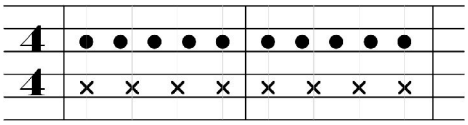

Figure 6.10. Rhythm played off the downbeat. The circles represent the actual onsets and the crosses represent the downbeats.The downbeats are not actually played.

The beat tracker tended to go for the tempo represented by the interval between these faster onsets rather than the actual downbeats. This got me thinking about what properties of this rhythm made it easy for humans to pick the right tempo, and it struck me that it could be expressed as phase constancy. For the correct tempo, represented by the downbeats, the phase stays constant, i.e. the spacing of beats is regular throughout a song. For the offset rhythm, the phase has to keep "realigning" itself every one or two bars for the overall effect to be that of the slower tempo.

To try this idea out, I wrote a variant of the Resonator class, called PhaseResonator, which keeps track of the phase constancy of the signal it is tracking. The idea is that if it's tracking the correct tempo, the phase is expected to stay fairly constant. The phase is simply the distance to the peak in the resonator's delay line. A resonator where the peak stays in the same place each subsequent period is said to be exhibiting high phase constancy. This constancy value is then used to weight the total energy output of the resonator returned by the GetEnergy method.

It seems to have worked quite well, performance improved dramatically for the tracks that had this particular rhythm. As evidenced by Figure 6.11, "Science Fiction", which was estimated at around 80 BPM (half the tempo of the rhythm played by the syncopated onsets) by the normal resonator, was now estimated at the correct tempo of 130 BPM.

6.2.5 Post-Processing

Apart from the techniques described above, the beat tracker includes no higher-level knowledge about how musical rhythm is likely to develop in a piece. For example, if a song is steadily reported as being 120 BPM for a longer period of time, and all of a sudden, the 156 BPM resonator happens to come out on top during one check only to be followed by the 120 BPM resonator again the next second, the tracker happily reports the tempo as being 156 BPM at that particular moment.

Since we know that it's highly unlikely for a song's tempo to jump to a completely different value for just a second or two, it would be desirable if this kind of knowledge could be incorporated to eliminate blips of this kind. This is where the RangeProcessor comes in. It is given the complete tempo range of the beat-tracked piece after ScheirerTracker is finished with it and tries to improve it by applying some higher-level heuristics.

It is essentially a state machine similar to a Turing machine in that it reads from its input (a sequence of tempi) and changes its internal state according to what it sees. It also changes the tempo values of the input depending on the state it's in. The state diagram in Figure 6.12 describes its operation. Each of the states is implemented as a method in the class.

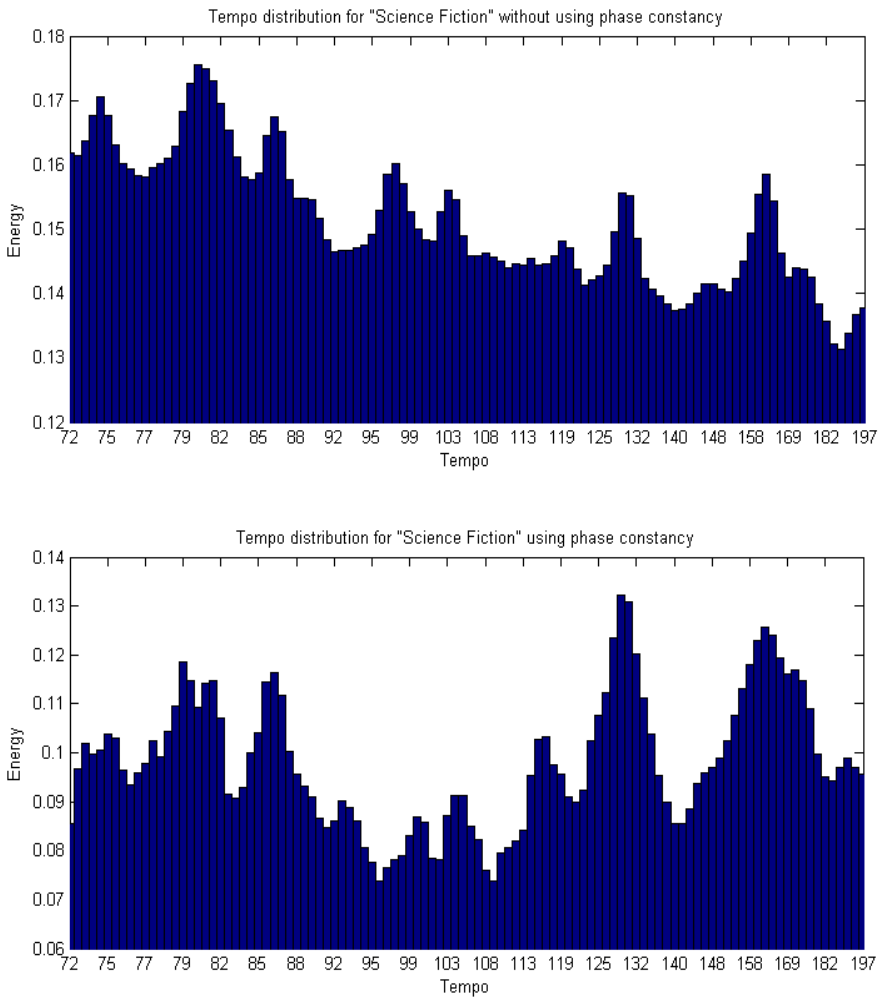

Figure 6.11. Summed resonator energy outputs for “Science Fiction” without using phase constancy (top) and with phase constancy weighting (bottom).

The basic idea is that tempo tends to change gradually and that if it changes rapidly, it should be sustained for some time to be regarded as a true tempo change. If not, it is reset to the tempo of the last previously stable tempo stretch. Two subsequent tempi are regarded as stable if the second is within a close range of the first or if it is double or half of the first allowing for the same relative deviation. The deviation allowed is currently set at 3.3 % of the current tempo per second. If the tempo changes abruptly, a counter is used to count the number of seconds the tempo stays within this new stable stretch. This is represented by the state Potential Stretch. If the tempo stays within this new stretch for longer than a predetermined number of seconds (currently 5 is used), the state is changed to Definite Stretch which means that the new tempo is considered a true tempo change and is retained in the output. If the tempo veers outside of this stretch, the state changes back to Searching again.

6.2.6 Structure of ScheirerTracker

The ScheirerTracker is laid out in a vaguely Template-like fashion with a main beat-tracking method called DoTrackTempo that in turn calls other methods such as BandPassFilter and ExtractEnvelopes to carry out the separate processing steps. Lots of facilities for debugging output have been included to be able to output, analyse and visualise the internals of the process, for example using Matlab. The different types of debugging output can be switched on or off with constant boolean flags defined in the header file. This means that the conditional checks at

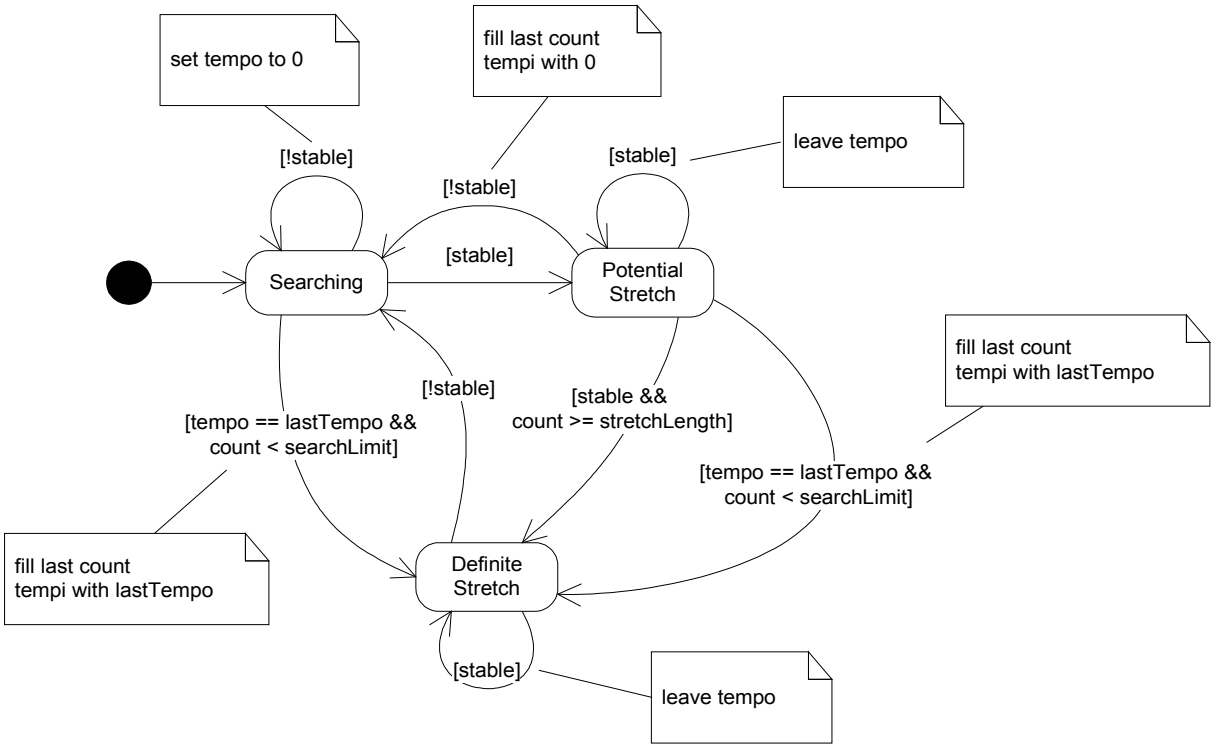

Figure 6.12. State diagram of the RangeProcessor.

runtime should be able to be optimised away by the compiler when the flags are set to false, thus incurring no extra runtime overhead even if left in the code.

6.2.7 Efficiency

The policy of using STL containers extends to the beat-tracking classes. For instance, the beat tracker uses vectors instead of arrays internally for storing sample values and other numbers associated with the process. Some people might argue that STL classes like vector should not be used in performance-critical code, but the fact is that vectors are allocated as contiguous chunks of memory and should be no slower than arrays if used wisely, i.e. initialised to the right size by a call to reserve beforehand so that no on-the-fly reallocation will occur.

The BeatTracker also provides an Accuracy setting, which represents a tradeoff between speed and accuracy of tempo detection. The parameters most influential on speed are the sample rate of the original audio and the envelope sampling rate. As for a simple relationship between parameter settings and accuracy, it's not as straight-forward. The audio rate does not seem to be hugely influential on accuracy, sample rates as low as 8,000 Hz can be used without much reduction in accuracy. For some music where the treble is important in determining the tempo, slightly better performance has been observed with a higher audio rate. The envelope rate is linked to accuracy in the sense that the higher the rate, the more closely spaced the resonators are. This should help improve accuracy in cases where the real tempo of a signal falls between resonator frequencies.

On this basis, the accuracy setting gradually increases the audio rate and the envelope rate the higher the desired accuracy. Possibly useful options that haven't been investigated involve using less than six frequency subbands at lower accuracies and lower-order filters to improve efficiency.

6.3 Audio Streams

The application currently support audio input in WAV, MP3 and CD format. The WAV support is handled by the STK class WvIn, MP3 decoding is taken care of by the MAD library and for CD reading, the AKRip library is used.

Despite using libraries for these tasks, a substantial effort still had to be made to make use of them and fit them into the context of the application. The implementation files in the package total some 1,200 lines of code. Both AKRip and MAD are C libraries so wrapper code had to be written to interface with the other C++ classes in the system. Furthermore, MAD in particular is very low-level and only concerned with the actual decoding of MP3 data, all the disk reading, buffering etc has to be done by the client. The stream classes responsible for this are shown in Figure 6.13.

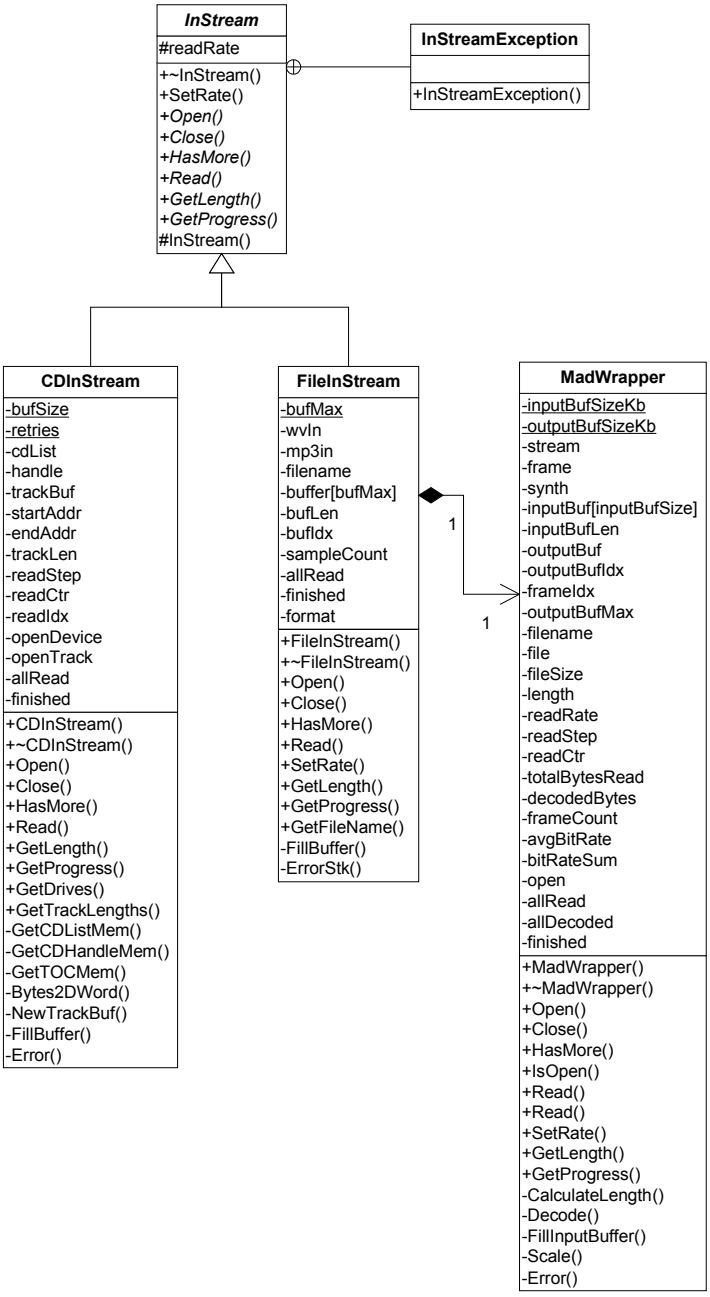

Figure 6.13. Audio stream classes.

All the stream classes have a read rate parameter, which is the sample rate at which the audio should be returned. The actual audio source, which is likely to have a rate of 44,100 Hz in most cases, is upsampled or downsampled by the streams as needed. Audio fidelity is not a concern since the output is only going to be used as input to the beat tracker, not for human consumption. Therefore, the up- and downsampling method used is the fastest, crudest possible. In the case of upsampling, it simply repeats the current value for the required amount of time and for downsampling, it picks out values from the higher-rate audio at regular intervals as dictated by the read rate.

6.3.1 FileInStream

The thinking behind FileInStream is that to the rest of the system, an audio file is an audio file, regardless of whether it happens to be in WAV format or MP3 or something else. The BeatFrame, AddFilesDialog or indeed the user, should not need to be aware of this difference as it's irrelevant to their job. The FileInStream encapsulates this distinction and does the right thing depending on the type of file it is passed.

In actual fact, the difference between an uncompressed WAV file and a compressed MP3 file is huge, so underneath the hood, very different activities take place. For a WAV file, the FileInStream delegates all work to WvIn, which provides disk reading, buffering and the rest. For MP3s, it delegates the work to MadWrapper, a class written to provide a more convenient interface to the MAD C library.

It could be argued that MadWrapper should be a third subclass of InStream called something like MP3InStream as it's very similar to the stream classes in design. It is a good argument that I've been considering, but it would mean that the higher-layer classes that do the scheduling of beat-tracking operations would need to be aware of the difference between different file input formats, losing the encapsulation described above.

The FileInStream keeps read or decoded data in an internal buffer so that the checks to see whether it should delegate to WvIn or MadWrapper only take place when the buffer needs filling.

6.3.2 CDInStream

Reading audio from CD is a distinctly different task from reading it from file as support for this is not often provided by operating system calls. Therefore, third-party layers like Adaptec's ASPI are often used for this purpose. ASPI is the underlying technology of the AKRip library that is doing the actual reading. A problematic issue is that ASPI doesn't come pre-installed on Windows, so the application needs to be distributed with the correct drivers or the user needs to get hold of them separately.

The CDInStream is essentially a wrapper around AKRip, to provide an InStream interface to the CD drives. AKRip is not a very pleasant library to use, it's full of incongruities and badly documented features. For this reason, it took a considerable amount of time to get CDInStream working as intended.

Another point worth noting is that AKRip is the only component in the application that is Windows-only, so in order to compile on Linux with CD reading support, a different library would need to be found.

6.4 Database

The database package, shown in Figure 6.14, consists of the perhaps too monolithic MusicCollection class along with the simple container classes MusicTrack and TempoRange. The MusicCollection takes care of all the interfacing to the SQLite database. The SQLite interface is again a C one, which requires the use of static callbacks to return the data fetched from the database.

6.4.1 Data Model

Figure 6.15 shows the rather simple data model. The music_track table holds all information about individual tracks in the database. Since the ability to account for tempo changes, i.e. several tempi per track, is needed, a tempo_map table is used. It links track IDs to time instants and the tempo the track changed to at that time. For a track with a constant tempo throughout, there will only be one entry for the track in the tempo_map.The genre and format tables are simple lists of all genres and formats present in the database.

A drawback of this design is that some fairly complex SQL statements had to be constructed for selecting tracks along with desired tempo information from the database. For example, since we want to be able to sort tracks on tempo, and sorting on tempo should mean sorting according to the

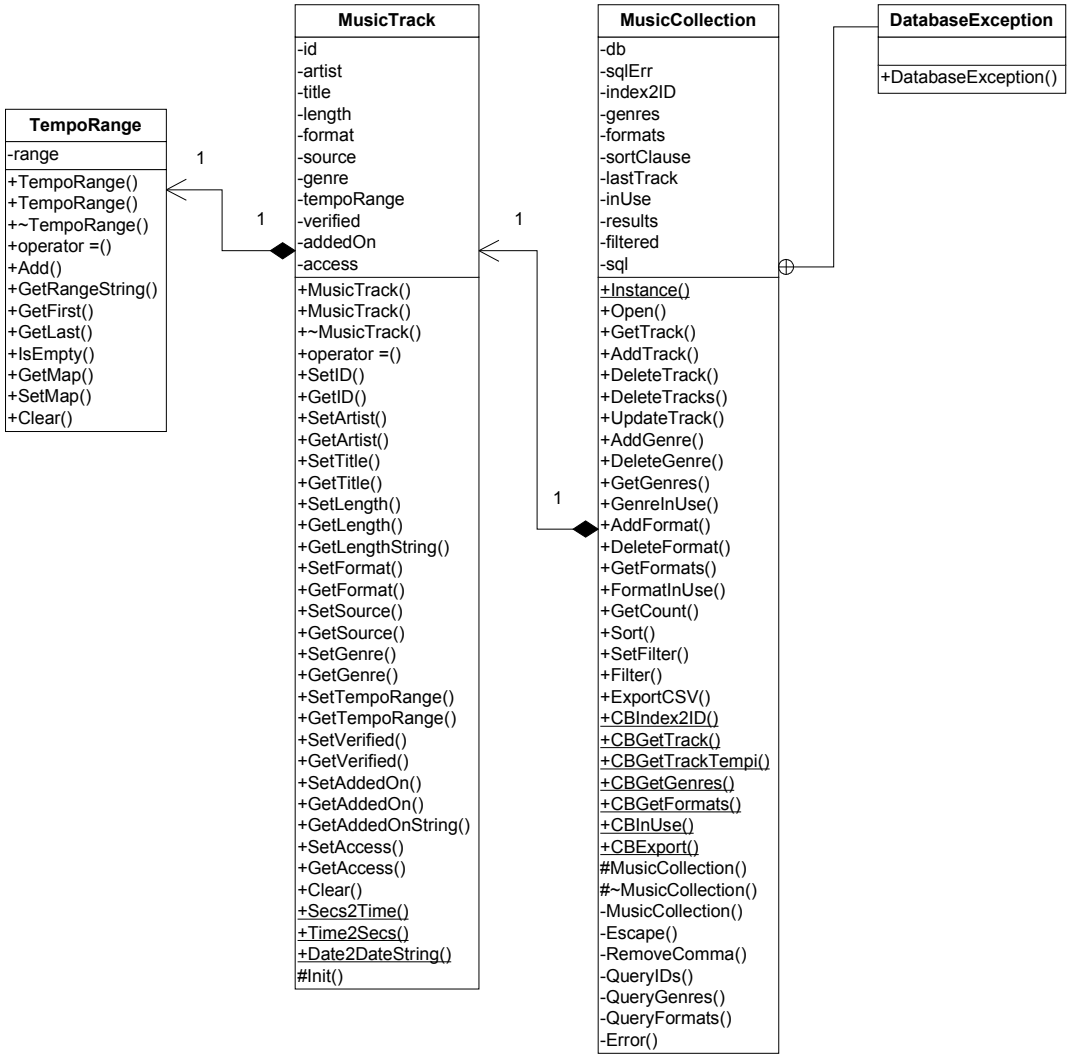

Figure 6.14. Database classes.

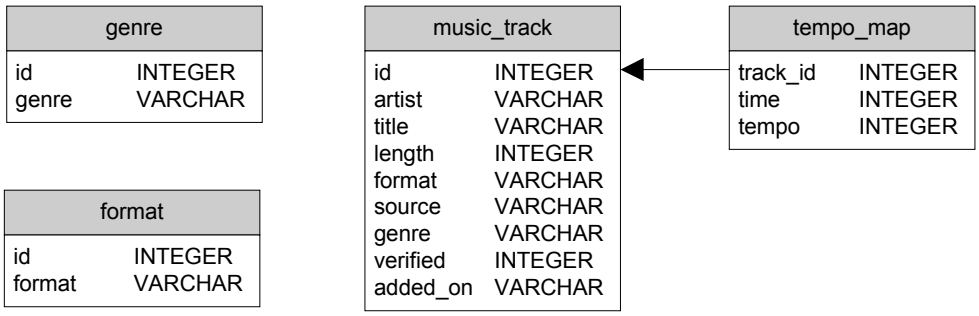

Figure 6.15. Data model.

first tempo in a track's tempo range, the following incantation is needed to retrieve the IDs in the order of ascending tempi:

```
SELECT   mt.id
FROM     music_track mt
         LEFT OUTER JOIN
         tempo_map tm
         ON mt.id = tm.track_id
           INNER JOIN (
             SELECT   track_id, MIN(time) AS min_time
             FROM     tempo_map
             GROUP BY track_id) AS
           tmd
           ON tm.track_id = tmd.track_id AND tm.time = tmd.min_time
ORDER BY tempo;
```

6.4.2 MusicCollection

Communication with the database is done through the various Add and Get methods of the MusicCollection class. It's currently rather simplistic in that it doesn't do any caching. Any request to get a specific track is sent straight to the database via an SQL SELECT statement. Requests to filter or sort the collection are also done through SQL.

This class is the brain behind the collection list in the GUI. All the functionality invoked from the list window resides here. It also holds a mapping between list indices and track IDs. When the CollectionListCtrl needs the details for a particular row index, it passes this index to the MusicCollection's GetTrack method and the track details are returned in the form of a MusicTrack object. When a column header is clicked in the CollectionListCtrl to sort one of the columns, a message is sent to MusicCollection which then resorts its index-to-track-ID mapping. The order of this mapping is what determines the current sort order of the list.

Because of this fairly naïve implementation, the CollectionListCtrl/MusicCollection combination is currently somewhat of a bottleneck in the system. As the number of tracks in the collection grows, operation of the list control becomes less responsive. This is most likely due to the fact that it has to go and fetch some data from the disk for each SELECT statement executed. More of the data clearly needs to be held in memory to achieve smoother interaction. At the moment, this is the number one priority for improvement in the application as a whole.

6.4.3 MusicTrack and TempoRange

The MusicTrack is a simple container class for information about music tracks. Each MusicTrack object contains a TempoRange which is a map between time instants and tempi. The MusicTrack does not know anything about how to save itself to the database as it always exists in the context of a MusicCollection.

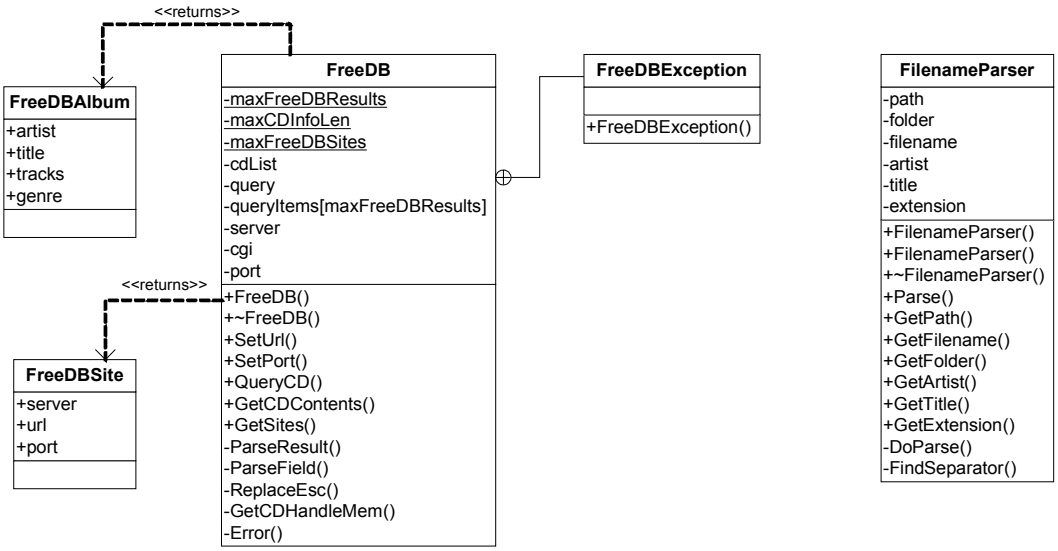

Figure 6.16. Utilities classes.

6.5 Utilities

The Utilities package (Figure 6.16) is the home of some helper classes that didn't fit anywhere else. The FilenameParser is a class for trying to retrieve meta information about music files from their filenames, and FreeDB is an interface to the online FreeDB CD database. The package also contains the file Globals.h, which isn't a class as such, and contains various global type definitions and a debug logging facility.

The motivation behind the FilenameParser is that many people have collections of music files organised into file system folders and identified by filenames containing the artist and title of the track, often separated by a hyphen or other separator symbol. In this case, embedded metadata like ID3 tags have often been ignored or may contain erroneous data. Many MP3-handling applications completely ignore this information and go straight to the ID3 tag to find artist and title information. The current application gives the user a choice as to what order meta information should be searched for. For a collection with mostly correct filenames, but unkempt ID3 tags, looking for meta information in the filenames first will return much more accurate data.

Through the FilenameParser class, information like artist, title, file extension and parent folder can be extracted after passing it a path name. It tries to deduce artist and title information by looking for a separator like "-" or variants somewhere in the filename and extracts the information on both sides of the separator as artist and title.

The FreeDB class is a wrapper around the FreeDB access facilities provided by the AKRip library. It allows querying the database for matches for a particular CD, and then retrieval of all the data for the chosen album. As the CD information returned by AKRip is raw data in the format of the CDDB protocol, a special parser had to be written for parsing this data into a more useful data structure.

The logging facility in Globals.h deserves a mention. It has been defined as a preprocessor macro LOG, which in debug builds outputs its argument to a log file. In a release build, LOG is defined to do nothing. This means that the LOG statements dotted throughout the code can remain in release builds without hurting efficiency one bit as they will be replaced by whitespace by the preprocessor before compilation.

7. Testing

Unfortunately, testing has not been as extensive as I would have liked due to the implementation stage taking longer than planned. All classes have however undergone class testing, although it has been somewhat ad-hoc and not followed a rigid test plan. Some stress testing has also been carried out where I spent some time trying out various strange things to see if some more bugs could be uncovered.

7.1 Conformance to Requirements Specification

The following table details the results of conformance tests to verify that the implementation matches the requirements specification. Only the core requirements are included as almost none of the extended requirements have been implemented.

ID	Requirement	Pass	Fail	Comment
001	Track the tempo of recorded music	✓		Implemented by ScheirerTracker
002	Track tempo changes	✓		Implemented by ScheirerTracker
003	Process WAV files	✓		Implemented by FileInStream
004	Process CD audio	✓		Implemented by CDInStream
005	Process MP3 audio	✓		Implemented by FileInStream and MadWrapper
006	Store information about analysed songs in an internal database	✓		Implemented by MusicCollection
007	Allow manual data entry of artist	✓		Implemented by MusicTrackDialog
008	Allow manual data entry of title	✓		Implemented by MusicTrackDialog
009	Allow manual data entry of genre	✓		Implemented by MusicTrackDialog
010	Automatically download CD song information where possible	✓		Implemented by FreeDB
011	Automatically determine MP3 song information where possible	✓		Implemented by AddFilesDialog and FilenameParser
012	Automatically store song source information	✓		Implemented by Source field in MusicTrack
013	Allow batch processing of files	✓		Implemented by AddFilesDialog and ProgressDialog
014	Display database contents to user	✓		Implemented by CollectionListCtrl
015	Allow sorting of tracks based on tempo and multiples thereof	✓	(×)	Sorting on tempo implemented by MusicCollection but not on multiples
016	Allow filtering of tracks based on tempo and multiples thereof	✓	(×)	Filtering on tempo implemented by MusicCollection but not on multiples
017	Allow more complex filtering based on other attributes	✓		Filtering on artist, title, genre and format implemented by FilterPanel
018	Allow manual verification of tempo	✓		Provided by Verify button in MusicTrackDialog and implemented in MusicTrack

ID	Requirement	Pass	Fail	Comment
019	Provide a “tap” function for manual tempo detection	✓		Implemented by TapperDialog
020	Allow customised display of columns in database view		×	Currently there is no way of not displaying certain columns
021	Allow resizing of columns in database view	✓		Implemented by CollectionListCtrl
022	Remember personalisation settings between sessions	✓		Implemented by config system
023	Provide an export to CSV function	✓		Implemented by MusicCollection
024	Allow cancelling of beat-tracking	✓		Provided by Stop button in ProgressDialog
025	Allow editing of data in collection view	✓		Implemented by MusicTrackDialog
026	Allow deletion of data in collection view	✓		Implemented by MusicTrackDialog
200	Process songs faster than their playing time	✓		On a reasonably modern computer, this is always the case
201	Detect tempo to within 1 BPM's accuracy	✓	(×)	Dependant on the envelope rate used. When using the standard envelope rate of 200 Hz, tempi in the upper regions of the scale are not detected to 1 BPM's accuracy due to restrictions imposed by the resonators (see Section 6.2.3).
202	Detect tempi between 80 and 200 BPM	✓		
203	Hold up to 100,000 songs without significant performance reduction		×	This is not true as the database interface is currently too slow to provide decent performance with this many tracks (see Section 6.4.2).
300	Run on Windows	✓		
301	Run in a minimum resolution of 640x480	✓		
400	Provide contextual help in the form of tooltips	✓		Tooltips are provided where needed
401	Provide keyboard shortcuts for the experienced user	✓		All major features have shortcuts
402	Provide feedback about beat-tracking process	✓		Implemented by ProgressDialog
403	Use consistent terminology throughout	✓		
404	Allow TAB navigation in all dialogs	✓		
405	Have a default button in all dialogs	✓		
406	Have an online help system	✓	(×)	Help text written but not yet integrated into application due to wxWidgets complications.
500	Look modern and sleek	✓		Matters of taste obviously influence whether this requirement is fulfilled or not, and admittedly, some components still leave a lot to be desired visually.
501	Have a native look-and-feel	✓		Due to the cross-platform nature of wxWidgets, some components do not use the native widgets, but at least 90% of the interface looks and feels native.
600	Be designed so that beat-tracking algorithm(s) can be reused	✓		Beat-tracking classes are completely independent of the other packages.

Table 7.1. Conformance to requirements specification.

As is evident from the table, the implementation is not 100% conformant. The major problem area is the speed of the database view as discussed in Section 6.4.2. Customised column views, i.e. the ability to show only certain columns, has not been implemented since there seemed to be no apparent need for it with the low number of columns currently used. The omission of sorting and filtering facilities on multiples of tempi is, I have to admit, entirely down to forgetfulness. I had simply forgotten about this requirement until looking back at the requirements at a late stage of development.

7.2 Performance of Beat Tracking

In order to test the beat tracking performance of the program, I picked a set of 20 songs in various styles to use as a test set. To speed things up, only a 30-second excerpt of each song was used. When testing the beat tracker, I repeatedly ran the test set through it with different parameter settings in order to find the settings that gave the best results. Naturally, no particular setting will work perfectly for all different styles so it's a matter of finding the best compromise.

7.2.1 The Test Set

The test set was chosen to include a wide range of different styles, but with the common feature of having some kind of rhythm that would be discernible to a human. The tables below show the songs comprising the set. The correct tempi were calculated by hand using the Tapper feature.

1. Acker Bilk – Stranger on the Shore			
Genre	Jazz	Tempo	88.5 BPM
Comment	Slow instrumental ballad featuring clarinet and strings and a very soft rhythmic backing.		

2. Boards of Canada – Kaini Industries			
Genre	Ambient Electronica	Tempo	96.5 BPM
Comment	A modulated synth melody line on its own without accompaniment.		

3. Johnny Cash – Ghost Riders in the Sky			
Genre	Country	Tempo	109 BPM
Comment	Typical quite steady country rhythm.		

4. Kerri Chandler – Harder, Gets Higher			
Genre	House	Tempo	125 BPM
Comment	Vocal house track with typical 4/4 house beat.		

5. Carl Craig – Science Fiction			
Genre	Detroit Techno	Tempo	130 BPM
Comment	A 4/4 track where the predominant rhythm is not on the downbeat.		

6. Miles Davis – So What			
Genre	Jazz	Tempo	139 BPM
Comment	A syncopated laid-back jazz number with the rhythm in the hihats.		

7. France Gall – Poupée de cire, poupée de son			
Genre	60s Pop	Tempo	149 BPM
Comment	A 60s pop song with orchestral backing and quite steady rhythm.		

8. Jimmy Castor Bunch – It's Just Begun			
Genre	Funk	Tempo	117.5 BPM

Comment	Early 70s funk with a fantastic syncopated beat.

9. King Tubby meets Lee Perry – Splash Out Dub			
Genre	Dub Reggae	Tempo	63.5 BPM
Comment	Elastic dub track with a typical sparse reggae rhythm.		

10. KRS One – Sound of the Police			
Genre	Hip Hop	Tempo	95
Comment	The excerpt starts off with voices only and continues with typical hip hop beats.		

11. Betty Lavette – Your Turn to Cry			
Genre	Soul	Tempo	106 BPM
Comment	A soul ballad with understated rhythm.		

12. Moby – 1000			
Genre	Techno	Tempo	137-292 BPM
Comment	Novelty track that gradually speeds up to 1,000 BPM, produced to get into the Guinness Book of Records for fastest record ever. Included to see if the tracker can follow the ever-increasing tempo.		

13. My Bloody Valentine – Only Shallow			
Genre	Indie Rock	Tempo	85 BPM
Comment	Noisy with lots of guitar feedback but with a clear rhythm.		

14. Bogdan Raczynski – Death to the Natives			
Genre	Drum & Bass	Tempo	190 BPM
Comment	Very busy drum & bass track with frantic rhythms pushing against each other.		

15. Anthony Rother – Genstruktur			
Genre	Electro	Tempo	130 BPM
Comment	Crisp electro with a typical electro beat but with a syncopated melody riff.		

16. The Shadows – Apache			
Genre	Pop	Tempo	132.5 BPM
Comment	60s instrumental guitar pop without drums and the rhythm in the guitars.		

17. The Smiths – William, it was Really Nothing			
Genre	Indie Pop	Tempo	130.5 BPM
Comment	80s guitar pop with drums		

18. The Stooges – I Wanna be Your Dog			
Genre	Rock	Tempo	121 BPM
Comment	Early 70s rock with steady rhythm and guitar feedback.		

19. Tweet – Oops, Oh My			
Genre	R&B	Tempo	80 BPM
Comment	Modern R&B with an unusual, almost tribal-sounding rhythm.		

20. Chuck Woods – 7 Days Too Long			
Genre	Northern Soul	Tempo	140 BPM
Comment	Upbeat 60s soul stomper with a Motown-type beat.		

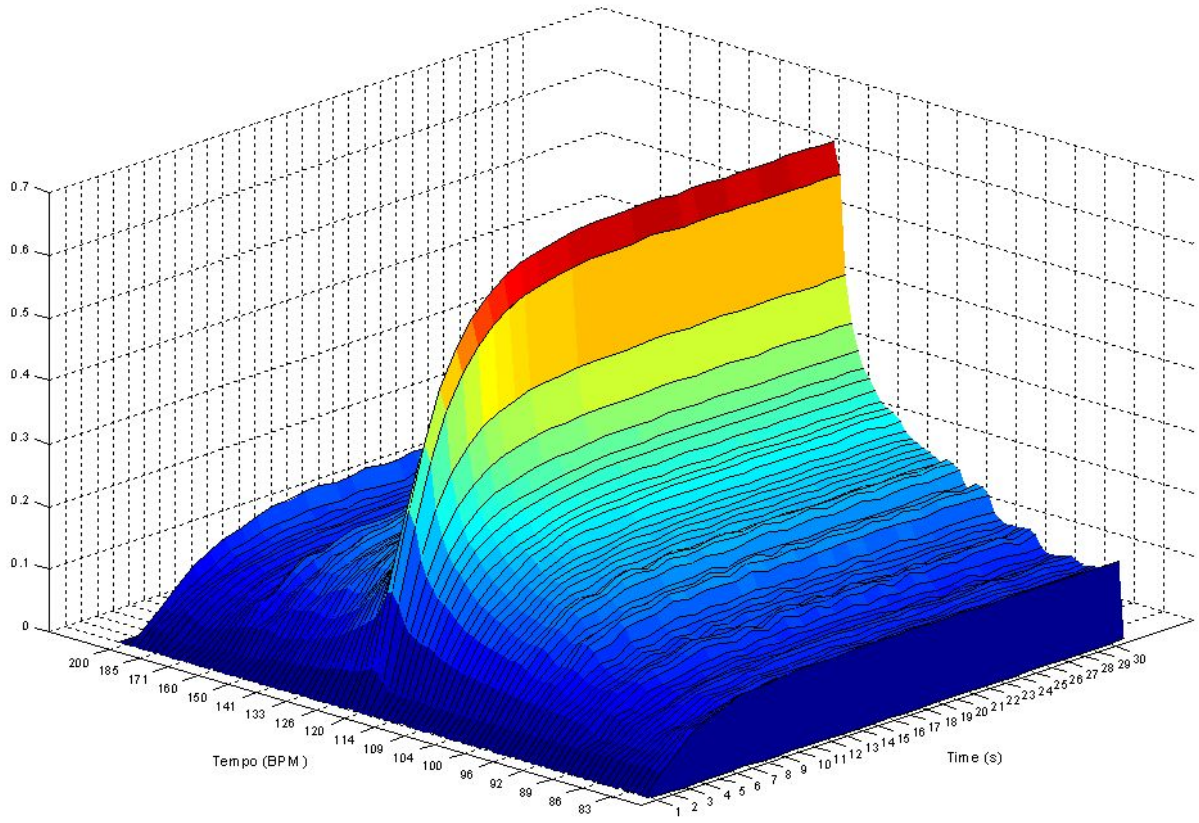

Figure 7.1. Tempo estimates over time for an ideal 120 BPM test signal.

Apart from the actual songs above, a test signal featuring a combined bass drum/snare/hihat sample banging away at exactly 120 BPM was constructed in order to test the proper functioning of the system.

7.2.2 Test Results

To aid in the testing process, I wrote some small Matlab routines for reading the test data produced by the beat tracker and visualise it in various ways. Figure 7.1 shows a visualisation of the estimated tempo of the ideal 120 BPM test signal over time. Time is running along the x axis, the y axis represents the resonators tuned to different tempi and the z axis shows summed resonator energy output. The correct tempo of 120 BPM is represented by the tall "shark fin" in the middle.

Using parameter settings arrived at after numerous test runs, the results in Table 7.2 were achieved for the test set. It should be pointed out, however, that there are many parameters to experiment with, and far from all combinations have been investigated. It's possible, and highly likely, that better performance can be achieved at the same computational cost by studying the effects of the parameters more closely. All test results were produced with the accuracy set to 3.

There is one column per song in the test set and the rows represent consecutive time instants in seconds from 0 to 30. The results shown is the post-processed output fed through the RangeProcessor after beat tracking.

Looking at the results, the conclusion must be that the implementation is fairly successful. Of the 20 excerpts, 19 are tracked correctly for the majority of their 30-second duration. For some of those 19, the algorithm has chosen double, or half, the tempo as the estimated one. For "Death to the Natives" (#14), the estimate is 95 BPM instead of the correct 190 BPM, and for "Oops, Oh My"

(#19), the estimate is 160 BPM instead of the correct 80 BPM. Furthermore, the algorithm sometimes changes its estimate to double the previous tempo, like in "Stranger on the Shore" (#1) which starts off being tracked at around 87 BPM and then switches to the double halfway through. Although not ideal, I have classed these as correct estimates since it's essentially the same tempo.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0.01	0	0	0	0	0	94	0	0	125	0	0	135	81	0	0	0	0	0	0	0
0.02	0	0	0	0	0	95	148	120	125	0	0	136	81	0	0	0	0	0	0	0
0.03	85	0	0	124	0	95	146	117	125	0	0	136	81	0	0	0	0	0	158	0
0.04	84	0	0	125	0	94	145	117	125	119	0	136	81	0	132	0	130	0	160	138
0.05	86	0	106	125	0	94	145	118	124	118	0	138	81	0	132	128	130	0	158	138
0.06	87	0	106	125	129	94	145	117	124	120	0	138	81	0	132	132	130	0	160	140
0.07	87	0	106	125	129	94	143	118	124	119	0	140	81	0	132	132	129	0	160	140
0.08	86	0	106	125	129	94	145	117	125	118	0	150	81	0	132	133	129	0	160	140
0.09	86	0	106	125	130	92	146	117	128	118	0	154	81	0	132	133	130	121	160	138
0.10	86	0	107	125	130	92	148	117	128	118	0	158	84	94	132	132	130	124	160	138
0.11	86	0	106	125	130	92	148	117	128	96	0	162	84	95	132	132	130	122	160	140
0.12	88	0	107	125	130	92	148	117	128	96	0	167	83	94	132	132	130	121	160	140
0.13	88	0	107	125	129	140	148	118	128	94	0	169	83	95	132	132	130	121	160	140
0.14	88	0	107	125	129	140	148	117	128	95	106	171	83	95	132	132	130	121	160	141
0.15	176	0	107	125	129	138	148	117	128	94	106	174	84	95	132	133	130	121	160	141
0.16	174	0	107	125	129	140	148	117	128	95	106	179	84	95	132	135	130	121	160	141
0.17	172	0	107	125	129	138	148	117	126	94	106	182	84	95	132	135	130	120	160	141
0.18	176	0	107	125	129	138	148	117	126	95	107	185	84	95	132	135	130	120	160	141
0.19	176	0	107	125	129	136	148	117	126	95	107	190	167	95	132	135	130	120	160	141
0.20	176	156	108	125	129	135	148	117	126	94	107	194	169	95	132	133	130	120	160	141
0.21	176	156	108	125	130	135	148	117	126	95	109	197	169	95	132	133	130	120	160	141
0.22	176	160	107	125	129	135	148	117	128	95	108	197	169	95	132	133	130	120	160	141
0.23	176	162	108	125	129	135	150	117	128	95	108	197	171	95	132	133	130	121	160	141
0.24	176	162	108	125	129	135	148	118	126	95	108	197	171	95	132	135	130	121	160	141
0.25	176	162	108	125	129	140	148	118	126	94	108	194	169	95	132	138	130	121	160	141
0.26	174	162	108	125	129	140	148	118	126	94	108	194	169	95	132	136	130	120	160	141
0.27	174	162	108	125	129	138	148	118	126	94	108	194	169	95	132	136	130	121	160	141
0.28	174	162	109	125	130	136	148	117	126	95	108	194	171	94	132	136	130	120	160	141
0.29	174	162	109	125	129	138	148	118	126	95	110	194	169	94	132	135	130	120	160	141
0.30	174	162	109	125	130	138	148	118	126	94	109	194	169	94	132	133	130	120	158	141

Table 7.2. Beat tracking test results for the 20 test excerpts.

The one track that failed completely was Boards of Canada's "Kaini Industries" (#2). On closer inspection, it turns out that this excerpt doesn't contain any fluctuations in amplitude to speak of. Instead, the perceived tempo lies in the note changes; a human listener can deduce the tempo from the changes in pitch in the melody line. There are however no amplitude peaks at these onsets, simply a frequency change. Since the algorithm uses the amplitude envelope to detect onsets, it fails to beat track this piece. To be able to successfully beat track this kind of signal, pitch change detection would need to be added. This would be an interesting candidate for future improvements to the algorithm. Some work in this area has already been carried out by Goto (2001), so there is a high chance that it would prove successful.

For most excerpts, the algorithm needs a few seconds to settle down on a stable tempo. For two of them, "Sound of the Police" (#10) and "So What" (#6), it initially picked the wrong tempo only to find the correct one later on. In both of these cases, this is down to an initial section with little rhythmic information. For "Police", the first 10 seconds contain voices only, and in "So What", the

rhythm is not very pronounced for the first 12 seconds or so. Still, a human listener would have no problem working out the rhythm in these intros, so there is clearly room for improvement.

The only test excerpt with a changing tempo was “1000” (#12). Here, the tempo acceleration was followed correctly to the upper limit of the tracked range. When reached, the algorithm seems to have got confused as to where to go next, so it stayed at 197 and then 194 BPM for the remainder of the excerpt whereas it should have switched down to 100 BPM when the tempo reached 200 BPM.

As an interesting contrast to the energy diagram resulting from the ideal 120 BPM signal in Figure 7.1, Figures 7.2 and 7.3 display the plots for two of the excerpts in the test set. 7.2 shows the frantic “Death to the Natives” (#4) and 7.3 shows the noisy “Only Shallow” (#13).

It’s interesting to see how the characteristics of the music are reflected in the plots. The polyrhythmic nature of “Death” can be seen in the many ridges appearing along the timeline indicating that there are strong, continuous rhythmic components at other tempi than the base one. In the case of “Only Shallow”, the noisy nature of the track manifests itself in high energy output across the tempo range, the correct tempo only just peaking above all the others.

7.2.3 Comparison with Traktor

As an indicator of how well the beat tracking measures up to that available in commercial DJ applications, I did a comparison of Beatrak's estimates with those of Native Instruments' Traktor. As Traktor only estimates overall tempo, the Overall tempo setting was used in the comparison. The results can be seen in Table 7.3.

	Actual Tempo	Traktor	Beatrak
1	88.5	43.6	87
2	96.5	133.5	81
3	109	108.4	108
4	125	125.2	125
5	130	173.1	129
6	139	138.2	137
7	149	150	148
8	117.5	184.3	118
9	63.5	-	127
10	95	142.8	95
11	106	108.3	107
12	137-292	137.5	197
13	85	133.2	84
14	190	126.5	95
15	130	132.2	132
16	132.5	133	132
17	130.5	173.9	130
18	121	121.2	121
19	80	159.5	160
20	140	133.1	140

Table 7.3. Comparison of overall tempo estimates by Traktor and Beatrak. Estimates considered correct are shown in green and erroneous ones in red.

There's no question that Beatrak is the clear winner. Traktor only gives the correct overall tempo for 10 of the 20 test excerpts. As the program is designed as a DJ tool for dance music, it's quite surprising to see that it correctly tracked many of the non-dance numbers in the test set, such as “Apache” (#16) and “So What” (#6), but failed on more dance-oriented tracks like “Science Fiction”

(#5) and “Death to the Natives” (#14). However, both of these have the kind of syncopated rhythm that phase constancy weighting compensates for. Presumably, Traktor does not incorporate this notion.

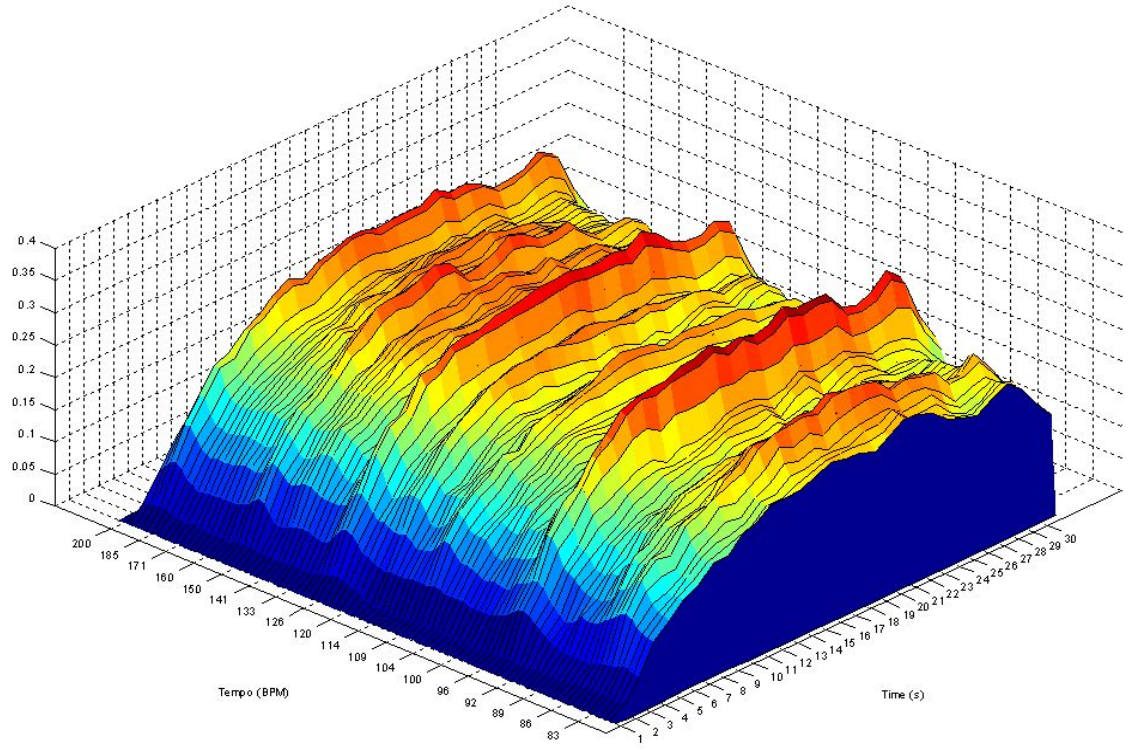

Figure 7.2. Tempo estimates over time for Bogdan Raczynski's "Death to the Natives".

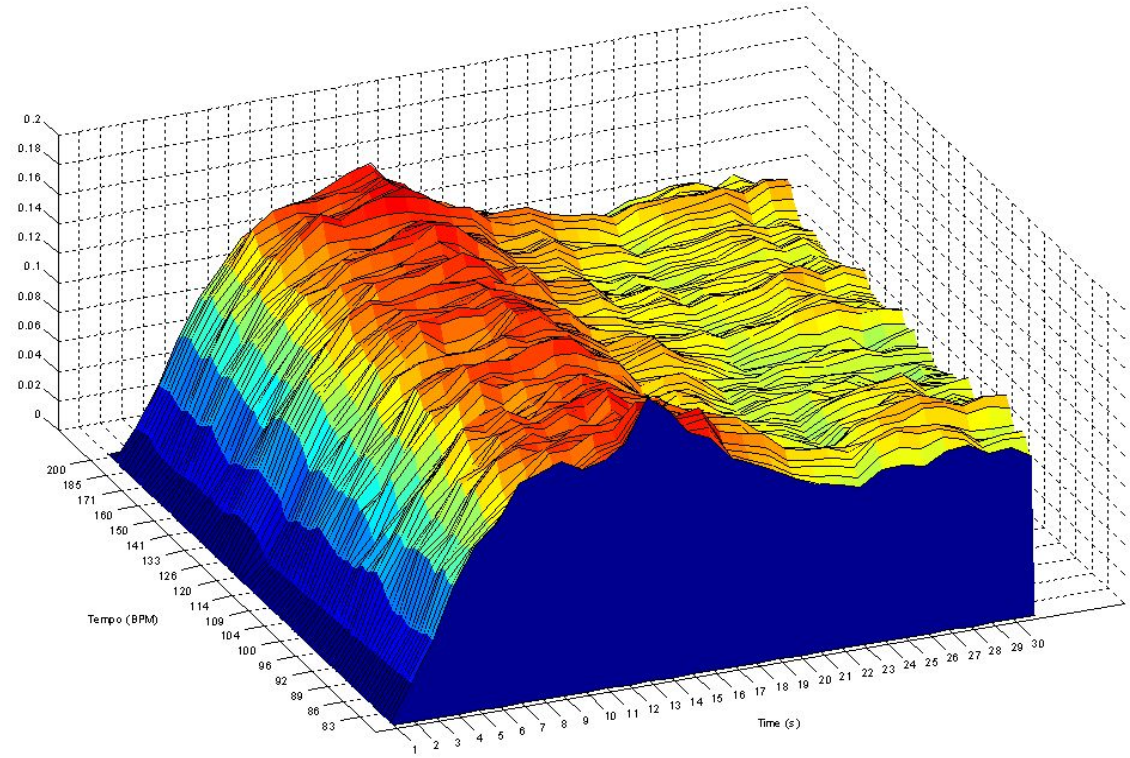

Figure 7.3. Tempo estimates over time for My Bloody Valentine's "Only Shallow".

8. Conclusion

The aim at the outset was to produce a free tool useful for DJs as a convenient way of determining the tempi of music collections. For this purpose, the tempo tracking needed to be accurate and the program easy and intuitive to use.

Overall, the project has been successful. As 19 of 20 test tracks were beat tracked mostly correctly, the beat-tracking aspect of the program would have to be deemed successful. I'm pleasantly surprised at the program being able to correctly determine the tempo of songs like Acker Bilk's "Stranger on the Shore" and Betty Lavette's "Your Turn to Cry", neither of which can be said to have strong rhythmic content.

Considerable effort has been made to produce a nice, usable GUI, and for the most part, this goal has been achieved. The interface adheres to good principles of usability, and is of a sufficiently high standard to rival many commercial applications. Despite this, there are a few niggles. Due to certain limitations of wxWidgets and its need for cross-platform compatibility, some components don't quite measure up to the standards they should ideally meet. Plans for improvements have however been devised. Unfortunately, there wasn't enough time to carry out user tests of the program as initially planned, so I can't give any definite, unbiased evidence for the quality of the user interface.

There are many, many opportunities for extensions and improvements of the application, and my plan is to carry on working on it to enable an Internet release in the near future. The most immediate needs for improvement are to do with the speed of interaction with the database, optimisation of the beat-tracking algorithm and the visual appearance and usability of certain GUI components. Further on, inclusion of other beat-tracking algorithms, support for more audio formats and evaluation of some new ideas for beat tracking will be looked at. The concept of phase constancy also warrants some more investigation.

On a more personal level, two of the main goals of this project were to become proficient in C++ and to learn more about DSP. On this level, the project has been a tremendous success, as I have learnt an incredible amount over the last year, both concerning C++ programming and DSP. C++ is a complex language, much more so than Java, and getting to grips with it has sometimes been frustrating, but also very rewarding. As for DSP, since the project only required knowledge of a few key areas I've really only scratched the surface, but I do hope to be able to dive deeper into this fascinating subject and its applications to music in the future.

Appendix A. Program Code

All the program code is presented in alphabetical order with header file first and implementation file second. Most abstract classes, and some other very simple classes are implemented fully in their header files so don't have a corresponding cpp.

Documentation about what the classes do can be found mainly in the header files and are intended for users of the classes. Each function has a comment preceding it explaining its purpose. Implementation comments explaining certain coding decisions are kept in the implementation files and are intended for maintainers.

Table of Contents

```
/////////////////////////////////////////////////////////////////////////////

  AboutDialog.h

  Author: Erik Jälevik

/////////////////////////////////////////////////////////////////////////////*/

#ifndef ABOUTDIALOG_H
#define ABOUTDIALOG_H

class AboutDialog : public wxDialog {

  public:

    /*////////////////////////////////////////////////////////////////////////
      Parameters are: parent window, window ID.
    ////////////////////////////////////////////////////////////////////////*/
    AboutDialog(wxWindow*, wxWindowID);

    /*////////////////////////////////////////////////////////////////////////
      Destructor
    ////////////////////////////////////////////////////////////////////////*/
    virtual ~AboutDialog() { }

    /*////////////////////////////////////////////////////////////////////////
      Handler for OK button.
    ////////////////////////////////////////////////////////////////////////*/
    void OnOK(wxCommandEvent&);


  private:

    /*////////////////////////////////////////////////////////////////////////
      Autogenerated method for building dialog
    ////////////////////////////////////////////////////////////////////////*/
    void BuildMe();


    /*////////////////////////////////////////////////////////////////////////
      Variables
    ////////////////////////////////////////////////////////////////////////*/
    wxButton* okBut;

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/////////////////////////////////////////////////////////////////////////////

  AboutDialog.cpp

  Author: Erik Jälevik

/////////////////////////////////////////////////////////////////////////////*/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "AboutDialog.h"
#include "Icons.h"

#include <wx/statline.h>

/*////////////////////////////////////////////////////////////////////////////
  Constructor
////////////////////////////////////////////////////////////////////////////*/
AboutDialog::AboutDialog(wxWindow* parent,
                         wxWindowID id) :
  wxDialog(parent, id, "About", wxDefaultPosition, wxDefaultSize,
    wxCAPTION | wxSYSTEM_MENU | wxRESIZE_BORDER) {

  BuildMe();

}

/*////////////////////////////////////////////////////////////////////////////
  Event table
////////////////////////////////////////////////////////////////////////////*/
BEGIN_EVENT_TABLE(AboutDialog, wxDialog)
  EVT_BUTTON(wxID_OK, AboutDialog::OnOK)
END_EVENT_TABLE()

/*////////////////////////////////////////////////////////////////////////////
  OnOK
////////////////////////////////////////////////////////////////////////////*/
void AboutDialog::OnOK(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*////////////////////////////////////////////////////////////////////////////
  BuildMe
////////////////////////////////////////////////////////////////////////////*/
void AboutDialog::BuildMe() {

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticBitmap *item2 = new wxStaticBitmap(this, -1, Icons::Speaker32(),
```

```
    wxDefaultPosition, wxDefaultSize);
  item1->Add(item2, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5);

  wxBoxSizer *item3 = new wxBoxSizer( wxVERTICAL );

  wxStaticText *item4 = new wxStaticText(this, -1, "Beatrak 0.1",
    wxDefaultPosition, wxDefaultSize, 0);
  item4->SetFont(wxFont(12, wxSWISS, wxNORMAL, wxBOLD));
  item3->Add(item4, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

  wxStaticText *item5 = new wxStaticText(this, -1,
    "© Erik Jälevik 2004",
    wxDefaultPosition, wxDefaultSize, 0);
  item3->Add( item5, 0, wxALIGN_LEFT|wxALL, 5 );

  item3->Add(12, 12);

  wxStaticText *item7 = new wxStaticText(this, -1,
    "Thanks to the creators of the following libraries,\n"
    "without which this program would never have seen the light of day:\n\n"
    "wxWidgets\nSTK\nMAD\nAKRip\nSQLite\nID3Lib\n",
    wxDefaultPosition, wxDefaultSize, 0);
  item3->Add( item7, 0, wxALIGN_LEFT|wxALL, 5 );

  item1->Add( item3, 0, wxALIGN_CENTER|wxALL, 5 );

  item0->Add( item1, 0, wxALIGN_CENTER|wxTOP|wxLEFT|wxRIGHT, 5 );

  wxStaticLine *item6 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL);
  item0->Add(item6, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  okBut = new wxButton(this, wxID_OK, "OK", wxDefaultPosition,
    wxDefaultSize, 0);
  item0->Add(okBut, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 10);

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}
```

```
/*****************************************************************************

  AddCDDialog.h

  GUI dialog window for adding tracks from CD.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef ADDCDDIALOG_H
#define ADDCDDIALOG_H

#include "AddDialog.h"
#include "CDInStream.h"
#include "FreeDB.h"

class AddCDDialog : public AddDialog {

  public:

    /*************************************************************************
      Parameters are parent window and ID.
    *************************************************************************/
    AddCDDialog(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~AddCDDialog();

    /*************************************************************************
      Returns a vector of the selected tracks as MusicTrack objects.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<MusicTrack> > GetSelectedTracks();

    /*************************************************************************
      Refreshes display when drive is changed.
    *************************************************************************/
    void OnDriveChange(wxCommandEvent&);

    /*************************************************************************
      Selects all tracks in list control.
    *************************************************************************/
    void OnSelectAll(wxCommandEvent&);

    /*************************************************************************
      Deselects all tracks in list control.
    *************************************************************************/
    void OnDeselectAll(wxCommandEvent&);

    /*************************************************************************
      Refreshes list by polling CD drive.
    *************************************************************************/
    void OnRefresh(wxCommandEvent&);

    /*************************************************************************
      Queries FreeDB for CD info.
    *************************************************************************/
    void OnFreeDB(wxCommandEvent&);

    /*************************************************************************
      Changes display to that of a compilation disc.
    *************************************************************************/
    void OnCompilation(wxCommandEvent&);

    /*************************************************************************
      OK.
```

```
  ***************************************************************************/
  void OnOK(wxCommandEvent&);

  /***************************************************************************
    Cancel.
  ***************************************************************************/
  void OnCancel(wxCommandEvent&);


private:

  /***************************************************************************
    Polls CD for number of tracks and displays them, returns false if
    no CD found
  ***************************************************************************/
  bool PollCD();

  /***************************************************************************
    Queries FreeDB for CD info and displays it
  ***************************************************************************/
  void QueryFreeDB();

  /***************************************************************************
    Populates dialog with info currently in tracks vector
  ***************************************************************************/
  void Populate();

  /***************************************************************************
    Autogenerated method for building dialog
  ***************************************************************************/
  void BuildMe();


  /***************************************************************************
    Variables
  ***************************************************************************/

  // Controls
  wxChoice* driveDD;

  wxCheckListBox* tocCLB;
  wxBitmapButton* selectAllBut;
  wxBitmapButton* deselectAllBut;
  wxBitmapButton* refreshBut;
  wxBitmapButton* freeDBBut;
  wxCheckBox* autoQueryCB;

  wxTextCtrl* artistTC;
  wxTextCtrl* albumTC;
  wxChoice* genreDD;
  wxCheckBox* compilationCB;

  wxButton* okBut;
  wxButton* cancelBut;

  // Model objects
  std::auto_ptr<std::vector<MusicTrack> > tracks; // current tracks
  CDInStream cdStream; // for finding no of drives and no of CD tracks
  FreeDB freeDB;

  bool autoQuery;

  enum {
    ID_DRIVE_DD,
    ID_TOC_CLB,
    ID_SELECTALL_BUT,
    ID_DESELECTALL_BUT,
    ID_REFRESH_BUT,
    ID_FREEDB_BUT,
    ID_AUTOQUERY_CB,
    ID_ARTIST_TC,
    ID_ALBUM_TC,
    ID_GENRE_DD,
    ID_COMPILATION_CB
  };

  DECLARE_EVENT_TABLE()

};

#endif
```

```
/*****************************************************************************

  AddCDDialog.cpp

  GUI dialog window for adding tracks from CD.

  Author: Erik Jälevik

*****************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "AddCDDialog.h"
#include "MusicCollection.h"
#include "Globals.h"
#include "Icons.h"

#include <wx/confbase.h>
#include <wx/statline.h>

using namespace std;

/*****************************************************************************
  Constructor
*****************************************************************************/
AddCDDialog::AddCDDialog(wxWindow* parent, wxWindowID id) :
  AddDialog(parent, -1, "Add CD"),
  tracks(new vector<MusicTrack>) {

  BuildMe();

  CentreOnParent();

  Show(true);

  PollCD();

}

/*****************************************************************************
  Destructor
*****************************************************************************/
AddCDDialog::~AddCDDialog() {

  wxConfigBase* cfg = wxConfigBase::Get();
  cfg->Write("/AddCDDialog/drive", driveDD->GetSelection());
  cfg->Write("/AddCDDialog/autoQuery", autoQueryCB->GetValue());

}

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(AddCDDialog, AddDialog)
  EVT_CHOICE(ID_DRIVE_DD, AddCDDialog::OnDriveChange)
  EVT_BUTTON(ID_SELECTALL_BUT, AddCDDialog::OnSelectAll)
  EVT_BUTTON(ID_DESELECTALL_BUT, AddCDDialog::OnDeselectAll)
  EVT_BUTTON(ID_REFRESH_BUT, AddCDDialog::OnRefresh)
  EVT_BUTTON(ID_FREEDB_BUT, AddCDDialog::OnFreeDB)
  EVT_CHECKBOX(ID_COMPILATION_CB, AddCDDialog::OnCompilation)
  EVT_BUTTON(wxID_OK, AddCDDialog::OnOK)
  EVT_BUTTON(wxID_CANCEL, AddCDDialog::OnCancel)
END_EVENT_TABLE()

/*****************************************************************************
  GetSelectedTracks
*****************************************************************************/
auto_ptr<vector<MusicTrack> > AddCDDialog::GetSelectedTracks() {

  auto_ptr<vector<MusicTrack> > v(new vector<MusicTrack>);

  for (int i = 0; i < tocCLB->GetCount(); ++i) {

    if (tocCLB->IsChecked(i)) {

      // Update any changes user has made
      tracks->at(i).SetTitle(tocCLB->GetString(i).c_str());
      tracks->at(i).SetArtist(artistTC->GetValue().c_str());
      tracks->at(i).SetSource(albumTC->GetValue().c_str());
      tracks->at(i).SetGenre(genreDD->GetStringSelection().c_str());

      if (compilationCB->IsChecked()) {

        // Parse for artist in title
        string tit = tracks->at(i).GetTitle();

        // Try /
        string::size_type idx = tit.find(" / ");
        if (idx == string::npos) {
          // Try -
          idx = tit.find(" - ");
        }

        if (idx != string::npos) {
          string art = tit.substr(0, idx);
          tit.erase(0, idx + 3);
          tracks->at(i).SetArtist(art);
          tracks->at(i).SetTitle(tit);
        }

      } // end if compilation

    v->push_back(tracks->at(i));

    } // end if is checked

  } // end for

  return v;

}

/*****************************************************************************
  OnDriveChange
*****************************************************************************/
void AddCDDialog::OnDriveChange(wxCommandEvent& event) {

  PollCD();

}

/*****************************************************************************
  OnSelectAll
*****************************************************************************/
void AddCDDialog::OnSelectAll(wxCommandEvent& event) {

  for (int i = 0; i < tocCLB->GetCount(); ++i) {
    tocCLB->Check(i, TRUE);
```

```
  }

}

/*****************************************************************************
  OnDeselectAll
*****************************************************************************/
void AddCDDialog::OnDeselectAll(wxCommandEvent& event) {

  for (int i = 0; i < tocCLB->GetCount(); ++i) {
    tocCLB->Check(i, FALSE);
  }

}

/*****************************************************************************
  OnRefresh
*****************************************************************************/
void AddCDDialog::OnRefresh(wxCommandEvent& event) {

  PollCD();

}

/*****************************************************************************
  OnFreeDB
*****************************************************************************/
void AddCDDialog::OnFreeDB(wxCommandEvent& event) {

  if (PollCD() && !autoQueryCB->IsChecked()) {
    QueryFreeDB();
  }

}

/*****************************************************************************
  OnCompilation
*****************************************************************************/
void AddCDDialog::OnCompilation(wxCommandEvent& event) {

}

/*****************************************************************************
  OnOK
*****************************************************************************/
void AddCDDialog::OnOK(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*****************************************************************************
  OnCancel
*****************************************************************************/
void AddCDDialog::OnCancel(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_CANCEL);
  }
  else {
    SetReturnCode(wxID_CANCEL);
    Show(FALSE);
    Destroy();
  }

}

/*****************************************************************************
  PollCD
*****************************************************************************/
bool AddCDDialog::PollCD() {

  int device = driveDD->GetSelection();
  auto_ptr<vector<uint> > lengths;
  try {
    lengths = cdStream.GetTrackLengths(device);
  }
  catch (InStream::InStreamException& e) {
    tocCLB->Clear();
    if (IsShown()) {
      ::wxMessageBox("No CD found.", "CD Error",
        wxOK | wxICON_EXCLAMATION | wxCENTRE);
      //::wxMessageBox(e.what(), "CD Error",
      //  wxOK | wxICON_EXCLAMATION | wxCENTRE);
    }
    return false;
  }

  tracks->clear();

  for (size_t i = 0; i < lengths->size(); ++i) {

    MusicTrack t;
    t.SetArtist("Unknown");
    t.SetSource("Unknown");
    t.SetLength(lengths->at(i));
    t.SetFormat("CD");

    ostringstream os;
    os << "Track " << (i + 1);
    t.SetTitle(os.str());

    os.str("");
    os << device;
    if ((i + 1) < 10) {
      os << "0";
    }
    os << (i + 1);
    t.SetAccess(os.str());

    tracks->push_back(t);

  }

  Populate();

  ::wxSafeYield();

  if (autoQueryCB->IsChecked()) {
    QueryFreeDB();
  }

  return true;

}

/*****************************************************************************
  QueryFreeDB
*****************************************************************************/
```

```
void AddCDDialog::QueryFreeDB() {

  try {

    ::wxBeginBusyCursor();

    // Get FreeDB options from config
    wxConfigBase* cfg = wxConfigBase::Get();
    wxString url = cfg->Read("/Options/FreeDB/url",
      "www.freedb.org/~cddb/cddb.cgi");
    long port = cfg->Read("/Options/FreeDB/port", 80L);

    // Configure FreeDB
    freeDB.SetUrl(url.c_str()); // can throw
    freeDB.SetPort(port);

    // Perform query
    auto_ptr<vector<string> > matches =
      freeDB.QueryCD(driveDD->GetSelection());

    size_t nMatches = matches->size();
    wxString* matchesArr = new wxString[nMatches];
    for (size_t i = 0; i < nMatches; ++i) {
      matchesArr[i] = matches->at(i).c_str();
    }

    wxSingleChoiceDialog dlg(this,
      "The following matches were found. Please select the right one or "
        "press Cancel if none is correct.",
      "FreeDB Matches",
      nMatches, matchesArr, NULL,
      wxOK | wxCANCEL | wxCENTRE | wxCAPTION);
    dlg.SetSelection(0);
    delete[] matchesArr;

    ::wxEndBusyCursor();

    // If OK, query FreeDB for details of that album
    if (dlg.ShowModal() == wxID_OK) {

      ::wxSafeYield();
      ::wxBeginBusyCursor();

      int choice = dlg.GetSelection();
      auto_ptr<FreeDBAlbum> album = freeDB.GetCDContents(choice);

      // Copy details into tracks
      for (size_t i = 0; i < tracks->size() && i < album->tracks.size(); ++i) {
        tracks->at(i).SetArtist(album->artist);
        tracks->at(i).SetSource(album->title);
        tracks->at(i).SetTitle(album->tracks.at(i));
        tracks->at(i).SetGenre(album->genre);
      }

      Populate();

      ::wxEndBusyCursor();

    }

  }
  catch (FreeDB::FreeDBException& e) {
    ::wxEndBusyCursor();
    ::wxMessageBox(e.what(), "FreeDB Error", wxOK | wxICON_ERROR | wxCENTRE);
  }
}
```

```
/*********************************************************************************
  Populate
*********************************************************************************/
void AddCDDialog::Populate() {

  // Do tracks window
  size_t nTracks = tracks->size();
  wxString* tracksArr = new wxString[nTracks];
  for (size_t i = 0; i < nTracks; ++i) {
    tracksArr[i] = tracks->at(i).GetTitle().c_str();
    tracksArr[i].Replace("&", "&&");
  }
  tocCLB->Set(nTracks, tracksArr);
  delete[] tracksArr;

  // Do other widgets
  wxString art = tracks->at(0).GetArtist().c_str();
  artistTC->SetValue(art);
  if (art.StartsWith("Various")) {
    compilationCB->SetValue(true);
  }
  else {
    compilationCB->SetValue(false);
  }

  albumTC->SetValue(tracks->at(0).GetSource().c_str());

  wxString genre = tracks->at(0).GetGenre().c_str();
  if (!genre.IsEmpty()) {

    // Check if genre is one of the existing ones,
    // if not add it to database and drop down menu
    if (genreDD->FindString(genre) == -1) {
      string g = genre.c_str();
      try {
        MusicCollection::Instance()->AddGenre(g);
      }
      catch (MusicCollection::DatabaseException& e) {
        ::wxMessageBox(e.what(), "Database Error",
          wxOK | wxICON_ERROR | wxCENTRE);
      }
      genreDD->Append(genre);
    }

    genreDD->SetStringSelection(genre);
  }

}


/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void AddCDDialog::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();
  long driveVal = cfg->Read("/AddCDDialog/drive", 0L);
  bool autoQueryVal;
  cfg->Read("/AddCDDialog/autoQuery", &autoQueryVal, false);

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticBox *item3 = new wxStaticBox(this, -1, "Select tracks");
  wxStaticBoxSizer *item2 = new wxStaticBoxSizer( item3, wxVERTICAL );
```

```
  wxBoxSizer* item4 = new wxBoxSizer(wxHORIZONTAL);

  wxStaticText* item6 = new wxStaticText(this, -1, "CD drive",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT);
  item4->Add(item6, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  // Get all CD drives and put into drop down box
  auto_ptr<vector<string> > drivesVec = cdStream.GetDrives();
  size_t nDrives = drivesVec->size();
  wxString* drivesArr = new wxString[nDrives];
  for (uint i = 0; i < nDrives; ++i) {
    drivesArr[i] = (*drivesVec)[i].c_str();
  }
  driveDD = new wxChoice(this, ID_DRIVE_DD, wxDefaultPosition,
    wxSize(100, -1), nDrives, drivesArr, 0);
  driveDD->SetSelection(driveVal);
  item4->Add(driveDD, 1, wxALIGN_CENTER_VERTICAL|wxALL, 5);
  delete[] drivesArr;

  item2->Add(item4, 0, wxGROW|wxALIGN_CENTER|wxALL, 5);

  tocCLB = new wxCheckListBox(this, ID_TOC_CLB, wxDefaultPosition,
    wxSize(290, 290), 0, NULL, 0);
  item2->Add(tocCLB, 1, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  // Toolbar version did not work

  wxBoxSizer *item5 = new wxBoxSizer( wxHORIZONTAL );
  item5->Add(4, 1, 0);

  // Buttons version
  selectAllBut = new wxBitmapButton(this, ID_SELECTALL_BUT,
    Icons::CheckedBox(), wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  selectAllBut->SetToolTip("Select all tracks");
  item5->Add(selectAllBut, 0, wxALIGN_CENTER|wxALL, 1);

  deselectAllBut = new wxBitmapButton(this, ID_DESELECTALL_BUT,
    Icons::UncheckedBox(), wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  deselectAllBut->SetToolTip("Deselect all tracks");
  item5->Add(deselectAllBut, 0, wxALIGN_CENTER|wxALL, 1);

//  wxStaticLine *item8 = new wxStaticLine( this, -1, wxDefaultPosition,
//    wxSize(-1,20), wxLI_VERTICAL );
//  item5->Add( item8, 0, wxGROW|wxALIGN_CENTER_HORIZONTAL|wxALL, 5);

  refreshBut = new wxBitmapButton(this, ID_REFRESH_BUT, Icons::Refresh(),
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  refreshBut->SetToolTip("Refresh track listing");
  item5->Add(refreshBut, 0, wxALIGN_CENTER|wxALL, 1);

//  wxStaticLine *item10 = new wxStaticLine( this, -1, wxDefaultPosition,
//    wxSize(-1,20), wxLI_VERTICAL );
//  item5->Add( item10, 0, wxALIGN_CENTER|wxALL, 5);

  freeDBBut = new wxBitmapButton(this, ID_FREEDB_BUT, Icons::FreeDB(),
    wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  freeDBBut->SetToolTip("Query FreeDB for CD information");
  item5->Add(freeDBBut, 0, wxALIGN_CENTER|wxALL, 1);

  autoQueryCB = new wxCheckBox(this, ID_AUTOQUERY_CB, "Autoquery FreeDB",
    wxDefaultPosition, wxDefaultSize, 0 );
  autoQueryCB->SetToolTip("If checked, FreeDB will be queried automatically on
refresh");
  autoQueryCB->SetValue(autoQueryVal);

  item5->Add(autoQueryCB, 0, wxALIGN_CENTER|wxALL, 5);

  item2->Add(item5, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

  item1->Add(item2, 1, wxGROW|wxALIGN_CENTER|wxALL, 5);

  wxStaticBox *item13 = new wxStaticBox(this, -1, "CD details");
  wxStaticBoxSizer *item12 = new wxStaticBoxSizer( item13, wxVERTICAL );

  wxFlexGridSizer *item14 = new wxFlexGridSizer( 2, 0, 0 );

  wxStaticText *item15 = new wxStaticText( this, -1, "Artist",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item14->Add( item15, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  artistTC = new wxTextCtrl(this, ID_ARTIST_TC, "", wxDefaultPosition,
    wxSize(140,-1), 0);
  item14->Add(artistTC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  wxStaticText *item17 = new wxStaticText( this, -1, "Album",
    wxDefaultPosition, wxDefaultSize, 0 );
  item14->Add( item17, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  albumTC = new wxTextCtrl( this, ID_ALBUM_TC, "", wxDefaultPosition,
    wxSize(80,-1), 0 );
  item14->Add(albumTC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxStaticText *item19 = new wxStaticText( this, -1, "Genre",
    wxDefaultPosition, wxDefaultSize, 0 );
  item14->Add( item19, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  // Get all genres and put into drop down box
  const vector<string>& genreVec =
    MusicCollection::Instance()->GetGenres();
  size_t nGenres = genreVec.size();
  wxString* genreArr = new wxString[nGenres];
  for (uint i = 0; i < nGenres; ++i) {
    genreArr[i] = genreVec[i].c_str();
  }
  genreDD = new wxChoice(this, ID_GENRE_DD, wxDefaultPosition,
    wxSize(100,-1), nGenres, genreArr, 0);
  item14->Add(genreDD, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5);
  delete[] genreArr;

  item14->Add( 20, 20, 0, wxALIGN_CENTER|wxALL, 5 );

  compilationCB = new wxCheckBox(this, ID_COMPILATION_CB, "Compilation",
    wxDefaultPosition, wxDefaultSize, 0 );
  item14->Add(compilationCB, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

  item12->Add( item14, 0, wxALIGN_CENTER|wxALL, 5 );

  item1->Add( item12, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5 );

  item0->Add( item1, 1, wxGROW|wxALIGN_CENTER|wxALL, 5 );

  wxStaticLine *item22 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL);
  item0->Add(item22, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  wxBoxSizer *item23 = new wxBoxSizer( wxHORIZONTAL );

  okBut = new wxButton(this, wxID_OK, "OK", wxDefaultPosition,
    wxDefaultSize, 0);
  okBut->SetDefault();
  item23->Add(okBut, 0, wxALIGN_CENTER|wxALL, 5);

  cancelBut = new wxButton(this, wxID_CANCEL, "Cancel",
    wxDefaultPosition, wxDefaultSize, 0);
  item23->Add(cancelBut, 0, wxALIGN_CENTER|wxALL, 5);
```

```
  item0->Add( item23, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}
```

```
/*****************************************************************************

  AddDialog.h

  Abstract base class for all dialogs for adding tracks to the collection.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef ADDDIALOG_H
#define ADDDIALOG_H

#include "MusicTrack.h"

#include <vector> // returned by GetSelectedTracks
#include <memory> // auto_ptr

class AddDialog : public wxDialog {

  public:

    /*************************************************************************
      Parameters are parent window, ID and frame caption.
    *************************************************************************/
    AddDialog(wxWindow* parent, wxWindowID id, const wxString& caption) :
      wxDialog(parent,
               id,
               caption,
               wxDefaultPosition,
               wxDefaultSize,
               wxCAPTION | wxSYSTEM_MENU | wxRESIZE_BORDER | wxMAXIMIZE_BOX) {}

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~AddDialog() {}

    /*************************************************************************
      Returns the selected files as a vector of MusicTracks.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<MusicTrack> > GetSelectedTracks() = 0;

};

#endif
```

```
/*****************************************************************************

  AddFilesDialog.h

  GUI dialog window for adding files.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef ADDFILESDIALOG_H
#define ADDFILESDIALOG_H

#include "AddDialog.h"
#include "CheckableDirCtrl.h"

class AddFilesDialog : public AddDialog {

  public:

    /*************************************************************************
      Parameters are parent window and ID.
    *************************************************************************/
    AddFilesDialog(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~AddFilesDialog();

    /*************************************************************************
      Returns the selected files as a vector of MusicTracks.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<MusicTrack> > GetSelectedTracks();

    /*************************************************************************
      Returns the user's choice of where to first look for name info.
    *************************************************************************/
    virtual bool LookInMetaFirst();

    /*************************************************************************
      Automatically changes other drop down when one is changed.
    *************************************************************************/
    void OnLookChange(wxCommandEvent&);

    /*************************************************************************
      Handler for OK button.
    *************************************************************************/
    void OnOK(wxCommandEvent&);

    /*************************************************************************
      Handler for Cancel button.
    *************************************************************************/
    void OnCancel(wxCommandEvent&);


  private:

    /*************************************************************************
      Calls the ID3 reading library to find artist, title and genre for the
      passed track and fills the relevant fields.
    *************************************************************************/
    void ParseID3(MusicTrack& track);

    /*************************************************************************
      Autogenerated method for building dialog
    *************************************************************************/
    void BuildMe();

    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    CheckableDirCtrl* dirTr;
    wxChoice* look1DD;
    wxChoice* look2DD;

    wxButton* okBut;
    wxButton* cancelBut;
    wxButton* applyBut;

    enum {
      ID_SELECTFILES_TR,
      ID_SELECTEDFILES_LC,
      ID_LOOK1_DD,
      ID_LOOK2_DD
    };

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/***************************************************************************

  AddFilesDialog.cpp

  GUI dialog window for adding files.

  Author: Erik Jälevik

***************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "AddFilesDialog.h"
#include "MusicCollection.h"
#include "FilenameParser.h"
#include "Globals.h"

#include "id3/tag.h"
#include <wx/confbase.h>
#include <wx/statline.h>

#include <algorithm>  // binary_search, transform

using namespace std;

/***************************************************************************
  Constructor
***************************************************************************/
AddFilesDialog::AddFilesDialog(wxWindow* parent,
                               wxWindowID id) :
  AddDialog(parent, id, "Add Files") {

  BuildMe();

  CentreOnParent();

}

/***************************************************************************
  Destructor
***************************************************************************/
AddFilesDialog::~AddFilesDialog() {

  wxConfigBase* cfg = wxConfigBase::Get();
  cfg->Write("/AddFilesDialog/look1", look1DD->GetSelection());
  cfg->Write("/AddFilesDialog/look2", look2DD->GetSelection());
  cfg->Write("/AddFilesDialog/selectedPath", dirTr->GetPath());
  wxSize size = GetSize();
  cfg->Write("/AddFilesDialog/width", size.GetWidth());
  cfg->Write("/AddFilesDialog/height", size.GetHeight());

}

/***************************************************************************
  Event table
***************************************************************************/
BEGIN_EVENT_TABLE(AddFilesDialog, AddDialog)
  EVT_BUTTON(wxID_OK, AddFilesDialog::OnOK)
  EVT_BUTTON(wxID_CANCEL, AddFilesDialog::OnCancel)
  EVT_CHOICE(ID_LOOK1_DD, AddFilesDialog::OnLookChange)
  EVT_CHOICE(ID_LOOK2_DD, AddFilesDialog::OnLookChange)
END_EVENT_TABLE()

/***************************************************************************
  GetSelectedTracks
***************************************************************************/
auto_ptr<vector<MusicTrack> > AddFilesDialog::GetSelectedTracks() {

  // Get all existing genres for comparison further down
  MusicCollection* coll = MusicCollection::Instance();
  const vector<string>& genres = coll->GetGenres();

  // Get meta data choice
  bool metaFirst = LookInMetaFirst();

  // Get file names from dir ctrl
  auto_ptr<wxArrayString> files = dirTr->GetSelectedFiles();

  FilenameParser fp;

  // Create MusicTracks from them and put in tracks vector
  auto_ptr<vector<MusicTrack> > tracks(new vector<MusicTrack>);
  for (size_t i = 0; i < files->GetCount(); ++i) {

    MusicTrack track;

    // Send filename to filename parser
    fp.Parse(files->Item(i).c_str());

    // Get format and path from it
    string format = fp.GetExtension();
    transform(format.begin(), format.end(), format.begin(), toupper);
    track.SetFormat(format);
    track.SetAccess(fp.GetPath());
    track.SetSource(fp.GetPath());

    if (metaFirst) {

      // Do meta first
      if (track.GetFormat() == "MP3") {
        ParseID3(track);
      }
      else {
        // should do the WAV meta thing one day
      }

      // Do filename second
      if (track.GetArtist() == "") { track.SetArtist(fp.GetArtist()); }
      if (track.GetTitle() == "") { track.SetTitle(fp.GetTitle()); }

    }
    else {

      // Do filename first
      track.SetArtist(fp.GetArtist());
      track.SetTitle(fp.GetTitle());

      // Do meta second
      if (track.GetFormat() == "MP3") {
        ParseID3(track);
      }
      else {
        // do the WAV meta thing
      }

    }

    // Check if genre is one of the existing ones, if not add it to database
```

```
    wxString genre = track.GetGenre().c_str();
    if (!genre.IsEmpty()) {
      if (!binary_search(genres.begin(), genres.end(), genre.c_str())) {
        string g = genre.c_str();
        coll->AddGenre(g);
      }
    }

    tracks->push_back(track);
  }

  return tracks;

}

/*********************************************************************
  LookInMetaFirst
*********************************************************************/
bool AddFilesDialog::LookInMetaFirst() {

  return look1DD->GetStringSelection() == "ID3 tag";

}

/*********************************************************************
  OnLookChange
*********************************************************************/
void AddFilesDialog::OnLookChange(wxCommandEvent& event) {

  // Automatically change the other look drop down
  if (event.GetId() == ID_LOOK1_DD) {
    look1DD->GetSelection() == 0 ?
      look2DD->SetSelection(1) :
      look2DD->SetSelection(0);
  }
  else {
    look2DD->GetSelection() == 0 ?
      look1DD->SetSelection(1) :
      look1DD->SetSelection(0);
  }

}

/*********************************************************************
  OnOK
*********************************************************************/
void AddFilesDialog::OnOK(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************
  OnCancel
*********************************************************************/
void AddFilesDialog::OnCancel(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_CANCEL);
  }
  else {
    SetReturnCode(wxID_CANCEL);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************
  ParseID3
*********************************************************************/
void AddFilesDialog::ParseID3(MusicTrack& track) {

  // Frame and field constants can be found in the id3lib file globals.h

  // TODO : id3lib is slow and not very good at all. Try using Rob Leslie's
  // id3 library instead when time permits.

  ID3_Frame* frame;
  ID3_Field* field;
  const char* cstr;
  string str;

  ID3_Tag tag = ID3_Tag(track.GetSource().c_str());

  // Artist
  if (track.GetArtist() == "") {
    if ((frame = tag.Find(ID3FID_LEADARTIST)) != NULL) {
      if ((field = frame->GetField(ID3FN_TEXT)) != NULL) {
        cstr = field->GetRawText();
        str = cstr == NULL ? "" : cstr;
        str.erase(0, str.find_first_not_of(" \t\n")); // trim
        str.erase(str.find_last_not_of(" \t\n") + 1); // whitespace
        track.SetArtist(str);
      }
    }
  }

  // Title
  if (track.GetTitle() == "") {
    if ((frame = tag.Find(ID3FID_TITLE)) != NULL) {
      if ((field = frame->GetField(ID3FN_TEXT)) != NULL) {
        cstr = field->GetRawText();
        str = cstr == NULL ? "" : cstr;
        str.erase(0, str.find_first_not_of(" \t\n")); // trim
        str.erase(str.find_last_not_of(" \t\n") + 1); // whitespace
        track.SetTitle(str);
      }
    }
  }

  // Genre
  if (track.GetGenre() == "") {
    if ((frame = tag.Find(ID3FID_CONTENTTYPE)) != NULL) {
      if ((field = frame->GetField(ID3FN_TEXT)) != NULL) {
        cstr = field->GetRawText();
        cstr = cstr == NULL ? "" : cstr;
        str = cstr;
        str.erase(0, str.find_first_not_of(" \t\n")); // trim
        str.erase(str.find_last_not_of(" \t\n") + 1); // whitespace

        // If genre is a bracketed number, need to look it up
        if (str.substr(0, 1) == "(") {
          string::size_type idx = str.find(')', 1);
          int n = atoi(str.substr(1, idx - 1).c_str());
          cstr = ID3_V1GENRE2DESCRIPTION(n);
          cstr = cstr == NULL ? "" : cstr;
        }
```

```
        track.SetGenre(cstr);

      }
    }
  }

/* These are pretty much never present

  if ((frame = tag.Find(ID3FID_BPM)) != NULL) {
    if ((field = frame->GetField(ID3FN_TEXT)) != NULL) {
      value = field->GetRawText();
      LOG("ID3 bpm: " << (value == NULL ? "" : value) << endl);
    }
  }

  if ((frame = tag.Find(ID3FID_SONGLEN)) != NULL) {
    if ((field = frame->GetField(ID3FN_TEXT)) != NULL) {
      value = field->GetRawText();
      LOG("ID3 length: " << (value == NULL ? "" : value) << endl);
    }
  }

  if ((frame = tag.Find(ID3FID_MIXARTIST)) != NULL) {
    if ((field = frame->GetField(ID3FN_TEXT)) != NULL) {
      value = field->GetRawText();
      LOG("ID3 mixartist: " << (value == NULL ? "" : value) << endl);
    }
  }
*/

}

/*****************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*****************************************************************************/
void AddFilesDialog::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString selectedPath = cfg->Read("/AddFilesDialog/selectedPath", "/");
  long look1Val = cfg->Read("/AddFilesDialog/look1", 0L);
  long look2Val = cfg->Read("/AddFilesDialog/look2", 1L);

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticBox *item3 = new wxStaticBox(this, -1, "File Selection");
  wxStaticBoxSizer *item2 = new wxStaticBoxSizer( item3, wxVERTICAL );

  wxStaticText *item4 = new wxStaticText( this, -1, "Select music files",
    wxDefaultPosition, wxDefaultSize, 0 );
  item2->Add( item4, 0, wxGROW | wxALL, 5 );

  dirTr = new CheckableDirCtrl(this, ID_SELECTFILES_TR,
    selectedPath,
    wxDefaultPosition, wxSize(240, 300),
    wxDIRCTRL_3D_INTERNAL | wxTR_LINES_AT_ROOT,
    "Music files|*.mp3;*.wav", 0);

  item2->Add( dirTr, 1, wxGROW | wxALL, 5 );

  item1->Add( item2, 1, wxGROW | wxALL, 5 );

  wxStaticBox *item7 = new wxStaticBox(this, -1,
    "Name Extraction Settings");
  wxStaticBoxSizer *item6 = new wxStaticBoxSizer( item7, wxVERTICAL );

  wxStaticText *item8 = new wxStaticText( this, -1,
    "Look for artist and title information", wxDefaultPosition,
    wxDefaultSize, 0 );
  item6->Add( item8, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxFlexGridSizer *item9 = new wxFlexGridSizer( 2, 0, 0 );

  wxStaticText *item10 = new wxStaticText( this, -1, "first in",
    wxDefaultPosition, wxDefaultSize, 0 );
  item9->Add( item10, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxString choices[] = { "ID3 tag", "file name" };
  look1DD = new wxChoice( this, ID_LOOK1_DD, wxDefaultPosition,
    wxSize(100,-1), 2, choices, 0 );
  look1DD->SetSelection(look1Val);
  item9->Add(look1DD, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item12 = new wxStaticText( this, -1, "then in",
    wxDefaultPosition, wxDefaultSize, 0 );
  item9->Add( item12, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  look2DD = new wxChoice( this, ID_LOOK2_DD, wxDefaultPosition,
    wxSize(100,-1), 2, choices, 0 );
  look2DD->SetSelection(look2Val);
  item9->Add( look2DD, 0, wxALIGN_CENTER|wxALL, 5 );

  item6->Add( item9, 0, wxALIGN_CENTER_VERTICAL|wxALL, 0 );

  item1->Add( item6, 0, wxALIGN_RIGHT|wxALIGN_CENTER_HORIZONTAL|wxALL, 5 );

  item0->Add( item1, 1, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxStaticLine *item15 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL);
  item0->Add(item15, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  wxBoxSizer *item14 = new wxBoxSizer( wxHORIZONTAL );

  okBut = new wxButton( this, wxID_OK, "OK",
    wxDefaultPosition, wxDefaultSize, 0 );
  okBut->SetDefault();
  item14->Add( okBut, 0, wxALIGN_BOTTOM|wxALIGN_CENTER_HORIZONTAL|wxALL, 5 );

  cancelBut = new wxButton( this, wxID_CANCEL, "Cancel",
    wxDefaultPosition, wxDefaultSize, 0 );
  item14->Add( cancelBut, 0, wxALIGN_BOTTOM|wxALIGN_CENTER_HORIZONTAL|wxALL, 5 );

  item0->Add( item14, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  SetAutoLayout( TRUE );
  SetSizer( item0 );
  item0->Fit( this );
  item0->SetSizeHints( this );

  wxSize size = GetSize();
  long width = cfg->Read("/AddFilesDialog/width", size.GetWidth());
  long height = cfg->Read("/AddFilesDialog/height", size.GetWidth());

  SetSize(width, height);

}
```

```
/*****************************************************************************

  beat.sql

  SQLite database definition file for the database used to store tracks and
  their tempo.

  Author: Erik Jälevik

*****************************************************************************/


-- Delete tables if they already exist

DROP TABLE tempo_map;
DROP TABLE format;
DROP TABLE genre;
DROP TABLE music_track;


-- Create tables anew

CREATE TABLE music_track (
  id        INTEGER PRIMARY KEY,
  artist    VARCHAR,
  title     VARCHAR,
  length    INTEGER, -- in seconds
  format    VARCHAR, -- there is some reason i've forgotten why this field is a
string
  source    VARCHAR,
  genre     VARCHAR, -- there is some reason i've forgotten why this field is a
string
  verified  INTEGER, -- boolean, 0 = false, 1 = true
  added_on  VARCHAR  -- format "YYYYMMDDHHMMSS"
);

CREATE INDEX artist_idx ON music_track(artist);
CREATE INDEX title_idx ON music_track(title);
CREATE INDEX format_idx ON music_track(format);
CREATE INDEX genre_idx ON music_track(genre);
CREATE INDEX filter_idx ON music_track(artist, title, format, genre);

CREATE TABLE genre (
  id        INTEGER PRIMARY KEY,
  genre     VARCHAR UNIQUE NOT NULL
);

CREATE TABLE format (
  id        INTEGER PRIMARY KEY,
  format    VARCHAR UNIQUE NOT NULL -- CD, file or vinyl etc
);

CREATE TABLE tempo_map (
  track_id  INTEGER,
  time      INTEGER, -- seconds since track start
  tempo     INTEGER NOT NULL,
  PRIMARY KEY(track_id, time)
);

CREATE INDEX time_idx ON tempo_map(time);
CREATE INDEX tempo_idx ON tempo_map(tempo);


-- Insert some genres

INSERT INTO genre VALUES (NULL, 'Pop'); -- NULL = auto ID
INSERT INTO genre VALUES (NULL, 'Indie');
INSERT INTO genre VALUES (NULL, 'Rock');
INSERT INTO genre VALUES (NULL, 'Metal');

INSERT INTO genre VALUES (NULL, 'Country');

INSERT INTO genre VALUES (NULL, 'Soul');
INSERT INTO genre VALUES (NULL, 'R&B');
INSERT INTO genre VALUES (NULL, 'Funk');
INSERT INTO genre VALUES (NULL, 'Disco');

INSERT INTO genre VALUES (NULL, 'Reggae');

INSERT INTO genre VALUES (NULL, 'Jazz');

INSERT INTO genre VALUES (NULL, 'Hip Hop');

INSERT INTO genre VALUES (NULL, 'House');
INSERT INTO genre VALUES (NULL, 'Techno');
INSERT INTO genre VALUES (NULL, 'Trance');
INSERT INTO genre VALUES (NULL, 'Drum & Bass');
INSERT INTO genre VALUES (NULL, 'Electro');
INSERT INTO genre VALUES (NULL, 'Electronica');

INSERT INTO genre VALUES (NULL, 'Industrial');


-- Insert formats

INSERT INTO format VALUES (NULL, 'CD');
INSERT INTO format VALUES (NULL, 'MP3');
INSERT INTO format VALUES (NULL, 'WAV');
INSERT INTO format VALUES (NULL, 'Vinyl');
INSERT INTO format VALUES (NULL, 'MD');
INSERT INTO format VALUES (NULL, 'DAT');
INSERT INTO format VALUES (NULL, 'MC');
```

```
/*****************************************************************************

  BeatApp.h

  The wxApp main application class for the beat tracking app.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef BEATAPP_H
#define BEATAPP_H

#include "wx/app.h"

class BeatApp : public wxApp {

  public:

    /*************************************************************************
      'Main' method that launches application window.
    *************************************************************************/
    virtual bool OnInit();

};

DECLARE_APP(BeatApp)

#endif
```

```
/*****************************************************************************

  BeatApp.cpp

  The wxApp main application class for the beat-tracking app.

  Author: Erik Jälevik

*****************************************************************************/
#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "BeatApp.h"
#include "BeatFrame.h"
#include "Globals.h"

#include <wx/confbase.h>
#include <wx/fileconf.h>

using namespace std;

IMPLEMENT_APP(BeatApp)

/*****************************************************************************
  OnInit
*****************************************************************************/
bool BeatApp::OnInit() {

  // Set up config file in same directory as application. Only works if
  // whole path is always used to launch program, therefore restricted to
  // Windows.
  wxString cfgFilename = "";
  #ifdef _WINDOWS
    cfgFilename << ::wxPathOnly(argv[0]) << "\\";
  #endif
  cfgFilename << "prefs.cfg";

  // Create config object and set it as default
  wxFileConfig* cfg = new wxFileConfig("", "",
    cfgFilename, wxEmptyString, wxCONFIG_USE_LOCAL_FILE);
  wxConfigBase::Set(cfg);
  wxConfigBase::DontCreateOnDemand();

  // Create main frame
  try {
    BeatFrame* frame = new BeatFrame("Beatrak");
    frame->Show(TRUE);
    SetTopWindow(frame);
  }
  catch (exception& e) {
    wxString msg = "Something went horribly wrong when trying to start the "
      "application.\nHere's what:\n\n";
    msg += e.what();
    ::wxMessageBox(msg, "Error", wxOK | wxICON_ERROR | wxCENTRE);
    return false;
  }

  return true;

}
```

```
/**********************************************************************

  BeatFrame.h

  Main application window inherited from wxFrame. Contains event table and most
  event handling methods.

  Author: Erik Jälevik

**********************************************************************/

#ifndef BEATFRAME_H
#define BEATFRAME_H

#include "BeatTracker.h"
#include "FilterPanel.h"
#include "CollectionListCtrl.h"
#include "MusicCollection.h"
#include "AddDialog.h"

#include <memory> // auto_ptr

class BeatFrame : public wxFrame {

  public:

    /*******************************************************************
      Constructor sets tracker pointer to 0, it's initialised when a new beat
      tracking operation is started. Parameter is the window caption.
    *******************************************************************/
    BeatFrame(const wxString&);

    /*******************************************************************
      Destructor
    *******************************************************************/
    virtual ~BeatFrame();

    /*******************************************************************
      Exporting database contents to a CSV file.
    *******************************************************************/
    void OnExport(wxCommandEvent&);

    /*******************************************************************
      Quit program.
    *******************************************************************/
    void OnExit(wxCommandEvent&);

    /*******************************************************************
      Add Files dialog.
    *******************************************************************/
    void OnAddFiles(wxCommandEvent&);

    /*******************************************************************
      Add CD dialog.
    *******************************************************************/
    void OnAddCD(wxCommandEvent&);

    /*******************************************************************
      Add Line-In dialog.
    *******************************************************************/
    void OnAddLineIn(wxCommandEvent&);

    /*******************************************************************
      Edit current track.
    *******************************************************************/
    void OnEdit(wxCommandEvent&);

    /*******************************************************************
      Delete current track.
    *******************************************************************/
    void OnDelete(wxCommandEvent&);

    /*******************************************************************
      Find a track.
    *******************************************************************/
    void OnFind(wxCommandEvent&);

    /*******************************************************************
      Launch tapper.
    *******************************************************************/
    void OnTapper(wxCommandEvent&);

    /*******************************************************************
      Options window.
    *******************************************************************/
    void OnOptions(wxCommandEvent&);

    /*******************************************************************
      Help.
    *******************************************************************/
    void OnHelp(wxCommandEvent&);

    /*******************************************************************
      About window.
    *******************************************************************/
    void OnAbout(wxCommandEvent&);


  private:

    /*******************************************************************
      Autogenerated method for building dialog
    *******************************************************************/
    void BuildMe();

    /*******************************************************************
      Launches the Add dialog and starts beat tracking.
    *******************************************************************/
    void BeatTrack(AddDialog&, std::auto_ptr<InStream>);


    /*******************************************************************
      Variables
    *******************************************************************/

    // Controls
    wxMenu* fileMenu;
    wxMenu* editMenu;
    wxMenu* toolsMenu;
    wxMenu* helpMenu;
    wxToolBar* toolBar;

    FilterPanel* filterPan;
    CollectionListCtrl* collList;

    // Model objects
    MusicCollection* coll;
    std::auto_ptr<BeatTracker> tracker;

    enum {
      ID_OPEN_MI = wxID_HIGHEST + 1,
      ID_EXPORT_MI,
      ID_EXIT_MI,
      ID_ADDFILES_MI,
      ID_ADDCD_MI,
      ID_ADDLINEIN_MI,
```

```
    ID_EDIT_MI,
    ID_DELETE_MI,
    ID_FIND_MI,
    ID_TAPPER_MI,
    ID_OPTIONS_MI,
    ID_HELP_MI,
    ID_ABOUT_MI,
    ID_COLLECTION_LC,
    ID_FILTER_PAN
  };

  DECLARE_EVENT_TABLE()

};

#endif
```

```
/***************************************************************************

  BeatFrame.cpp

  Main application window inherited from wxFrame. Contains event table and
  most event handling methods.

  Author: Erik Jälevik

***************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "BeatFrame.h"
#include "ScheirerTracker.h"
#include "FileInStream.h"
#include "CDInStream.h"
#include "AddFilesDialog.h"
#include "AddCDDialog.h"
#include "ProgressDialog.h"
#include "TapperDialog.h"
#include "OptionsDialog.h"
#include "AboutDialog.h"
#include "Icons.h"
#include "Globals.h"

#include <wx/confbase.h>

using namespace std;

/***************************************************************************
  Constructor
***************************************************************************/
BeatFrame::BeatFrame(const wxString& title) :
  wxFrame((wxFrame*)NULL, -1, title) {

  // Read database filename from config
  wxConfigBase* cfg = wxConfigBase::Get();
  wxString db = cfg->Read("/Options/Database/file", "beat.db");

  // Need to write it out in case this is the first time
  cfg->Write("/Options/Database/file", db);

  // Send filename to MusicCollection for opening the db
  coll = MusicCollection::Instance();
  coll->Open(db.c_str());

  BuildMe();

}

/***************************************************************************
  Destructor
***************************************************************************/
BeatFrame::~BeatFrame() {

  // Don't save config values if frame is iconized because then pos values
  // will be -32000, -32000 on next start and window will be hidden
  if (!IsIconized()) {
    wxConfigBase* cfg = wxConfigBase::Get();
```

```
    wxSize size = GetSize();
    cfg->Write("/BeatFrame/width", size.GetWidth());
    cfg->Write("/BeatFrame/height", size.GetHeight());

    wxPoint pos = GetPosition();
    cfg->Write("/BeatFrame/xpos", pos.x);
    cfg->Write("/BeatFrame/ypos", pos.y);

    bool mxm = IsMaximized();
    cfg->Write("/BeatFrame/maximized", mxm);
  }

  LOG("BeatFrame dtor end.\n");

}

/*********************************************************************************
  Event table
*********************************************************************************/
BEGIN_EVENT_TABLE(BeatFrame, wxFrame)
  EVT_MENU(ID_EXPORT_MI, BeatFrame::OnExport)
  EVT_MENU(ID_EXIT_MI, BeatFrame::OnExit)
  EVT_MENU(ID_ADDFILES_MI, BeatFrame::OnAddFiles)
  EVT_MENU(ID_ADDCD_MI, BeatFrame::OnAddCD)
  EVT_MENU(ID_ADDLINEIN_MI, BeatFrame::OnAddLineIn)
  EVT_MENU(ID_EDIT_MI, BeatFrame::OnEdit)
  EVT_MENU(ID_DELETE_MI, BeatFrame::OnDelete)
  EVT_MENU(ID_FIND_MI, BeatFrame::OnFind)
  EVT_MENU(ID_TAPPER_MI, BeatFrame::OnTapper)
  EVT_MENU(ID_OPTIONS_MI, BeatFrame::OnOptions)
  EVT_MENU(ID_HELP_MI, BeatFrame::OnHelp)
  EVT_MENU(ID_ABOUT_MI, BeatFrame::OnAbout)
END_EVENT_TABLE()

/*********************************************************************************
  OnExport
*********************************************************************************/
void BeatFrame::OnExport(wxCommandEvent& event) {

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString dbName = cfg->Read("/Options/Database/file");

  // Remove extension
  dbName = dbName.BeforeLast('.').IsEmpty() ? dbName : dbName.BeforeLast('.');
  dbName << ".csv";

  wxString file = ::wxFileSelector("Export to CSV", "", dbName, ".csv",
    "CSV file (*.csv)|*.csv", wxSAVE | wxOVERWRITE_PROMPT);

  wxBusyCursor busy;

  if (!file.IsEmpty()) {
    try {
      coll->ExportCSV(file.c_str());
    }
    catch (MusicCollection::DatabaseException& e) {
      ::wxMessageBox(e.what(), "Database Error", wxOK|wxICON_ERROR|wxCENTRE);
    }
  }

}

/*********************************************************************************
  OnExit
*********************************************************************************/
void BeatFrame::OnExit(wxCommandEvent& event) {
  Close(FALSE);

}

/*********************************************************************************
  OnAddFiles
*********************************************************************************/
void BeatFrame::OnAddFiles(wxCommandEvent& event) {

  try {
    // OK to create modal dialogs on stack
    AddFilesDialog dlg(this, -1);
    BeatTrack(dlg, auto_ptr<InStream>(new FileInStream()));
  }
  catch (exception& e) {
    ::wxMessageBox(e.what(), "Error", wxOK|wxICON_ERROR|wxCENTRE);
  }

}

/*********************************************************************************
  OnAddCD
*********************************************************************************/
void BeatFrame::OnAddCD(wxCommandEvent& event) {

  try {
    // OK to create modal dialogs on stack
    AddCDDialog dlg(this, -1);
    BeatTrack(dlg, auto_ptr<InStream>(new CDInStream()));
  }
  catch (exception& e) {
    ::wxMessageBox(e.what(), "Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*********************************************************************************
  OnAddLineIn
*********************************************************************************/
void BeatFrame::OnAddLineIn(wxCommandEvent& event) {

  ::wxMessageBox("Add Line-in not yet implemented.");

}

/*********************************************************************************
  OnEdit
*********************************************************************************/
void BeatFrame::OnEdit(wxCommandEvent& event) {

  // Try-catch block in CollectionListCtrl
  collList->EditSelectedTracks();

}

/*********************************************************************************
  OnDelete
*********************************************************************************/
void BeatFrame::OnDelete(wxCommandEvent& event) {

  // Try-catch block in CollectionListCtrl
  collList->DeleteSelectedTracks();

}

/*********************************************************************************
  OnFind
*********************************************************************************/
```

```
void BeatFrame::OnFind(wxCommandEvent& event) {

  // No need for try catch, function only includes wx code
  collList->Find();

}

/*****************************************************************************
  OnTapper
*****************************************************************************/
void BeatFrame::OnTapper(wxCommandEvent& event) {

  try {
    // Tapper should not be modal
    TapperDialog* tapper = new TapperDialog(this, -1);
    tapper->Show(TRUE);
  }
  catch (exception& e) {
    wxString msg = "Could not open Tapper window.\n\n";
    #ifdef _DEBUG
      msg << e.what();
    #endif
    ::wxMessageBox(msg, "Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  OnOptions
*****************************************************************************/
void BeatFrame::OnOptions(wxCommandEvent& event) {

  try {
    OptionsDialog options(this, -1);
    options.ShowModal();
  }
  catch (exception& e) {
    wxString msg = "Could not open Options window.\n\n";
    #ifdef _DEBUG
      msg << e.what();
    #endif
    ::wxMessageBox(msg, "Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  OnHelp
*****************************************************************************/
void BeatFrame::OnHelp(wxCommandEvent& event) {

  ::wxMessageBox("Help not yet implemented.");

}

/*****************************************************************************
  OnAbout
*****************************************************************************/
void BeatFrame::OnAbout(wxCommandEvent& event) {

  try {
    AboutDialog dlg(this, -1);
    dlg.ShowModal();
  }
  catch (exception& e) {
    wxString msg = "Could not open About window.\n\n";
    #ifdef _DEBUG
      msg << e.what();
    #endif
    ::wxMessageBox(msg, "Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  BeatTrack
*****************************************************************************/
void BeatFrame::BeatTrack(AddDialog& dlg, auto_ptr<InStream> stream) {

  // Show passed dialog
  if (dlg.ShowModal() == wxID_OK) {

    ::wxSafeYield();
    wxBusyCursor busy;

    // Get selected tracks from dialog
    auto_ptr<vector<MusicTrack> > selectedTracks =
      dlg.GetSelectedTracks();

    if (!selectedTracks->empty()) {

      // Get tracker options from config
      wxConfigBase* cfg = wxConfigBase::Get();
      long algorithm = cfg->Read("/Options/BeatTracking/algorithm", 0L);
      long accuracy = cfg->Read("/Options/BeatTracking/accuracy", 3L);
      bool trackChanges;
      cfg->Read("/Options/BeatTracking/trackChanges", &trackChanges, true);

      // Configure tracker. Tracker should depend on algorithm parameter but
      // only Scheirer implemented yet.
      tracker.reset(new ScheirerTracker(stream, accuracy, trackChanges,
        false));

      // Start tracking from progress dialog
      ProgressDialog progDlg(this, *tracker, selectedTracks);
      progDlg.Run();

    }

  }

}

/*****************************************************************************
  BuildMe
*****************************************************************************/
void BeatFrame::BuildMe() {

  // Create application icons
  wxIconBundle icons;
  wxIcon icon;
  icon.CopyFromBitmap(Icons::Speaker16());
  icons.AddIcon(icon);
  icon.CopyFromBitmap(Icons::Speaker32());
  icons.AddIcon(icon);
  SetIcons(icons);

  // Make file menu
  fileMenu = new wxMenu();
  fileMenu->Append(ID_EXPORT_MI, "&Export...\tCtrl-E");
  fileMenu->AppendSeparator();
  fileMenu->Append(ID_EXIT_MI, "E&xit\tCtrl-Q");

  // Make edit menu
  editMenu = new wxMenu();
  editMenu->Append(ID_ADDFILES_MI, "Add F&iles...\tCtrl-I");
```

```
editMenu->Append(ID_ADDCD_MI, "Add &CD...\tCtrl-D");
editMenu->Append(ID_ADDLINEIN_MI, "Add from &Line-in...\tCtrl-L");
editMenu->AppendSeparator();

// Adding a space makes these work as expected in the FilterPanel.
// If not they get used as accelerators and the controls in the FilterPanel
// never gets them.
editMenu->Append(ID_EDIT_MI, "&Edit Track...\tEnter ");
editMenu->Append(ID_DELETE_MI, "&Delete Track\tDel ");
editMenu->AppendSeparator();
editMenu->Append(ID_FIND_MI, "&Find...\tCtrl-F");

// Make tools menu
toolsMenu = new wxMenu();
toolsMenu->Append(ID_TAPPER_MI, "&Tapper...\tCtrl-T");
toolsMenu->AppendSeparator();
toolsMenu->Append(ID_OPTIONS_MI, "&Options...\tCtrl-O");

// Make help menu
helpMenu = new wxMenu();
helpMenu->Append(ID_HELP_MI, "&Help...\tF1");
helpMenu->AppendSeparator();
helpMenu->Append(ID_ABOUT_MI, "&About...");

// Make a menubar and append menus to it
wxMenuBar* menuBar = new wxMenuBar();
menuBar->Append(fileMenu, "&File");
menuBar->Append(editMenu, "&Edit");
menuBar->Append(toolsMenu, "&Tools");
menuBar->Append(helpMenu, "&Help");

// Associate the menu bar with the frame
SetMenuBar(menuBar);

// Make a toolbar
toolBar = CreateToolBar();
toolBar->SetToolBitmapSize(wxSize(16, 16));
toolBar->AddTool(ID_ADDFILES_MI, "Add Files", Icons::AddFile(),
  "Add audio file(s)");
toolBar->AddTool(ID_ADDCD_MI, "Add CD", Icons::AddCD(),
  "Add tracks from CD");
toolBar->AddTool(ID_ADDLINEIN_MI, "Add Line-In", Icons::AddLineIn(),
  "Add track from soundcard line-in");
toolBar->AddSeparator();
toolBar->AddTool(ID_EDIT_MI, "Edit", Icons::Edit(),
  "Edit track");
toolBar->AddTool(ID_DELETE_MI, "Delete", Icons::Delete(),
  "Delete track");
toolBar->AddSeparator();
toolBar->AddTool(ID_TAPPER_MI, "Tapper", Icons::Tap(),
  "Open Tapper");

toolBar->Realize();

wxStatusBar* statusBar = CreateStatusBar();

filterPan = new FilterPanel(this, ID_FILTER_PAN);
collList = new CollectionListCtrl(this, ID_COLLECTION_LC, *statusBar);

wxBoxSizer* backSiz = new wxBoxSizer(wxVERTICAL);
backSiz->Add(filterPan, 0, wxEXPAND);
backSiz->Add(collList, 1, wxEXPAND);

SetAutoLayout(TRUE);
SetSizer(backSiz);

// Read config for window size and pos
wxConfigBase* cfg = wxConfigBase::Get();
  bool maximized;
  cfg->Read("/BeatFrame/maximized", &maximized, false);
  if (maximized) {
    Maximize();
  }
  else {
    long width = cfg->Read("/BeatFrame/width", 650L);
    long height = cfg->Read("/BeatFrame/height", 440L);
    long xpos = cfg->Read("/BeatFrame/xpos", 20L);
    long ypos = cfg->Read("/BeatFrame/ypos", 20L);
    SetSize(width, height);
    Move(xpos, ypos);
  }

}
```

```
/*****************************************************************************

  BeatTracker.h

  Base class for beat trackers. All this class implements is the InStream
  used for getting audio from a streaming source.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef BEATTRACKER_H
#define BEATTRACKER_H

#include "Observable.h"
#include "InStream.h"
#include "Globals.h"

#include <string>
#include <map>    // returned from TrackTempo
#include <memory> // auto_ptr

class BeatTracker : public Observable {

  public:

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~BeatTracker() { }

    /*************************************************************************
      Sets the InStream to use.
    *************************************************************************/
    virtual void SetInStream(std::auto_ptr<InStream> stream) { in = stream; }

    /*************************************************************************
      Returns a reference to the currently used InStream.
    *************************************************************************/
    virtual InStream& GetInStream() { return *in; }

    /*************************************************************************
      Sets the accuracy to use. The argument should be a value between 1
      and 5. A low accuracy means faster operation but less accurate beat
      tracking.
    *************************************************************************/
    virtual void SetAccuracy(int) = 0; // pure virtual

    /*************************************************************************
      Set whether to track tempo changes or not. If false, the track's overall
      tempo will be returned by TrackTempo.
    *************************************************************************/
    virtual void SetTrackChanges(bool t) { trackChanges = t; }

    /*************************************************************************
      Tracks the tempo of the named file.
    *************************************************************************/
    virtual std::auto_ptr<IntShortMap> TrackTempo(const std::string&) = 0;

    /*************************************************************************
      Returns a value between 0 and 1 indicating how far the tracking of
      the current file has got.
    *************************************************************************/
    virtual float GetProgress() = 0;

    /*************************************************************************
      Call to stop an ongoing tracking operation.
    *************************************************************************/
    virtual void Stop() = 0;

    /*************************************************************************
      Call this from scheduler after each Notify to check whether tracker has
      been stopped.
    *************************************************************************/
    virtual bool WasStopped() = 0;


  protected:

    /*************************************************************************
      Constructor takes the InStream to use for audio input.
    *************************************************************************/
    BeatTracker::BeatTracker(std::auto_ptr<InStream> stream) :
      in(stream), // copying of auto_ptr makes stream a null pointer
      audioRate(44100),
      stopped(false),
      trackChanges(true) { in->SetRate(audioRate); }


    /*************************************************************************
      Variables
    *************************************************************************/
    std::auto_ptr<InStream> in; // audio in stream
    uint audioRate; // sample frequency of audio input (Hz)
    bool trackChanges; // should tempo changes be tracked?
    bool stopped; // was tracker stopped during last Notify?

};

#endif
```

```
/*****************************************************************************

  CDInStream.h

  Subclass of InStream dealing with input from CD. This file is using a
  library called AKRip which is Windows-only so this class can not be used
  when compiling under Linux.

  Open must be called with a valid parameter before any of the other methods
  can be called.

  This class requires a working ASPI layer to be installed on the computer.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef CDINSTREAM_H
#define CDINSTREAM_H

#include "InStream.h"
#include "Globals.h"

#include "akrip32.h"

#include <vector> // returned by various methods
#include <string>

class CDInStream : public InStream {

  public:

    /*************************************************************************
      Constructor discovers CD drives and allocates buffer memory.
    *************************************************************************/
    CDInStream();

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~CDInStream();

    /*************************************************************************
      Opens the named device and track. Strings passed in should be 3
      characters long and of the format "dtt" where d is the device number
      (starting from 0) and tt is the track number (starting from 1).
    *************************************************************************/
    virtual void Open(const std::string& file);

    /*************************************************************************
      Closes the stream.
    *************************************************************************/
    virtual void Close();

    /*************************************************************************
      Returns true if there is more data to read in the stream.
    *************************************************************************/
    virtual bool HasMore() { return !finished; }

    /*************************************************************************
      Read one sample. If the file is a stereo file, the average of the two
      channels is returned.
    *************************************************************************/
    virtual double Read();

    /*************************************************************************
      Returns length of stream in seconds.
    *************************************************************************/
    virtual int GetLength() { return trackLen / 75; }

    /*************************************************************************
      Returns a number between 0 and 1 indicating how much of the stream
      has been read.
    *************************************************************************/
    virtual float GetProgress();

    /*************************************************************************
      Returns string descriptions of CD drives in the system.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<std::string> > GetDrives();

    /*************************************************************************
      Returns vector of the lengths in seconds of all tracks on the CD. The
      0-based parameter specifies which drive to query.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<uint> > GetTrackLengths(int);


  private:

    // "Mem" used for "member" in the following functions to distinguish them
    // from AKrip functions of the same names

    /*************************************************************************
      Get the CD drive list.
    *************************************************************************/
    CDLIST GetCDListMem();

    /*************************************************************************
      Get the CD handle for specified device.
    *************************************************************************/
    HCDROM GetCDHandleMem(int);

    /*************************************************************************
      Get CD table of contents.
    *************************************************************************/
    TOC GetTOCMem(const HCDROM&);

    /*************************************************************************
      Convert 4 individual bytes to a DWORD.
    *************************************************************************/
    DWORD Bytes2DWord(BYTE*);

    /*************************************************************************
      Allocate LPTRACKBUF.
    *************************************************************************/
    LPTRACKBUF NewTrackBuf(DWORD);

    /*************************************************************************
      Fill up cache to bufSize limit.
    *************************************************************************/
    void FillBuffer();

    /*************************************************************************
      Function for formatting error message and throwing exception.
    *************************************************************************/
    void Error(const char*);


    /*************************************************************************
      Variables
    *************************************************************************/

    static const uint bufSize = 100; // buffer size in kB
                                     // max 100 if using Nero ASPI
```

```
    static const int retries = 3; // number of read retries on CD read failure

    CDLIST cdList;       // list of CDs present in system
    HCDROM handle;       // handle to currently open CD drive
    LPTRACKBUF trackBuf; // structure that holds read data

    DWORD startAddr; // current LBA frame start address for current track
    DWORD endAddr;   // LBA frame end address for current track
    uint trackLen;   // length in frames of currently open track

    float readStep; // sample steps to advance in each call to Read
    float readCtr;  // read counter
    uint readIdx;   // fixed point index into output buffer

    int openDevice; // currently open device number
    int openTrack;  // currently open track number

    bool allRead;  // has all data of a track been read into cache
    bool finished; // is stream finished

};

#endif
```

```
/*****************************************************************************

  CDInStream.cpp

  Subclass of InStream dealing with input from CD. This file is using a
  library called AKRip which is Windows-only so this class can not be used
  when compiling under Linux. Most of it is written in non-OO C
  since that's dictated by AKRip.

  Open must be called with a valid parameter before any of the other methods
  can be called.

  This class requires a working ASPI layer to be installed on the computer.

  A CD has 75 frames per second.

  Author: Erik Jälevik

*****************************************************************************/

#include "CDInStream.h"
#include "Globals.h"

#include <sstream>      // ostringstream
#include <cmath>        // floorf

using namespace std;

/*****************************************************************************
  Constructor
*****************************************************************************/
CDInStream::CDInStream() :
  allRead(true),
  finished(true),
  startAddr(0),
  endAddr(0),
  trackBuf(0),
  readStep(1.0),
  readCtr(0.0),
  readIdx(0),
  openDevice(-1),
  openTrack(-1),
  trackLen(0) {

  // Get list of cd drives.
  cdList = GetCDListMem();

  if (cdList.num == 0) {
    Error("No CD-ROM drives found.");
  }

  // Allocate memory for the audio buffer
  trackBuf = NewTrackBuf((bufSize * 1000) / 2352); // 2352 bytes per frame
  if (!trackBuf) {
    Error("Could not allocate memory to read CD audio into.");
  }

  LOG("CD buffer no of frames: " << trackBuf->numFrames << endl);

}

/*****************************************************************************
  Destructor
*****************************************************************************/
CDInStream::~CDInStream() {

  CloseCDHandle(handle);
  free(trackBuf);
```

```
}

/*****************************************************************************
  Open
*****************************************************************************/
void CDInStream::Open(const string& file) {

  if (file.length() != 3) {
    throw logic_error(
      "Argument string to CDInStream::Open needs to be 3 characters long.");
  }

  // Parse parameter string
  int device = atoi(file.substr(0, 1).c_str());
  int track = atoi(file.substr(1).c_str());

  LOG("CD device requested: " << device << endl);
  LOG("CD track requested: " << track << endl);

  // Check that device param is not too high
  if (device > cdList.num - 1) {
    Error("Specified CD-ROM drive does not seem to exist.");
  }

  // Get handle
  handle = GetCDHandleMem(device);
  if (!handle) {
    ostringstream os;
    os << "Could not get handle to CD-ROM drive " << device;
    Error(os.str().c_str());
  }

  // Get table of contents
  TOC toc = GetTOCMem(handle);

  LOG("CD no of tracks: " << (int)(toc.lastTrack) << endl);

  // Check that track param is not too high
  if (track > toc.lastTrack) {
    Error("Specified track number does not seem to exist on CD.");
  }

  // Use toc to determine start and end addresses for reading
  startAddr = Bytes2DWord(toc.tracks[track - 1].addr);
  endAddr = Bytes2DWord(toc.tracks[track].addr);
  trackLen = static_cast<int>(endAddr - startAddr);

  LOG("CD track start addr: " << startAddr << endl);
  LOG("CD track end addr: " << endAddr << endl);

  // Reset trackBuf size
  trackBuf->numFrames = ((bufSize * 1000) / 2352);
  trackBuf->len = 0;

  // Calculate read step
  readStep = 44100.0f / readRate;
  readCtr = 0.0;
  readIdx = 0;

  LOG("CD read step: " << readStep << endl);

  openDevice = device;
  openTrack = track;
  finished = false;
  allRead = false;

}

/*****************************************************************************
  Close
*****************************************************************************/
void CDInStream::Close() {

  CloseCDHandle(handle);
  openDevice = -1;
  openTrack = -1;
  finished = true;
  allRead = true;

}

/*****************************************************************************
  Read
*****************************************************************************/
double CDInStream::Read() {

  // 4 bytes per output sample in buffer
  uint idx = readIdx * 4 + trackBuf->startOffset;

  // Check if buffer is empty.
  // Using while guards against the situation when FillBuffer returns 0 bytes
  // which can happen at the end of a file.
  while (idx >= trackBuf->len) {
    if (!allRead) {
      FillBuffer();
      idx = readIdx * 4 + trackBuf->startOffset;
    }
    else {
      finished = true;
      CloseCDHandle(handle);
      return 0.0;
    }
  }

  // Get bytes out of buffer and transform into 16-bit left and right values.
  // Bytes are big endian so need to swap bytes.
  short left = trackBuf->buf[idx + 1];
  left = (left << 8) + trackBuf->buf[idx];
  short right = trackBuf->buf[idx + 3];
  right = (right << 8) + trackBuf->buf[idx + 2];

  // Move buffer offset ahead one step
  readCtr += readStep;
  readIdx = static_cast<uint>(readCtr);

  // Average channels and scale to range -1 to +1
  double val = ((left + right) / 2) / 32768.0;

  return val;

}

/*****************************************************************************
  GetProgress
*****************************************************************************/
float CDInStream::GetProgress() {

  int pos = trackLen -
    (endAddr - trackBuf->startFrame) +
    (static_cast<float>(readIdx * 4) / trackBuf->len) * trackBuf->numFrames;

  //LOG("trackLen: " << (trackLen) <<
  //    ", left incl buf: " << (endAddr - trackBuf->startFrame) <<
  //    ", readIdx*4: " << readIdx * 4 <<
  //    ", bufLen: " << trackBuf->len <<
```

```
  //    ", ratio: " << (static_cast<float>(readIdx * 4) / trackBuf->len) <<
  //    ", pos: " << pos <<
  //    ", progr: " << static_cast<float>(pos) / (trackLen) << endl);

  return static_cast<float>(pos) / trackLen;

}

/*****************************************************************************
  GetDrives
*****************************************************************************/
auto_ptr<vector<string> > CDInStream::GetDrives() {

  // Get info from cd list and put it formatted into vector
  auto_ptr<vector<string> > v(new vector<string>);
  for(int i = 0; i < cdList.num; ++i) {
    stringstream s;
    s << (i + 1) << ": ";
    s << cdList.cd[i].info.vendor << " ";
    s << cdList.cd[i].info.prodId;
    v->push_back(s.str());
  }

  return v;

}

/*****************************************************************************
  GetTrackLengths
*****************************************************************************/
auto_ptr<vector<uint> > CDInStream::GetTrackLengths(int device) {

  // Check that device param is not too high
  if (device > cdList.num - 1) {
    Error("Specified CD-ROM drive does not seem to exist.");
  }

  // Get handle
  handle = GetCDHandleMem(device);
  if (!handle) {
    ostringstream os;
    os << "Could not get handle to CD-ROM drive " << device;
    Error(os.str().c_str());
  }

  // Get table of contents
  TOC toc = GetTOCMem(handle);

  auto_ptr<vector<uint> > v(new vector<uint>);

  for (int i = 0; i < toc.lastTrack; ++i) {
    DWORD start = Bytes2DWord(toc.tracks[i].addr);
    DWORD end = Bytes2DWord(toc.tracks[i + 1].addr);
    uint len = static_cast<int>((end - start) / 75);
    v->push_back(len);
  }

  CloseCDHandle(handle);

  return v;

}

/*****************************************************************************
  GetCDList
*****************************************************************************/
CDLIST CDInStream::GetCDListMem() {

  CDLIST cdl;
  memset(&cdl, 0, sizeof(cdl));
  cdl.max = MAXCDLIST;
  GetCDList(&cdl);
  return cdl;

}

/*****************************************************************************
  GetCDHandle
*****************************************************************************/
HCDROM CDInStream::GetCDHandleMem(int dev) {

  GETCDHAND gcd;
  memset(&gcd, 0, sizeof(gcd));
  gcd.size = sizeof(gcd);
  gcd.ver = 1;
  gcd.ha = cdList.cd[dev].ha;
  gcd.tgt = cdList.cd[dev].tgt;
  gcd.lun = cdList.cd[dev].lun;
  gcd.readType = CDR_ANY;
  gcd.numJitter = 1;
  gcd.numOverlap = 3;

  HCDROM cdh = GetCDHandle(&gcd);
  return cdh;

}

/*****************************************************************************
  GetTOC
*****************************************************************************/
TOC CDInStream::GetTOCMem(const HCDROM& cdh) {

  // Make sure toc is returned in LBA format
  ModifyCDParms(cdh, CDP_MSF, FALSE);

  TOC toc;
  memset(&toc, 0, sizeof(toc));
  if (ReadTOC(cdh, &toc) != SS_COMP) {
    Error("Could not access CD to get table of contents.");
  }
  return toc;

}

/*****************************************************************************
  Bytes2DWord
*****************************************************************************/
DWORD CDInStream::Bytes2DWord(BYTE* addr) {

  DWORD val = (DWORD)addr[0];
  val = (val << 8) + (DWORD)addr[1];
  val = (val << 8) + (DWORD)addr[2];
  val = (val << 8) + (DWORD)addr[3];

  return val;

}

/*****************************************************************************
  NewTrackBuf
*****************************************************************************/
LPTRACKBUF CDInStream::NewTrackBuf(DWORD nFrames) {

  LPTRACKBUF t; // LPTRACKBUF is a pointer

  // Work out bytes needed for frames + struct
```

```
  int bytesToAlloc = (((int)nFrames) * 2352) + TRACKBUFEXTRA;
  t = (LPTRACKBUF)malloc(bytesToAlloc);

  if (!t) {
    return NULL;
  }

  t->startFrame = 0;
  t->numFrames = nFrames;
  t->maxLen = nFrames * 2352;
  t->len = 0;
  t->status = 0;
  t->startOffset = 0;

  return t;

}

/*****************************************************************************
  FillBuffer
*****************************************************************************/
void CDInStream::FillBuffer() {

  uint length = endAddr - startAddr;

  // If the remainder of the song is shorter than the buffer, amend numFrames
  if (length < trackBuf->numFrames) {
    trackBuf->numFrames = length;
  }

  // Try reading until run out of retries or successful
  for (int r = retries; r > 0; r--) {

    trackBuf->startFrame = startAddr;
    trackBuf->startOffset = 0;
    trackBuf->len = 0;

    DWORD status = ReadCDAudioLBA(handle, trackBuf);

    if (status == SS_COMP) { // if OK

      // Increase startAddr to point to next chunk to cache
      startAddr += trackBuf->numFrames;
      if (startAddr >= endAddr ) {
        allRead = true;
      }

      // Reset readCtr preserving decimals
      readCtr = readCtr - floorf(readCtr);
      readIdx = 0;

      return;

    }

    LOG("Read retry occurred on addr " <<
      static_cast<int>(trackBuf->startFrame) << endl);

  }

  Error("Could not read audio from CD. Check that your ASPI layer is "
    "installed and working properly.");

}

/*****************************************************************************
  Error
*****************************************************************************/
void CDInStream::Error(const char* msg) {

  ostringstream tmp;
  tmp << msg;
  #ifdef _DEBUG
    tmp << "\n\nAKRip error: " << GetAspiLibError() << "\n";
    tmp << "ASPI error: " << static_cast<int>(GetAspiLibAspiError());
  #endif

  CloseCDHandle(handle);
  throw InStreamException(tmp.str());

}
```

```
/*****************************************************************************

  CollectionListCtrl.h

  GUI list control object for displaying the music collection.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef COLLECTIONLISTCTRL_H
#define COLLECTIONLISTCTRL_H

#include "MusicCollection.h"

#include <wx/listctrl.h>
#include <wx/fdrepdlg.h> // find replace dialog

#include <deque>  // sort history
#include <string>

class CollectionListCtrl : public wxListCtrl, public Observer {

  public:

    /*************************************************************************
      Parameters are the parent window, window ID and a reference to the main
      frame's status bar so that the collection list ctrl can update the
      status.
    *************************************************************************/
    CollectionListCtrl(wxWindow*, wxWindowID, wxStatusBar&);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~CollectionListCtrl();

    /*************************************************************************
      Called by the control when it needs data from the collection.
    *************************************************************************/
    wxString OnGetItemText(long, long) const;

    /*************************************************************************
      Called by the control when it needs attributes for an item.
    *************************************************************************/
    wxListItemAttr* OnGetItemAttr(long) const;

    /*************************************************************************
      This is not used but has to be overriden.
    *************************************************************************/
    int OnGetItemImage(long) const;

    /*************************************************************************
      Handler for when a track is activated by double-clicking or pressing
      Enter.
    *************************************************************************/
    void OnActivate(wxListEvent&);

    /*************************************************************************
      Handler for when a column header is clicked, i.e. sorting requested.
    *************************************************************************/
    void OnSort(wxListEvent&);

    /*************************************************************************
      Handler for detecting key presses.
    *************************************************************************/
    void OnChar(wxKeyEvent&);

    /*************************************************************************
      Handler for find dialog events.
    *************************************************************************/
    void OnFindDialogEvt(wxFindDialogEvent&);

    /*************************************************************************
      Launches track dialog(s).
    *************************************************************************/
    void EditSelectedTracks();

    /*************************************************************************
      Implements the delete track functionality.
    *************************************************************************/
    void DeleteSelectedTracks();

    /*************************************************************************
      Implements find dialog functionality.
    *************************************************************************/
    void Find();

    /*************************************************************************
      Inherited from Observer.
    *************************************************************************/
    void Update();

    // This enum defines order of columns
    enum COLUMN {
      ARTIST_COL = 0,
      TITLE_COL,
      TEMPO_COL,
      LENGTH_COL,
      FORMAT_COL,
      GENRE_COL,
      SOURCE_COL,
      ADDED_COL
    };


  private:

    /*************************************************************************
      Autogenerated method for building component.
    *************************************************************************/
    void BuildMe();

    /*************************************************************************
      Convenience method for putting together SQL sort clause to pass to
      MusicCollcetion.
    *************************************************************************/
    std::string GetSortString();


    /*************************************************************************
      Variables
    *************************************************************************/
    static const int nCols = 8;

    // Controls
    wxStatusBar* statusBar;

    wxFindReplaceDialog* findDlg;
    wxFindReplaceData findData; // find dialog's associated data object

    wxListItemAttr red; // used for colouring unverified records red

    // Model objects
    MusicCollection* coll;
    MusicTrack lastTrack;
```

```
    bool sortOrders[nCols]; // keeps track of  ASC (true) or DESC (false) order
    std::deque<std::string> sortHistory; // previously sorted on columns

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/*********************************************************************************

  CollectionListCtrl.cpp

  GUI list control object for displaying the music collection.

  Author: Erik Jälevik

*********************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include "wx/wx.h"
#endif

#include "CollectionListCtrl.h"
#include "MusicTrackDialog.h"
#include "Icons.h"
#include "Globals.h"

#include <wx/confbase.h>

#include <sstream> // ostringstream

using namespace std;

/*********************************************************************************
  Constructor
*********************************************************************************/
CollectionListCtrl::CollectionListCtrl(wxWindow* parent,
                                       wxWindowID id,
                                       wxStatusBar& sb) :
        wxListCtrl(parent, id, wxDefaultPosition, wxDefaultSize,
          wxLC_REPORT | wxLC_VIRTUAL | wxLC_HRULES),
        findDlg(0),
        statusBar(&sb) {

  BuildMe();

  // Initialise red attribute
  red.SetTextColour(Icons::Red());
  red.SetBackgroundColour(GetBackgroundColour());
  red.SetFont(GetFont());

  // Initialise all sort orders to false, this means that first click on a
  // header is going to change the sort to ascending order.
  for (int i = 0; i < nCols; ++i) {
    sortOrders[i] = false;
  }

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString sort1 = cfg->Read("/CollectionListCtrl/sort1", "tempo ASC");
  wxString sort2 = cfg->Read("/CollectionListCtrl/sort2", "artist ASC");
  wxString sort3 = cfg->Read("/CollectionListCtrl/sort3", "title ASC");
  sortHistory.push_back(sort1.c_str());
  sortHistory.push_back(sort2.c_str());
  sortHistory.push_back(sort3.c_str());

  coll = MusicCollection::Instance();
  coll->Sort(GetSortString());
  coll->Attach(this); // register self as an observer of collection

  Update(); // initialise display
```

```
}

/*****************************************************************************
  Destructor
*****************************************************************************/
CollectionListCtrl::~CollectionListCtrl() {

  coll->Detach(this);

  wxConfigBase* cfg = wxConfigBase::Get();
  cfg->Write("/CollectionListCtrl/artistWidth", GetColumnWidth(ARTIST_COL));
  cfg->Write("/CollectionListCtrl/titleWidth", GetColumnWidth(TITLE_COL));
  cfg->Write("/CollectionListCtrl/tempoWidth", GetColumnWidth(TEMPO_COL));
  cfg->Write("/CollectionListCtrl/lengthWidth", GetColumnWidth(LENGTH_COL));
  cfg->Write("/CollectionListCtrl/formatWidth", GetColumnWidth(FORMAT_COL));
  cfg->Write("/CollectionListCtrl/genreWidth", GetColumnWidth(GENRE_COL));
  cfg->Write("/CollectionListCtrl/sourceWidth", GetColumnWidth(SOURCE_COL));
  cfg->Write("/CollectionListCtrl/addedWidth", GetColumnWidth(ADDED_COL));

  cfg->Write("/CollectionListCtrl/sort1", sortHistory.at(0).c_str());
  cfg->Write("/CollectionListCtrl/sort2", sortHistory.at(1).c_str());
  cfg->Write("/CollectionListCtrl/sort3", sortHistory.at(2).c_str());

}

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(CollectionListCtrl, wxListCtrl)
  EVT_LIST_ITEM_ACTIVATED(-1, CollectionListCtrl::OnActivate)
  EVT_LIST_COL_CLICK(-1, CollectionListCtrl::OnSort)
  EVT_CHAR(CollectionListCtrl::OnChar)
  EVT_FIND(-1, CollectionListCtrl::OnFindDialogEvt)
  EVT_FIND_NEXT(-1, CollectionListCtrl::OnFindDialogEvt)
  EVT_FIND_CLOSE(-1, CollectionListCtrl::OnFindDialogEvt)
END_EVENT_TABLE()

/*****************************************************************************
  OnActivate
*****************************************************************************/
void CollectionListCtrl::OnActivate(wxListEvent& event) {

  EditSelectedTracks();

}

/*****************************************************************************
  OnSort
*****************************************************************************/
void CollectionListCtrl::OnSort(wxListEvent& event) {

  try {

    wxBusyCursor busy;
    int column = event.GetColumn();

    string order = sortOrders[column] ? "DESC" : "ASC";
    sortOrders[column] = !sortOrders[column]; // invert order for next time

    ostringstream ss;
    if      (column == ARTIST_COL) ss << "artist " << order;
    else if (column == TITLE_COL)  ss << "title " << order;
    else if (column == TEMPO_COL)  ss << "tempo " << order;
    else if (column == LENGTH_COL) ss << "length " << order;
    else if (column == FORMAT_COL) ss << "format " << order;
    else if (column == SOURCE_COL) ss << "source " << order;
    else if (column == GENRE_COL)  ss << "genre " << order;
    else if (column == ADDED_COL)  ss << "added_on " << order;
    else return;

    // TODO : think about what to do when it's the same column,
    // should probably just amend it rather than popping/pushing
    sortHistory.pop_back(); // remove oldest
    sortHistory.push_front(ss.str()); // insert newest

    coll->Sort(GetSortString());

  /* Use for when putting sort icons in
    wxListItem item;
    item.SetMask(wxLIST_MASK_IMAGE);
    item.SetImage(image);
    SetColumn(col, item);
  */

  }

  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  OnChar
*****************************************************************************/
void CollectionListCtrl::OnChar(wxKeyEvent& event) {

  if (event.GetKeyCode() == WXK_DELETE) {
    DeleteSelectedTracks();
  }

  event.Skip();

}

/*****************************************************************************
  OnFindDialogEvt
*****************************************************************************/
void CollectionListCtrl::OnFindDialogEvt(wxFindDialogEvent& event) {

  wxEventType type = event.GetEventType();

  if (type == wxEVT_COMMAND_FIND || type == wxEVT_COMMAND_FIND_NEXT) {
    wxString findWhat = event.GetFindString().c_str();
    int flags = event.GetFlags();
    bool down = flags & wxFR_DOWN;
    bool wholeWord = flags & wxFR_WHOLEWORD;
    bool matchCase = flags & wxFR_MATCHCASE;

    ::wxMessageBox("Find not implemented yet because wxListCtrl sucks.");

  }

  else if (type == wxEVT_COMMAND_FIND_CLOSE) {
    findDlg->Destroy();
  }

  else {
    LOG("Unknown find dialog event");
  }

}

/*****************************************************************************
  Find
```

```
/*********************************************************************************/
void CollectionListCtrl::Find() {

  findData = wxFindReplaceData();

  findDlg = new wxFindReplaceDialog(this, &findData, "Find");
  findDlg->Show(TRUE);

}

/*********************************************************************************
  OnGetItemText
*********************************************************************************/
wxString CollectionListCtrl::OnGetItemText(long row, long column) const {

  try {

    MusicTrack lastTrack = coll->GetTrack(row);

    if (column == ARTIST_COL)
      return wxString(lastTrack.GetArtist().c_str());

    else if (column == TITLE_COL)
      return wxString(lastTrack.GetTitle().c_str());

    else if (column == TEMPO_COL)
      return wxString(lastTrack.GetTempoRange().GetRangeString().c_str());

    else if (column == LENGTH_COL)
      return wxString(lastTrack.GetLengthString().c_str());

    else if (column == FORMAT_COL)
      return wxString(lastTrack.GetFormat().c_str());

    else if (column == SOURCE_COL)
      return wxString(lastTrack.GetSource().c_str());

    else if (column == GENRE_COL)
      return wxString(lastTrack.GetGenre().c_str());

    else if (column == ADDED_COL)
      return wxString(lastTrack.GetAddedOnString().c_str());

    else
      return wxString(""); // should never happen

  }

  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
    return "";
  }

}

/*********************************************************************************
  OnGetItemImage, not used but has to be overridden
*********************************************************************************/
int CollectionListCtrl::OnGetItemImage(long item) const {
  return -1;
}

/*********************************************************************************
  OnGetItemAttr
*********************************************************************************/
wxListItemAttr* CollectionListCtrl::OnGetItemAttr(long item) const {

  try {

    MusicTrack lastTrack = coll->GetTrack(item);
    if (!lastTrack.GetVerified()) {
      return const_cast<wxListItemAttr*>(&red);
    }
    else {
      return NULL;
    }
  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
    return NULL;
  }

}

/*********************************************************************************
  EditSelectedTracks
*********************************************************************************/
void CollectionListCtrl::EditSelectedTracks() {

  try {

    int maxTracks = 10;

    // Get first selection
    long index = GetNextItem(-1, wxLIST_NEXT_ALL,
      wxLIST_STATE_SELECTED);

    // Collect selected tracks in vector
    vector<MusicTrack> tracks;
    for (int i = 0; (index != -1) && (i < maxTracks); ++i) {
      MusicTrack trk = coll->GetTrack(index);
      tracks.push_back(trk);
      index = GetNextItem(index, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    }

    // Display warning if tracks reached max allowed and there's still more
    if (index != -1) {
      wxString msg = "More than ";
      msg << maxTracks << " tracks were selected. Only the first " <<
        maxTracks << " will be displayed.";
      ::wxMessageBox(msg, "Too many tracks",
        wxOK | wxICON_INFORMATION | wxCENTRE);
    }

    // Show dialogs
    if (tracks.size() == 1) {
      MusicTrackDialog* trkDialog =
        new MusicTrackDialog(this, -1, wxDefaultPosition, tracks[0]);
      trkDialog->Show(true);
    }
    else {
      for (size_t j = 0; j < tracks.size(); ++j) {
        MusicTrackDialog* trkDialog = new MusicTrackDialog(this, -1,
          wxPoint(j*25+20, j*25+20), tracks[j]);
        trkDialog->Show(true);
      }
    }

  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }
  // In case the MusicTrackDialog throws something exotic
  catch (exception& e) {
    ::wxMessageBox(e.what(), "Error", wxOK | wxICON_ERROR | wxCENTRE);
  }
```

```
}

/////////////////////////////////////////////////////////////////////////////
  DeleteSelectedTracks
/////////////////////////////////////////////////////////////////////////////*/
void CollectionListCtrl::DeleteSelectedTracks() {

  try {

    wxBusyCursor busy;

    // Collect selected indices in a vector
    long index = GetNextItem(-1, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    vector<uint> indices;
    while (index != -1) {
      indices.push_back(index);
      index = GetNextItem(index, wxLIST_NEXT_ALL, wxLIST_STATE_SELECTED);
    }

    // None selected
    if (indices.size() < 1) {
      return;
    }

    // Ask for confirmation if deleting multiple files
    if (indices.size() > 1) {
      wxString msg = "You selected ";
      msg << indices.size() << " tracks for removal. Are you sure you want to "
        "delete these tracks?";
      int ans = ::wxMessageBox(msg, "Confirm Delete",
        wxYES_NO | wxICON_QUESTION);
      if (ans != wxYES) {
        return;
      }
    }

    // Deselect them
    vector<uint>::const_iterator it;
    for (it = indices.begin(); it != indices.end(); ++it) {
      SetItemState(*it, !wxLIST_STATE_SELECTED, wxLIST_STATE_SELECTED);
    }

    // Delete them
    coll->DeleteTracks(indices);

  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/////////////////////////////////////////////////////////////////////////////
  Update
/////////////////////////////////////////////////////////////////////////////*/
void CollectionListCtrl::Update() {

  try {
    int count = coll->GetCount();

    // Setting a new count will force the control to be refreshed
    SetItemCount(count);

    wxString s;
    s << count << " tracks";
    statusBar->SetStatusText(s);
  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/////////////////////////////////////////////////////////////////////////////
  GetSortString
/////////////////////////////////////////////////////////////////////////////*/
string CollectionListCtrl::GetSortString() {

  // Make string from sortHistory
  ostringstream ss;
  deque<string>::iterator it;
  for (it = sortHistory.begin(); it != sortHistory.end(); ++it) {
    ss << *it << ", ";
  }
  ss.seekp(-2, ios::cur);
  ss << " "; // remove last comma

  return ss.str();

}

/////////////////////////////////////////////////////////////////////////////
  BuildMe
/////////////////////////////////////////////////////////////////////////////*/
void CollectionListCtrl::BuildMe() {

  InsertColumn(ARTIST_COL, "Artist", wxLIST_FORMAT_LEFT);
  InsertColumn(TITLE_COL, "Title", wxLIST_FORMAT_LEFT);
  InsertColumn(TEMPO_COL, "Tempo", wxLIST_FORMAT_LEFT);
  InsertColumn(LENGTH_COL, "Length", wxLIST_FORMAT_RIGHT);
  InsertColumn(FORMAT_COL, "Format", wxLIST_FORMAT_LEFT);
  InsertColumn(GENRE_COL, "Genre", wxLIST_FORMAT_LEFT);
  InsertColumn(SOURCE_COL, "Source", wxLIST_FORMAT_LEFT);
  InsertColumn(ADDED_COL, "Added", wxLIST_FORMAT_LEFT);

  wxConfigBase* cfg = wxConfigBase::Get();
  long artistWidth = cfg->Read("/CollectionListCtrl/artistWidth", 160L);
  long titleWidth = cfg->Read("/CollectionListCtrl/titleWidth", 190L);
  long tempoWidth = cfg->Read("/CollectionListCtrl/tempoWidth", 60L);
  long lengthWidth = cfg->Read("/CollectionListCtrl/lengthWidth", 60L);
  long formatWidth = cfg->Read("/CollectionListCtrl/formatWidth", 60L);
  long genreWidth = cfg->Read("/CollectionListCtrl/genreWidth", 110L);
  long sourceWidth = cfg->Read("/CollectionListCtrl/sourceWidth", 300L);
  long addedWidth = cfg->Read("/CollectionListCtrl/addedWidth", 100L);

  SetColumnWidth(ARTIST_COL, artistWidth);
  SetColumnWidth(TITLE_COL, titleWidth);
  SetColumnWidth(TEMPO_COL, tempoWidth);
  SetColumnWidth(LENGTH_COL, lengthWidth);
  SetColumnWidth(FORMAT_COL, formatWidth);
  SetColumnWidth(GENRE_COL, genreWidth);
  SetColumnWidth(SOURCE_COL, sourceWidth);
  SetColumnWidth(ADDED_COL, addedWidth);

  // TODO : create an image list with arrows for sort order
  //SetImageList(m_imageListSmall, wxIMAGE_LIST_SMALL);

}
```

```
/*****************************************************************************

  FileInStream.h

  Subclass of InStream dealing with input from all audio files.

  STK WvIn takes care of WAV, AIFF, SND (AU), MAT-file (Matlab) and RAW.
  MadWrapper takes care of MP3.

  Open must be called with a valid filename before any of the other methods
  can be called.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef FILEINSTREAM_H
#define FILEINSTREAM_H

#include "InStream.h"
#include "MadWrapper.h"
#include "Globals.h"

#include "WvIn.h"

#include <string>

class FileInStream : public InStream {

  public:

    /*************************************************************************
      Constructor
    *************************************************************************/
    FileInStream();

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~FileInStream();

    /*************************************************************************
      Opens the named file.
    *************************************************************************/
    virtual void Open(const std::string&);

    /*************************************************************************
      Closes the stream. It's not mandatory to close a stream before opening
      a new one. The previous one will be closed automatically on the next
      call to Open.
    *************************************************************************/
    virtual void Close();

    /*************************************************************************
      Returns true if there is more data to read from the stream.
    *************************************************************************/
    virtual bool HasMore() { return !finished; }

    /*************************************************************************
      Read one sample. If the file is a stereo file, the average of the two
      channels is returned.
    *************************************************************************/
    virtual double Read();

    /*************************************************************************
      Set the sample rate at which values should be returned by the stream.
    *************************************************************************/
    virtual void SetRate(uint);

    /*************************************************************************
      Returns length of stream in seconds.
    *************************************************************************/
    virtual int GetLength();

    /*************************************************************************
      Returns a number between 0 and 1 indicating how much of the stream
      has been read.
    *************************************************************************/
    virtual float GetProgress();

    /*************************************************************************
      Returns name of open file.
    *************************************************************************/
    virtual std::string GetFileName() { return filename; }


  private:

    /*************************************************************************
      Fills audio buffer with data.
    *************************************************************************/
    void FillBuffer();

    /*************************************************************************
      Formats error string and throws InStreamException.
    *************************************************************************/
    void ErrorStk(const std::string&, StkError&);


    /*************************************************************************
      Variables
    *************************************************************************/
    static const uint bufMax = 500 * 1000 / 8; // buffer size, first number is
                                               // kB (a double takes 8 bytes)

    WvIn wvIn;        // STK wav-reading class
    MadWrapper mp3in; // wrapper around MAD for reading mp3 input

    std::string filename;  // currently open file

    double buffer[bufMax]; // buffer of audio ready to return
    uint bufLen;           // actual buffer length
    uint bufIdx;           // buffer index
    ulong sampleCount;     // number of samples returned

    bool allRead;  // has all audio has been read into buffer?
    bool finished; // is there no more audio to return?

    enum filetype {
      MP3,
      WAV
    };

    filetype format; // format of currently open file

};

#endif
```

```
/*******************************************************************************

  FileInStream.cpp

  Subclass of InStream dealing with input from all audio files.

  STK WvIn takes care of WAV, AIFF, SND (AU), MAT-file (Matlab) and RAW.
  MadWrapper takes care of MP3.

  Open must be called with a valid filename before any of the other methods
  can be called.

  Author: Erik Jälevik

*******************************************************************************/

#include "FileInStream.h"

#include "Stk.h"            // setSampleRate

#include <sstream>          // ostringstream
#include <algorithm>        // transform

using namespace std;

/*******************************************************************************
  Constructor
*******************************************************************************/
FileInStream::FileInStream() :
  bufLen(bufMax),
  bufIdx(bufLen),
  sampleCount(0),
  allRead(true),
  finished(true) {

  Stk::setSampleRate(readRate);
  mp3in.SetRate(readRate);

}

/*******************************************************************************
  Destructor
*******************************************************************************/
FileInStream::~FileInStream() {

  if (mp3in.IsOpen()) {
    mp3in.Close();
  }

}

/*******************************************************************************
  Open
*******************************************************************************/
void FileInStream::Open(const string& fname) {

  filename = fname;

  // Determine file type by looking at extension
  string::size_type idx = filename.rfind('.');
  string ext = filename.substr(idx + 1);

  // Make lower case
  transform(ext.begin(), ext.end(), ext.begin(), tolower);

  if (ext == "mp3") {
    if (mp3in.IsOpen()) {
      mp3in.Close();
    }
    mp3in.Open(filename);
    format = MP3;
  }

  else {
    try {
      wvIn.openFile(filename.c_str());
      format = WAV;
    }
    catch (StkError& e) {
      string s = "File '" + filename + "' not found or could not be opened.";
      ErrorStk(s, e);
    }
  }

  sampleCount = 0;
  bufLen = bufMax;
  bufIdx = bufLen;
  allRead = false;
  finished = false;

}

/*******************************************************************************
  Close
*******************************************************************************/
void FileInStream::Close() {

  if (format == MP3) {
    mp3in.Close();
  }
  else {
    wvIn.closeFile();
  }

  allRead = true;
  finished = true;

}

/*******************************************************************************
  Read
*******************************************************************************/
double FileInStream::Read() {

  // Check if buffer is empty.
  // Using while guards against the situation when FillBuffer returns 0 bytes
  // which can happen at the end of a file.
  while (bufIdx >= bufLen) {
    if (!allRead) {
      FillBuffer();
    }
    else {
      finished = true;
      return 0.0;
    }
  }

  ++sampleCount;

  return buffer[bufIdx++];

}

/*******************************************************************************
  SetRate
*******************************************************************************/
```

```
void FileInStream::SetRate(uint r) {

  InStream::SetRate(r);
  Stk::setSampleRate(readRate);
  mp3in.SetRate(readRate);

}

/*****************************************************************************
  GetLength
*****************************************************************************/
int FileInStream::GetLength() {

  if (format == MP3) {
    return mp3in.GetLength();
  }
  else {
    return static_cast<int>(wvIn.getSize() / wvIn.getFileRate());
  }

}

/*****************************************************************************
  GetProgress
*****************************************************************************/
float FileInStream::GetProgress() {

  float secsRead = static_cast<float>(sampleCount) / readRate;

  if (format == MP3) {
    return secsRead / mp3in.GetLength();
  }
  else {

    //LOG("sampleCount: " << sampleCount <<
    //    ", readRate: " << readRate <<
    //    ", getSize: " << wvIn.getSize() <<
    //    ", getFileRate: " << wvIn.getFileRate() <<
    //    ", progr: " << (static_cast<float>(sampleCount) / readRate) << "/" <<
    //    (wvIn.getSize() / wvIn.getFileRate()) << endl);

    // Number of seconds read divided by total number of seconds
    return secsRead / (wvIn.getSize() / wvIn.getFileRate());

  }

}

/*****************************************************************************
  FillBuffer
*****************************************************************************/
void FileInStream::FillBuffer() {

  if (format == MP3) {
    unsigned long nSamples = mp3in.Read(buffer, bufLen);
    if (nSamples < bufLen) {
      bufLen = nSamples;
      allRead = true;
    }
  }

  else {
    try {
      uint i;
      for (i = 0; i < bufLen && !wvIn.isFinished(); ++i) {
        buffer[i] = wvIn.tick();
      }
      if (i < bufLen) {
        bufLen = i;
        allRead = true;
      }
    }
    catch (StkError& e) {
      string s = "Failed reading next sample from file '" + filename + "'.";
      ErrorStk(s, e);
    }

  }

  bufIdx = 0;

}


/*****************************************************************************
  ErrorStk
*****************************************************************************/
void FileInStream::ErrorStk(const string& msg, StkError& e) {

  ostringstream tmp;
  tmp << msg;
  #ifdef _DEBUG
    tmp << "\n\nSTK Error: " << e.getMessage() << endl;
    tmp << "STK Error Type: " << e.getType();
  #endif

  throw InStreamException(tmp.str());

}
```

```
/*****************************************************************************

  FilenameParser.h

  Class that tries to work out the artist and the title given a filename.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef FILENAMEPARSER_H
#define FILENAMEPARSER_H

#include <string>

class FilenameParser {

  public:

    /*************************************************************************
      Parameterised constructor parses the filename given to it.
      Parameters:
      fname - filename to parse
    *************************************************************************/
    FilenameParser(const std::string& fname) { path = fname; DoParse(); }
    FilenameParser() { }
    virtual ~FilenameParser() { }

    /*************************************************************************
      Parses a new filename.
    *************************************************************************/
    void Parse(const std::string& fname) { path = fname; DoParse(); }

    /*************************************************************************
      Returns the path it was initialised with.
    *************************************************************************/
    std::string GetPath() { return path; }

    /*************************************************************************
      Returns the filename only.
    *************************************************************************/
    std::string GetFilename() { return filename; }

    /*************************************************************************
      Returns the parent folder of the file.
    *************************************************************************/
    std::string GetFolder() { return folder; }

    /*************************************************************************
      Returns the artist found in name. The empty string if none found.
    *************************************************************************/
    std::string GetArtist() { return artist; }

    /*************************************************************************
      Returns the title found in name. The empty string if none found.
    *************************************************************************/
    std::string GetTitle() { return title; }

    /*************************************************************************
      Returns the extension found in name. The empty string if none found.
    *************************************************************************/
    std::string GetExtension() { return extension; }


  private:

    /*************************************************************************
      Does the actual parsing.
    *************************************************************************/
    void DoParse();

    /*************************************************************************
      Looks for a separator character in the passed string.
    *************************************************************************/
    std::string::size_type FindSeparator(const std::string&);


    /*************************************************************************
      Variables
    *************************************************************************/
    std::string path;      // current full path
    std::string folder;    // current parent folder
    std::string filename;  // current filename
    std::string artist;    // current artist
    std::string title;     // current title
    std::string extension; // current file extension

};

#endif
```

```cpp
/*****************************************************************************

  FilenameParser.cpp

  Class that tries to work out the artist and the title given a filename.

  Author: Erik Jälevik

*****************************************************************************/

#include "FilenameParser.h"
#include "Globals.h"

using namespace std;

/*****************************************************************************
  Parse
*****************************************************************************/
void FilenameParser::DoParse() {

  folder = "";
  filename = "";
  artist = "";
  title = "";
  extension = "";

  // Get filename
  string::size_type idxFile = path.find_last_of("/\\");
  if (idxFile != string::npos) {
    filename = path.substr(idxFile + 1);
  }
  else {
    // No slashes found, filename has no path information
    filename = path;
  }

  // Get folder
  if (idxFile != string::npos) {
    string::size_type idxFolder = path.find_last_of("/\\", idxFile - 1);
    if (idxFolder != string::npos) {
      folder = path.substr(idxFolder + 1, idxFile - idxFolder - 1);
    }
  }

  // Get extension
  string::size_type idxExt = filename.rfind('.');
  if (idxExt != string::npos) {
    extension = filename.substr(idxExt + 1);
  }

  // Get a temporary string consisting of the filename without the extension
  string tmp;
  if (idxExt != string::npos) {
    tmp = filename.substr(0, idxExt);
  }
  else {
    tmp = filename;
  }

  string::size_type idxSep = FindSeparator(tmp);

  // No separator found, set title to filename and return
  if (idxSep == string::npos) {
    title = tmp;
    return;
  }

  // Separator found, get string before separator
  artist = tmp.substr(0, idxSep);

  // Check if it's only a number
  bool isNum = true;
  for (string::size_type i = 0; i < artist.size(); ++i) {
    // If finding a non-digit, a space or a full stop, it's not a number
    if (!isdigit(artist[i]) && artist[i] != ' ' && artist[i] != '.') {
      isNum = false;
      break;
    }
  }

  if (isNum) {

    // It's a number, delete number and look for next separator
    artist = "";
    tmp.erase(0, idxSep + 1);

    idxSep = FindSeparator(tmp);

    // No separator found, set title to filename after number and return
    if (idxSep == string::npos) {
      title = tmp;
      // Remove whitespace
      title.erase(0, title.find_first_not_of(" _"));
      title.erase(title.find_last_not_of(" _") + 1);
      return;
    }

    // Separator found, extract new artist
    else {
      artist = tmp.substr(0, idxSep);
    }

  }

  // Get title from after separator
  title = tmp.substr(idxSep + 1);

  // Remove whitespace
  artist.erase(0, artist.find_first_not_of(" _"));
  artist.erase(artist.find_last_not_of(" _") + 1);
  title.erase(0, title.find_first_not_of(" _"));
  title.erase(title.find_last_not_of(" _") + 1);

}

/*****************************************************************************
  FindSeparator
*****************************************************************************/
string::size_type FilenameParser::FindSeparator(const string& s) {

  // Try splitting on " - "
  string::size_type idx = s.find(" - ");
  if (idx != string::npos) {
    return ++idx; // idx should correspond to location of -
  }

  // Try splitting on "_-_"
  idx = s.find("_-_");
  if (idx != string::npos) {
    return ++idx; // idx should correspond to location of -
  }

  // Try splitting on "-"
  idx = s.find('-');
  return idx;
```

```
}
```

```
/*****************************************************************************

  FilterPanel.h

  GUI panel object for the filter bar above the main collection view.

  It inherits from Observer and registers with MusicCollection in order to
  update its Format and Genre drop down menus as soon as media or genres
  are added or removed.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef FILTERPANEL_H
#define FILTERPANEL_H

#include "MusicCollection.h"
#include "MusicTrack.h"
#include "Observer.h"

#include <wx/spinctrl.h>
#include <wx/tglbtn.h>

#include "toggle.h" // for wxCustomButton

class FilterPanel : public wxPanel, Observer {

  public:

    /*************************************************************************
      Constructor takes parent window and window ID.
    *************************************************************************/
    FilterPanel(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~FilterPanel();

    /*************************************************************************
      Handler for text control changes. Sets filter options.
    *************************************************************************/
    void OnChange(wxCommandEvent& event);

    /*************************************************************************
      Separate handler for spin changes needed.
    *************************************************************************/
    void OnSpin(wxSpinEvent& event);

    /*************************************************************************
      Handler for when Filter button is pressed.
    *************************************************************************/
    void OnFilter(wxCommandEvent&);

    /*************************************************************************
      Observer method.
    *************************************************************************/
    void Update();


  private:

    /*************************************************************************
      Autogenerated method for building component.
    *************************************************************************/
    void BuildMe();
```

```
    /*************************************************************************
      Fill format drop down with values from MusicCollection.
    *************************************************************************/
    void PopulateFormat();

    /*************************************************************************
      Fill genre drop down with values from MusicCollection.
    *************************************************************************/
    void PopulateGenre();

    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxTextCtrl* artistTC;
    wxTextCtrl* titleTC;
    wxChoice* formatDD;
    wxChoice* genreDD;
    wxSpinCtrl* tempoSC;
    wxSpinCtrl* tempoPMSC;
    wxCustomButton* filterBut;

    // Model objects
    MusicCollection* coll;
    MusicTrack filter; // used to hold the current filter settings

    enum {
      ID_ARTIST_TC,
      ID_TITLE_TC,
      ID_FORMAT_DD,
      ID_GENRE_DD,
      ID_TEMPO_SC,
      ID_TEMPOPM_SC,
      ID_FILTER_BUT
    };

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/*************************************************************************

  FilterPanel.cpp

  GUI panel object for the filter bar above the main collection view.

  Author: Erik Jälevik

*************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "FilterPanel.h"
#include "Globals.h"
#include "Icons.h"

#include <wx/confbase.h>

using namespace std;

/*************************************************************************
  Constructor
*************************************************************************/
FilterPanel::FilterPanel(wxWindow* parent, wxWindowID id) :
  wxPanel(parent, id, wxDefaultPosition, wxDefaultSize,
    wxSUNKEN_BORDER | wxTAB_TRAVERSAL) {

  coll = MusicCollection::Instance();

  BuildMe();

  filter.SetArtist(artistTC->GetValue().c_str());
  filter.SetTitle(titleTC->GetValue().c_str());
  filter.SetFormat(formatDD->GetStringSelection().c_str());
  filter.SetGenre(genreDD->GetStringSelection().c_str());
  int low = tempoSC->GetValue() - tempoPMSC->GetValue();
  int high = tempoSC->GetValue() + tempoPMSC->GetValue();
  TempoRange& range = filter.GetTempoRange();
  range.Add(0, low);
  range.Add(1, high);

  coll->SetFilter(filter);
  coll->Filter(filterBut->GetValue());

  // Register as observer, used by Genre and Format drop downs
  coll->Attach(this);

}

/*************************************************************************
  Destructor
*************************************************************************/
FilterPanel::~FilterPanel() {

  wxConfigBase* cfg = wxConfigBase::Get();
  cfg->Write("/FilterPanel/artist", artistTC->GetValue());
  cfg->Write("/FilterPanel/title", titleTC->GetValue());
  cfg->Write("/FilterPanel/format", formatDD->GetSelection());
  cfg->Write("/FilterPanel/genre", genreDD->GetSelection());
  cfg->Write("/FilterPanel/tempo", tempoSC->GetValue());
```

```
  cfg->Write("/FilterPanel/tempoPM", tempoPMSC->GetValue());
  cfg->Write("/FilterPanel/filtered", filterBut->GetValue());

}

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(FilterPanel, wxPanel)
  EVT_TEXT(ID_ARTIST_TC, FilterPanel::OnChange)
  EVT_TEXT(ID_TITLE_TC, FilterPanel::OnChange)
  EVT_CHOICE(ID_FORMAT_DD, FilterPanel::OnChange)
  EVT_CHOICE(ID_GENRE_DD, FilterPanel::OnChange)
  EVT_SPINCTRL(ID_TEMPO_SC, FilterPanel::OnSpin)
  EVT_TEXT(ID_TEMPO_SC, FilterPanel::OnChange)
  EVT_SPINCTRL(ID_TEMPOPM_SC, FilterPanel::OnSpin)
  EVT_TEXT(ID_TEMPOPM_SC, FilterPanel::OnChange)
  EVT_TOGGLEBUTTON(ID_FILTER_BUT, FilterPanel::OnFilter)
END_EVENT_TABLE()

/*****************************************************************************
  OnChange
*****************************************************************************/
void FilterPanel::OnChange(wxCommandEvent& event) {

  wxBusyCursor busy;

  try {

    int id = event.GetId();

    if (id == ID_ARTIST_TC) {
      filter.SetArtist(artistTC->GetValue().c_str());
    }
    else if (id == ID_TITLE_TC) {
      filter.SetTitle(titleTC->GetValue().c_str());
    }
    else if (id == ID_FORMAT_DD) {
      filter.SetFormat(formatDD->GetStringSelection().c_str());
    }
    else if (id == ID_GENRE_DD) {
      filter.SetGenre(genreDD->GetStringSelection().c_str());
    }
    else if (id == ID_TEMPO_SC || id == ID_TEMPOPM_SC) {
      int low = tempoSC->GetValue() - tempoPMSC->GetValue();
      low = low < 0 ? 0 : low;
      int high = tempoSC->GetValue() + tempoPMSC->GetValue();
      TempoRange& range = filter.GetTempoRange();
      range.Clear();
      range.Add(0, low);
      range.Add(1, high);
    }

    coll->SetFilter(filter);

    if (filterBut->GetValue()) {
      coll->Filter(true);
    }

  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  OnSpin
*****************************************************************************/
void FilterPanel::OnSpin(wxSpinEvent& event) {

  OnChange(event);

}

/*****************************************************************************
  OnFilter
*****************************************************************************/
void FilterPanel::OnFilter(wxCommandEvent& event) {

  wxBusyCursor busy;

  try {
    coll->Filter(filterBut->GetValue());
  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  Update
*****************************************************************************/
void FilterPanel::Update() {

  // Save current selections
  wxString selectedGenre = genreDD->GetStringSelection();
  wxString selectedFormat = formatDD->GetStringSelection();

  // Get new formats/genres from database
  PopulateFormat();
  PopulateGenre();

  // Check that the selections still exist before resetting them
  if (genreDD->FindString(selectedGenre) != -1) {
    genreDD->SetStringSelection(selectedGenre);
  }
  if (formatDD->FindString(selectedFormat) != -1) {
    formatDD->SetStringSelection(selectedFormat);
  }

}

/*****************************************************************************
  PopulateFormat
*****************************************************************************/
void FilterPanel::PopulateFormat() {

  try {
    // Get all formats and put into drop down box
    const vector<string>& formatVec = coll->GetFormats();
    formatDD->Clear();
    formatDD->Append("");
    for (uint i = 0; i < formatVec.size(); ++i) {
      formatDD->Append(formatVec.at(i).c_str());
    }
  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  PopulateGenre
```

```
/*****************************************************************************/
void FilterPanel::PopulateGenre() {

  try {
    // Get all genres and put into drop down box
    const vector<string>& genreVec = coll->GetGenres();
    genreDD->Clear();
    genreDD->Append("");
    for (uint i = 0; i < genreVec.size(); ++i) {
      genreDD->Append(genreVec.at(i).c_str());
    }
  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*****************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*****************************************************************************/
void FilterPanel::BuildMe() {

  const int labelBorder = 0;

  // Disable event handler while building to prevent filling the text boxes
  // with saved values from triggering OnChange
  SetEvtHandlerEnabled(false);

//  SetBackgroundColour(wxColour(239, 239, 241));
//  SetForegroundColour(wxColour(0, 0, 0));

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString artistVal = cfg->Read("/FilterPanel/artist", "");
  wxString titleVal = cfg->Read("/FilterPanel/title", "");
  long formatVal = cfg->Read("/FilterPanel/format", 0L);
  long genreVal = cfg->Read("/FilterPanel/genre", 0L);
  wxString tempoVal = cfg->Read("/FilterPanel/tempo", "120");
  wxString tempoPMVal = cfg->Read("/FilterPanel/tempoPM", "100");
  bool filtered; cfg->Read("/FilterPanel/filtered", &filtered, false);

  wxBoxSizer *item0 = new wxBoxSizer(wxHORIZONTAL);
  item0->Add(5, 0);

  wxStaticText *item1 = new wxStaticText( this, -1, "Artist",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item0->Add( item1, 0, wxALIGN_CENTER|wxALL, labelBorder );

  artistTC = new wxTextCtrl( this, ID_ARTIST_TC, artistVal, wxDefaultPosition,
    wxSize(80,-1), 0 );
  item0->Add( artistTC, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item3 = new wxStaticText( this, -1, "Title",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item0->Add( item3, 0, wxALIGN_CENTER|wxALL, labelBorder );

  titleTC = new wxTextCtrl( this, ID_TITLE_TC, titleVal, wxDefaultPosition,
    wxSize(80,-1), 0 );
  item0->Add( titleTC, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item5 = new wxStaticText( this, -1, "Format",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item0->Add( item5, 0, wxALIGN_CENTER|wxALL, labelBorder );

  formatDD = new wxChoice(this, ID_FORMAT_DD, wxDefaultPosition,
    wxSize(50,-1), 0, NULL, 0);
  PopulateFormat();
  formatDD->SetSelection(formatVal);
  item0->Add( formatDD, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item7 = new wxStaticText( this, -1, "Genre",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item0->Add( item7, 0, wxALIGN_CENTER|wxALL, labelBorder );

  genreDD = new wxChoice( this, ID_GENRE_DD, wxDefaultPosition,
    wxSize(85,-1), 0, NULL, 0 );
  PopulateGenre();
  genreDD->SetSelection(genreVal);
  item0->Add( genreDD, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item9 = new wxStaticText( this, -1, "Tempo",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item0->Add( item9, 0, wxALIGN_CENTER|wxALL, labelBorder );

  // TODO : call beat tracker here to find out min and max tempo
  tempoSC = new wxSpinCtrl( this, ID_TEMPO_SC, tempoVal, wxDefaultPosition,
    wxSize(50,-1), wxSP_ARROW_KEYS, 80, 200, 0 );
  item0->Add( tempoSC, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item11 = new wxStaticText( this, -1, "±",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item0->Add( item11, 0, wxALIGN_CENTER|wxALL, labelBorder );

  tempoPMSC = new wxSpinCtrl( this, ID_TEMPOPM_SC, tempoPMVal,
    wxDefaultPosition, wxSize(50,-1), wxSP_ARROW_KEYS, 0, 100, 0 );
  item0->Add( tempoPMSC, 0, wxALIGN_CENTER|wxALL, 5 );

  filterBut = new wxCustomButton(this, ID_FILTER_BUT,
    Icons::Filter(), wxDefaultPosition, wxSize(22, 20));
  filterBut->SetToolTip("Filter on/off");
  filterBut->SetValue(filtered);
  item0->Add( filterBut, 0, wxALIGN_CENTER|wxALL, 5 );

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);

  SetEvtHandlerEnabled(true);

}
```

```
/***************************************************************************

  FreeDB.h

  Wrapper around the FreeDB access facilities of the AKRip library.

  Author: Erik Jälevik

***************************************************************************/

#ifndef FREEDB_H
#define FREEDB_H

#include "Globals.h"

#include "akrip32.h"

#include <string>
#include <vector> // returned by various methods
#include <memory> // auto_ptr

class FreeDBAlbum; // these are defined
class FreeDBSite;  // further down

class FreeDB {

  public:

    /*************************************************************************
      Constructor sets default parameters and discovers CD drives.
    *************************************************************************/
    FreeDB();

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~FreeDB() { }

    /*************************************************************************
      Exception class used for anything that can go wrong when reading input.
    *************************************************************************/
    class FreeDBException : public std::runtime_error {
      public:
        FreeDBException(const std::string& msg) : std::runtime_error(msg) { }
    };

    /*************************************************************************
      Set FreeDB URL to use. Must be called before attempting to query
      FreeDB.
    *************************************************************************/
    virtual void SetUrl(const std::string&);

    /*************************************************************************
      Set FreeDB port to use. Must be called before attempting to query
      FreeDB.
    *************************************************************************/
    virtual void SetPort(int p) { port = p; }

    /*************************************************************************
      Returns list of possible matches for CD in passed drive.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<std::string> > QueryCD(int);

    /*************************************************************************
      Gets CD info for one of the matches returned by QueryFreeDB.
      The int parameter is the 0-based index of one of the returned matches.
      QueryFreeDB must have been called before using this method.
    *************************************************************************/
    virtual std::auto_ptr<FreeDBAlbum> GetCDContents(int);

    /*************************************************************************
      Queries the FreeDB server for list of available server sites.
    *************************************************************************/
    virtual std::auto_ptr<std::vector<FreeDBSite> > GetSites();


  private:

    /*************************************************************************
      Parse FreeDB result string and return album data.
    *************************************************************************/
    std::auto_ptr<FreeDBAlbum> ParseResult(CDDBQUERYITEM&, char*);

    /*************************************************************************
      Parses a particular field from the stream and adds it to string.
    *************************************************************************/
    void ParseField(std::istringstream&, std::string&);

    /*************************************************************************
      Replaces escape sequences \t, \n and \\ in passed string.
    *************************************************************************/
    void ReplaceEsc(std::string&);

    /*************************************************************************
      For getting hold of CD handle.
    *************************************************************************/
    HCDROM GetCDHandleMem(int);

    /*************************************************************************
      Throws exception.
    *************************************************************************/
    void Error(const char*);


    /*************************************************************************
      Variables
    *************************************************************************/
    static const int maxFreeDBResults = 20; // max returned matches
    static const int maxCDInfoLen = 10000;  // max returned chars in CD info
    static const int maxFreeDBSites = 50;   // max sites returned

    CDLIST cdList;   // list of CDs present in system
    CDDBQUERY query; // last returned query result for getting disc info
    CDDBQUERYITEM queryItems[maxFreeDBResults]; // items in query

    std::string server; // host name part of server URL
    std::string cgi;    // CGI part of server URL
    int port;           // port to use (normally 80)

};

/***************************************************************************
  Data holder for an album entry.
***************************************************************************/
class FreeDBAlbum {

  public:
    std::string artist;
    std::string title;
    std::vector<std::string> tracks;
    std::string genre;

};

/***************************************************************************
  Data holder for a FreeDBSite.
```

```
*****************************************************************************/
class FreeDBSite {

  public:
    std::string server;
    std::string url;
    int port;

};

#endif
```

```
/*****************************************************************************

  FreeDB.cpp

  Wrapper around the FreeDB access facilities of the AKRip library.

  Author: Erik Jälevik

*****************************************************************************/

#include "FreeDB.h"

#include <sstream>  // ostringstream, istringstream

using namespace std;

/*****************************************************************************
  Constructor
*****************************************************************************/
FreeDB::FreeDB() :
  server("www.freedb.org"),
  cgi("/~cddb/cddb.cgi"),
  port(80) {

  // Init query with 0
  query.num = 0;
  query.q = 0;

  // Get list of cd drives
  memset(&cdList, 0, sizeof(cdList));
  cdList.max = MAXCDLIST;
  GetCDList(&cdList);

  if (cdList.num == 0) {
    Error("No CD-ROM drives found.");
  }

}

/*****************************************************************************
  SetUrl
*****************************************************************************/
void FreeDB::SetUrl(const string& url) {

  string tmp = url;

  // Remove any http prefix
  if (tmp.substr(0, 7) == "http://") {
    tmp.erase(0, 7);
  }

  string::size_type idx = tmp.find_first_of('/');
  if (idx != string::npos) {
    server = tmp.substr(0, idx);
    cgi = tmp.substr(idx);
  }
  else {
    Error("The FreeDB URL is not in a valid format.");
  }

}

/*****************************************************************************
  QueryCD
*****************************************************************************/
auto_ptr<vector<string> > FreeDB::QueryCD(int device) {

  // Check that device param is not too high
```

```
  if (device > cdList.num - 1) {
    Error("Specified CD-ROM drive does not seem to exist.");
  }

  // Get handle
  HCDROM handle = GetCDHandleMem(device);
  if (!handle) {
    ostringstream os;
    os << "Could not get handle to CD-ROM drive " << device;
    Error(os.str().c_str());
  }

  memset(&queryItems, 0, sizeof(queryItems));
  query.num = maxFreeDBResults;
  query.q = queryItems;

  CDDBSetOption(CDDB_OPT_SERVER, const_cast<char*>(server.c_str()), 0);
  CDDBSetOption(CDDB_OPT_CGI, const_cast<char*>(cgi.c_str()), 0);
  CDDBSetOption(CDDB_OPT_HTTPPORT, "", port);
  CDDBSetOption(CDDB_OPT_USER, "beatrakuser@beatrak", 0);
  CDDBSetOption(CDDB_OPT_AGENT, "Beatrak 0.1", 0);
  CDDBSetOption(CDDB_OPT_USECDPLAYERINI, "", false);

  LOG("About to query FreeDB server: " << server << cgi << ":" <<
    port << endl);

  // Send query
  if (CDDBQuery(handle, &query) == SS_COMP) {

    if (query.num > 0) {
      auto_ptr<vector<string> > v(new vector<string>);
      for(int i = 0; i < query.num; ++i) {
        ostringstream tmp;
        tmp << queryItems[i].artist << " - ";
        tmp << queryItems[i].title << ", ";
        tmp << queryItems[i].categ << " [id: ";
        tmp << queryItems[i].cddbId << "]";
        v->push_back(tmp.str());
      }
      CloseCDHandle(handle);
      return v;
    }
    // If no connection, or if no matches were found, 0 items will be returned
    else {
      CloseCDHandle(handle);
      Error("No matches were found or the connection failed.");
    }

  }
  else {
    CloseCDHandle(handle);
    Error("FreeDB query failed.");
  }

}

/*****************************************************************************
  GetCDContents
*****************************************************************************/
auto_ptr<FreeDBAlbum> FreeDB::GetCDContents(int choice) {

  if (choice > (query.num - 1)) {
    throw logic_error("CDInStream::QueryFreeDB must be called before "
      "CDInStream::GetCDContents can be called and one of the valid, "
      "returned indices must be passed to GetCDContents.");
  }

  char info[maxCDInfoLen]; // buffer that receives CD information

  if (CDDBGetDiskInfo(&query.q[choice], info, maxCDInfoLen) == SS_COMP) {
    return ParseResult(query.q[choice], info);
  }
  else {
    Error("Failed to get CD information from FreeDB.");
  }

}

/*****************************************************************************
  GetSites
*****************************************************************************/
auto_ptr<vector<FreeDBSite> > FreeDB::GetSites() {

  CDDBSITELIST siteList;
  CDDBSITE sites[maxFreeDBSites];

  memset(&sites, 0, sizeof(CDDBSITE) * maxFreeDBSites);

  siteList.num = maxFreeDBSites;
  siteList.s = (LPCDDBSITE)&sites;

  if (CDDBGetServerList(&siteList) != SS_ERR) {

    if (siteList.num > 0) {
      auto_ptr<vector<FreeDBSite> > v(new vector<FreeDBSite>);
      for(int i = 0; i < siteList.num; ++i) {
        FreeDBSite site;
        ostringstream os;
        os << siteList.s[i].szServer << ", " << siteList.s[i].szLocation;
        site.server = os.str();
        os.str("");
        os << siteList.s[i].szServer << siteList.s[i].szCGI;
        site.url = os.str();
        site.port = siteList.s[i].iPort;
        v->push_back(site);
      }
      return v;
    }
    // If no connection, or if no matches were found, 0 items will be returned
    else {
      Error("No servers were found or the connection failed.");
    }

  }
  else {
    Error("Could not retrieve list of sites from FreeDB.");
  }

}

/*****************************************************************************
  ParseResult
*****************************************************************************/
auto_ptr<FreeDBAlbum> FreeDB::ParseResult(CDDBQUERYITEM& queryItem,
                                          char* info) {

  // Real max is 256 but using the double just to be on the safe side
  int maxLineLen = 512;

  // Use genre from query initially and replace it if another one is found
  // in result further down
  string genre = queryItem.categ;
  string songs[99]; // a CD holds max 99 songs

  // Convert return data into a stream for easier parsing
  istringstream in(info);
```

```
// Some temporary holders
string s;
char c;

// Keeping track of highest track number
int maxNum = 0;

while (in.good()) {

  s = "";

  in >> c; // this ignores both whitespace and new lines

  // Ignore comments
  if (c == '#') {
    in.ignore(maxLineLen, '\n'); // discard until end of line
  }

  // Deal with other lines
  else {

    // Keep reading chars until one is not a letter
    while (isalpha(c)) {
      s = s + c;
      in >> c;
    }
    in.unget(); // read one too many, put it back

    if (s == "DGENRE") {
      ParseField(in, genre);
    }

    else if (s == "TTITLE") {
      int num;
      if (in >> num) {
        ParseField(in, songs[num]);
        maxNum = num > maxNum ? num : maxNum;
      }
    }

    else {
      in.ignore(maxLineLen, '\n'); // discard until end of line
    }

  } // else not a comment

} // while in.good()

// If stream reading stopped due to an error and it wasn't end of file,
// throw an exception.
if (!in.eof()) {
  Error("The data returned from FreeDB could not be read.");
}

// Make first letter of genre uppercase
genre.replace(0, 1, 1, static_cast<char>(toupper(genre[0])));

// Go through all fields and replace escape sequences
ReplaceEsc(genre);
for (int i = 0; i <= maxNum; ++i) {
  ReplaceEsc(songs[i]);
}

// Insert data in FreeDBAlbum
auto_ptr<FreeDBAlbum> album(new FreeDBAlbum());
album->artist = queryItem.artist;
album->title = queryItem.title;
album->genre = genre;
  for (int i = 0; i <= maxNum; ++i) {
    album->tracks.push_back(songs[i]);
  }

  return album;

}

/*****************************************************************************
  ParseField
*****************************************************************************/
void FreeDB::ParseField(istringstream& in, string& s) {

  // FreeDB data seems to come back with CRLF line breaks and C++ seems to
  // expect CR, so need to read char by char and keep an eye out for both
  // CR and LF. This also makes sense since we can't assume FreeDB data
  // to adhere to any specific platform convention for line breaks.

  char c;
  in >> c; // reads whitespace, =

  in.get(c); // reads first char
  while (in && c != 10 && c != 13) { // 10 = LF, 13 = CR
    s = s + c; // append in case second line of field
    in.get(c);
  }

}

/*****************************************************************************
  ReplaceEsc
*****************************************************************************/
void FreeDB::ReplaceEsc(string& s) {

  string::size_type idx = s.find("\\n");
  while (idx != string::npos) {
    s.replace(idx, 2, "\n");
    idx = s.find("\\n", idx);
  }

  idx = s.find("\\t");
  while (idx != string::npos) {
    s.replace(idx, 2, "\t");
    idx = s.find("\\t", idx);
  }

  idx = s.find("\\\\");
  while (idx != string::npos) {
    s.replace(idx, 2, "\\");
    idx = s.find("\\\\", idx);
  }

}

/*****************************************************************************
  GetCDHandleMem
*****************************************************************************/
HCDROM FreeDB::GetCDHandleMem(int dev) {

  GETCDHAND gcd;
  memset(&gcd, 0, sizeof(gcd));
  gcd.size = sizeof(gcd);
  gcd.ver = 1;
  gcd.ha = cdList.cd[dev].ha;
  gcd.tgt = cdList.cd[dev].tgt;
  gcd.lun = cdList.cd[dev].lun;
  gcd.readType = CDR_ANY;
  gcd.numJitter = 1;
```

```
  gcd.numOverlap = 3;

  HCDROM cdh = GetCDHandle(&gcd);
  return cdh;

}

/*****************************************************************************
  Error
*****************************************************************************/
void FreeDB::Error(const char* msg) {

  ostringstream tmp;
  tmp << msg;
  #ifdef _DEBUG
    tmp << "\n\nAKRip error: " << GetAspiLibError() << "\n";
    tmp << "ASPI error: " << static_cast<int>(GetAspiLibAspiError());
  #endif

  throw FreeDBException(tmp.str());

}
```

```
/*****************************************************************************

  Globals.h

  Some typedefs and other things needed by the entire application.

  Defines the macro LOG(msg) and the Logger class for outputting messages
  to a log file in debug builds.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef GLOBALS_H
#define GLOBALS_H

#include <map>

typedef unsigned int uint;
typedef unsigned short ushort;
typedef unsigned long ulong;

typedef std::map<uint, ushort> IntShortMap;


// In a debug build, define LOG and Logger
#ifdef _DEBUG

  #include <fstream>
  #include <string>

  #define LOG(msg) Logger::log << Logger::TimePrefix() << msg

  class Logger {

    public:
      static std::ofstream log;
      static std::string TimePrefix();

  };

// Otherwise it does nothing
#else

  #define LOG(msg)

#endif

#endif
```

```
/*****************************************************************************

  Globals.cpp

  Contains the implementation for the debug logging stream.

  Author: Erik Jälevik

*****************************************************************************/

#ifdef _DEBUG

#include "Globals.h"

#include <ctime>

// Open stream. Initialization of static member has to be outside of methods.
std::ofstream Logger::log("output/log.txt", std::ios::out | std::ios::app);

std::string Logger::TimePrefix() {

  // Get and format current time
  time_t t = time(NULL);
  tm* now = localtime(&t);
  char str[21];
  strftime(str, 21, "%d/%m/%y %H.%M.%S: ", now);
  return str;

}

#endif
```

```
/*****************************************************************************

  HanningFollower.h

  An envelope follower that uses a Hanning window to convolve the signal.
  Since computing convolution is computationally expensive, convolution only
  occurs when GetOutput is called. Calling Tick does not return a value, it
  merely stores the value internally.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef HANNINGFOLLOWER_H
#define HANNINGFOLLOWER_H

#include "Globals.h"

class HanningFollower {

  public:

    /*************************************************************************
      Parameters:
      lenSecs - window length in seconds
      freq - audio sample rate
    *************************************************************************/
    HanningFollower(double lenSecs, int freq);
    HanningFollower();
    ~HanningFollower();

    /*************************************************************************
      Send one sample to filter for storing internally.
    *************************************************************************/
    void Tick(double input) {
      buffer[index] = input < 0 ? -input : input;
      if (++index == length) { index = 0; }
    }

    /*************************************************************************
      Perform convolution and return output.
    *************************************************************************/
    double GetOutput();


  private:

    /*************************************************************************
      Variables
    *************************************************************************/
    double* buffer;  // holds recent inputs
    uint length;     // length of buffer
    uint index;      // pointer to current index in buffer
    double* hanning; // hanning window

};

#endif
```

```
/*****************************************************************************

  HanningFollower.cpp

  An envelope follower that uses a Hanning window to convolve the signal.
  Since computing convolution is computationally expensive, convolution only
  occurs when GetOutput is called. Calling Tick does not return a value, it
  merely stores the value internally.

  Author: Erik Jälevik

*****************************************************************************/

#include "HanningFollower.h"

#include <cmath> // cos

/*****************************************************************************
  Constructor (double, double)
*****************************************************************************/
HanningFollower::HanningFollower(double lenSecs, int freq) :
  index(0) {

  length = static_cast<int>(lenSecs * freq);
  buffer = new double[length];
  hanning = new double[length];

  for (uint i = 0; i < length; ++i) {

    // Initialise buffer to zero
    buffer[i] = 0.0;

    // Create Hanning window.
    // Window is scaled by a factor relative to its length to keep output
    // values within 0.0-1.0.
    hanning[i] = (1.0 / length) * (0.5 - 0.5 * cos(3.1415927 * i / (length)));

  }

}

/*****************************************************************************
  Constructor ()
*****************************************************************************/
HanningFollower::HanningFollower() :
  length(0),
  index(0),
  buffer(0),
  hanning(0) {

}

/*****************************************************************************
  Destructor
*****************************************************************************/
HanningFollower::~HanningFollower() {

  if (buffer != 0) { delete[] buffer; }
  if (hanning != 0) { delete[] hanning; }

}

/*****************************************************************************
  GetOutput
*****************************************************************************/
double HanningFollower::GetOutput() {

  // Perform convolution

  double sum = 0.0;
  int winIndex = 0;

  // Start at end of buffer...
  for (uint i = index; i < length; ++i, ++winIndex) {
    sum += hanning[winIndex] * buffer[i];
  }

  // ...and wrap around
  for (uint i = 0; i < index; ++i, ++winIndex) {
    sum += hanning[winIndex] * buffer[i];
  }

  return sum;


/* RMS calc:

  // store input in buffer
  buffer[index] = input;
  if (++index == length) index = 0;

  // calculate rms
  double sum = 0.0;
  for (int i = 0; i < length; ++i) {
    sum += buffer[i] * buffer[i];
  }

  return sqrt(sum / length);

*/

}
```

```
/*****************************************************************************

  Icons.h

  Contains XPM icons and raw bitmap data accessed through static functions.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef ICONS_H
#define ICONS_H

#include <wx/bitmap.h>
#include <wx/icon.h>

class Icons {

  public:

    // Icons
    static wxBitmap& AddFile();
    static wxBitmap& AddCD();
    static wxBitmap& AddLineIn();
    static wxBitmap& Delete();
    static wxBitmap& Filter();
    static wxBitmap& UncheckedBox();
    static wxBitmap& CheckedBox();
    static wxBitmap& Refresh();
    static wxBitmap& FreeDB();
    static wxBitmap& AudioFile();
    static wxBitmap& Verify();
    static wxBitmap& Tap();
    static wxBitmap& Empty();
    static wxBitmap& Speaker32();
    static wxBitmap& Speaker16();
    static wxBitmap& Edit();

    // Colours
    static wxColour& Red();


  private:

    // Raw data
    static const char* addFileData[];
    static const char* addCDData[];
    static const char* addLineInData[];
    static const char* deleteData[];
    static const unsigned char filterData[];
    static const char* uncheckedBoxData[];
    static const unsigned char checkedBoxData[];
    static const char* refreshData[];
    static const unsigned char freeDBData[];
    static const char* audioFileData[];
    static const char* verifyData[];
    static const char* tapData[];
    static const char* emptyData[];
    static const char* speaker32Data[];
    static const char* speaker16Data[];
    static const char* editData[];

    // Ready-made bitmaps
    static wxBitmap addFileBM;
    static wxBitmap addCDBM;
    static wxBitmap addLineInBM;
    static wxBitmap deleteBM;
    static wxBitmap filterBM;
    static wxBitmap uncheckedBoxBM;
    static wxBitmap checkedBoxBM;
    static wxBitmap refreshBM;
    static wxBitmap freeDBBM;
    static wxBitmap audioFileBM;
    static wxBitmap verifyBM;
    static wxBitmap tapBM;
    static wxBitmap emptyBM;
    static wxBitmap speaker32BM;
    static wxBitmap speaker16BM;
    static wxBitmap editBM;

    // Colours
    static wxColour red;

};

#endif
```

```
/*****************************************************************************

  Icons.cpp

  Contains XPM icons and raw bitmap data accessed through static functions.

  Author: Erik Jälevik

*****************************************************************************/

#include "Icons.h"
#include <wx/image.h>

/*****************************************************************************
  Static data
*****************************************************************************/

const char* Icons::audioFileData[] = {
  /* columns rows colors chars-per-pixel */
  "16 15 4 1",
  "   c None",
  "a c Black",
  "b c #C0C0C0",
  "d c #FFFFFF",
  /* pixels */
  " aaaaaaaaaa     ",
  " addddddddaa    ",
  " addddddddbada  ",
  " addddddddbaaaa ",
  " addddddddbddda ",
  " addddddddddda ",
  " adddddaaadddba ",
  " adddddadadddba ",
  " adddddadadddba ",
  " addaaaadaaadba ",
  " addddddddddda ",
  " addddddddddda ",
  " addddddddddda ",
  " adbbbbbbbbbba ",
  " aaaaaaaaaaaaa "
};

const char* Icons::addFileData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 7 1",
  "   c None",
  "a c Black",
  "b c #E8D1A0",
  "c c #ACACAC",
  "e c #CCD4D5",
  "f c #FFFFFF",
  "g c #F4E8D0",
  /* pixels */
  "                ",
  "     aaaaaaa    ",
  "     affffaa    ",
  "    aaaffffafa  ",
  "    agaffffaaaa ",
  "  aaabaaaffffa  ",
  " agbgggaccccfa  ",
  " aaagaaaffffa   ",
  "    agaccccccfa ",
  "    aaaffffffa  ",
  "     afccccccfa ",
  "     affffffffa ",
  "     affffffffae",
  "     aaaaaaaaaee",
  "                ",
  "                "
};

const char* Icons::addCDData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 9 1",
  "   c None",
  "a c Black",
  "b c #E8D1A0",
  "c c #535353",
  "e c #CCD4D5",
  "f c #F4E8D0",
  "g c #62DAF9",
  "h c #ABF9C5",
  "i c #F2F5F9",
  /* pixels */
  "                ",
  "                ",
  "    aaa         ",
  "    afa ccc     ",
  "  aaabaaahicc   ",
  "  afbfffahiiic  ",
  "  aaafaaahiiic  ",
  "    afaggciiiic ",
  "    aaaic ciiic ",
  "    ciiiiciiiic ",
  "     ciiiiiiic  ",
  "     ciiiiiiic  ",
  "      cciiiccee ",
  "        ccceeeee",
  "                ",
  "                "
};

const char* Icons::addLineInData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 9 1",
  "   c None",
  "a c Black",
  "b c #E8D1A0",
  "c c #535353",
  "d c #ABAB01",
  "e c #FDFD01",
  "g c #FFFFFF",
  "h c #F4E8D0",
  "i c #464646",
  /* pixels */
  "                ",
  "    aaa         ",
  "    aha         ",
  "  aaabaaa    aa ",
  "  ahbhhha   aea ",
  "  aaahaaa  aea  ",
  "    aha  aaea   ",
  "    aaa adea    ",
  "       adega    ",
  "       aega     ",
  "      iaaa      ",
  "      ia   ac   ",
  "      i  aa     ",
  "       aa       ",
  "                ",
  "                "
};

const char* Icons::deleteData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 5 1",
```

```
  "  c None",
  "a c Black",
  "b c #FF0000",
  "c c #FF8585",
  "e c #CCD4D5",
  /* pixels */
  "                ",
  "                ",
  "    a     a     ",
  "   aca   aca    ",
  "  acbba acbba   ",
  "   abbbacbba    ",
  "    abbbbbba    ",
  "     abbba      ",
  "    acbbbba     ",
  "   acbbabbba    ",
  "  acbba abbba   ",
  "   aba   aba    ",
  "    aee   aee   ",
  "                ",
  "                ",
  "                "
};

const char* Icons::uncheckedBoxData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 20 1",
  "  c None",
  "a c #F1F1EF",
  "b c #DCDCD7",
  "c c #F9F9F8",
  "d c #EFEFEC",
  "e c #E5E5E2",
  "f c #7898B5",
  "g c #F7F7F6",
  "i c #ECECE9",
  "j c #E2E2DE",
  "k c #FFFFFF",
  "l c #F5F5F4",
  "m c #FEFEFE",
  "n c #E0E0DB",
  "o c #FDFDFC",
  "p c #F3F3F1",
  "q c #B0C2D3",
  "r c #E8E8E5",
  "s c #DEDED9",
  "t c #FBFBFA",
  /* pixels */
  "                ",
  "                ",
  " qfffffffffffq  ",
  " fbbbsnjeridaf  ",
  " fbbsnjeridapf  ",
  " fbsnjeridaplf  ",
  " fsnjeridaplgf  ",
  " fnjeridaplgcf  ",
  " fjeridaplgctf  ",
  " feridaplgctof  ",
  " fridaplgctomf  ",
  " fidaplgctomkf  ",
  " fdaplgctomkkf  ",
  " faplgctomkkkf  ",
  " qfffffffffffq  ",
  "                "
};

const char* Icons::refreshData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 2 1",
  "  c None",
  "a c #21A121",
  /* pixels */
  "                ",
  "                ",
  "        a       ",
  "        aa      ",
  "     aaaaaa     ",
  "    aa  aa      ",
  "    a   a       ",
  "    a      a    ",
  "    a      a    ",
  "      a   a     ",
  "     aa  aa     ",
  "     aaaaaa     ",
  "      aa        ",
  "       a        ",
  "                ",
  "                "
};

const unsigned char Icons::filterData[] = {
  236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,0,0,0,0,0,
0,0,
  0,0,0,0,0,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,
236,233,
  216,236,233,216,236,233,216,0,0,0,0,0,0,0,0,0,241,233,213,238,229,204,235,223,194,
  231,218,183,0,0,0,0,0,0,0,0,0,236,233,216,236,233,216,236,233,216,0,
  0,0,0,0,0,244,238,222,227,227,227,242,235,216,213,213,213,236,225,198,195,195,195,
229,215,
  176,177,177,177,223,205,157,162,162,162,0,0,0,0,0,0,236,233,216,0,0,0,241,233,212,
  243,237,219,245,239,225,243,237,219,241,233,212,238,228,202,234,222,192,231,217,18
1,227,212,171,224,
  206,161,221,203,153,219,199,146,219,199,145,219,199,145,0,0,0,0,0,0,242,235,216,22
7,227,
  227,244,238,222,220,220,220,239,230,207,203,203,203,232,220,186,184,184,184,226,20
9,165,167,167,167,
  220,200,149,158,158,158,219,199,145,158,158,158,0,0,0,236,233,216,0,0,0,245,239,22
4,243,
  236,218,241,232,210,237,227,201,234,222,191,230,216,179,227,211,170,223,206,159,22
1,202,152,219,199,
  146,219,199,145,219,199,145,0,0,0,236,233,216,236,233,216,236,233,216,0,0,0,242,23
4,214,
  211,211,211,235,224,195,192,192,192,228,213,174,174,174,174,222,204,155,160,160,16
1,219,199,145,158,
  158,158,0,0,0,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,0,0,0,23
7,226,
  199,233,221,189,230,216,178,226,210,168,223,205,159,221,201,151,219,199,145,219,19
9,145,0,0,0,
  236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,21
6,0,0,0,231,
  218,183,182,182,182,225,208,162,166,166,166,219,200,147,158,158,158,0,0,0,236,233,
216,236,233,
  216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,23
3,216,0,0,0,
  226,209,166,223,205,157,221,201,150,219,199,145,0,0,0,236,233,216,236,233,216,236,
233,216,236,
  233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,23
6,233,216,0,0,
  0,0,0,0,0,0,0,0,0,0,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,
  236,233,216
};

const unsigned char Icons::checkedBoxData[] = {
  236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,21
6,236,233,216,236,
```

```
  233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,23
6,233,216,236,233,
  216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,23
3,216,236,233,216,
  236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,21
6,236,233,216,176,
  194,211,120,152,181,120,152,181,120,152,181,120,152,181,120,152,181,120,152,181,12
0,152,181,120,152,
  181,120,152,181,120,152,181,120,152,181,176,194,211,236,233,216,236,233,216,236,23
3,216,120,152,181,
  220,220,215,220,220,215,220,220,215,222,222,217,224,224,219,226,226,222,229,229,22
6,232,232,229,236,
  236,233,239,239,236,241,241,239,120,152,181,236,233,216,236,233,216,236,233,216,12
0,152,181,220,220,
  215,220,220,215,222,222,217,224,224,219,226,226,222,229,229,226,232,232,229,236,23
6,233,239,239,236,
  241,241,239,243,243,241,120,152,181,236,233,216,236,233,216,236,233,216,120,152,18
1,221,221,216,223,
  223,218,225,225,220,226,226,222,229,229,226,232,232,229,236,236,233,239,239,236,0,
0,0,243,243,
  241,245,245,244,120,152,181,236,233,216,236,233,216,236,233,216,120,152,181,224,22
4,219,225,225,220,
  227,227,224,229,229,226,232,232,229,236,236,233,239,239,236,0,0,0,0,0,0,245,245,24
4,247,
  247,245,120,152,181,236,233,216,236,233,216,236,233,216,120,152,181,226,226,221,22
6,226,222,0,0,
  0,232,232,229,236,236,233,239,239,236,0,0,0,0,0,0,0,0,247,247,245,249,248,246,
  120,152,181,236,233,216,236,233,216,236,233,216,120,152,181,227,227,224,229,229,22
6,0,0,0,0,
  0,0,239,239,236,0,0,0,0,0,0,0,0,247,247,246,248,248,246,250,250,249,120,152,
  181,236,233,216,236,233,216,236,233,216,120,152,181,229,229,226,232,232,229,0,0,0,
0,0,0,
  0,0,0,0,0,0,0,0,0,247,246,245,248,248,247,250,250,249,253,253,252,120,152,181,236,
  233,216,236,233,216,236,233,216,120,152,181,232,232,229,236,236,233,239,239,235,0,
0,0,0,0,
  0,0,0,0,246,246,245,247,247,245,250,250,249,253,253,252,254,254,254,120,152,181,23
6,233,216,
  236,233,216,236,233,216,120,152,181,236,236,233,239,239,236,240,238,236,241,241,23
8,0,0,0,245,
  245,243,246,245,244,250,250,248,253,253,252,254,254,254,255,255,255,120,152,181,23
6,233,216,236,233,
  216,236,233,216,120,152,181,239,239,236,241,241,239,242,241,239,242,241,239,245,24
5,243,245,245,242,
  249,248,247,253,253,252,254,254,254,255,255,255,255,255,255,120,152,181,236,233,21
6,236,233,216,236,
  233,216,120,152,181,241,241,239,243,243,241,245,245,244,245,245,243,247,247,245,24
9,249,247,253,253,
  252,254,254,254,255,255,255,255,255,255,255,255,255,120,152,181,236,233,216,236,23
3,216,236,233,216,
  176,194,211,120,152,181,120,152,181,120,152,181,120,152,181,120,152,181,120,152,18
1,120,152,181,120,
  152,181,120,152,181,120,152,181,120,152,181,176,194,211,236,233,216,236,233,216,23
6,233,216,236,233,
  216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,23
3,216,236,233,216,
  236,233,216,236,233,216,236,233,216,236,233,216,236,233,216,236,233,216
};

const unsigned char Icons::freeDBData[] = {
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,219,219,233,0,
0,89,11,11,
  110,17,17,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,11
2,
  16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16
,
  16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,255,255,255,255,255,255,2
55,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,219,219,233,0,0,89,11,11,110,17,17,112,16,16,1
12,16,
  16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16
,
  112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,11
2,
  16,16,112,16,16,112,16,16,112,255,255,255,255,255,255,255,255,255,255,255,255,250,
250,252,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,219,219,233,0,0,89,11,11,110,17,17,112,16,16,112,16,16,112,16,16,112,16,16,112
,
  16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16
,
  16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16
,
  112,255,255,255,255,255,255,251,251,239,15,15,95,0,0,16,20,20,115,255,255,255,255,
255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,129,129,173,0,0,80,255,255,255
,255,255,255,
  0,0,85,82,82,136,255,255,255,255,255,255,255,255,255,255,255,255,219,219,233,0,0,8
9,11,
  11,110,17,17,112,16,16,112,17,17,112,19,19,114,19,19,118,19,19,118,18,18,113,17,17
,
  112,16,16,112,16,16,112,18,18,117,18,18,113,18,18,113,19,19,118,16,16,112,18,18,11
3,
  19,19,118,18,18,117,18,18,113,19,19,118,18,18,117,16,16,111,255,255,255,255,255,25
5,104,
  104,151,0,0,43,183,183,220,231,231,241,255,255,255,255,255,255,255,255,255,255,255
,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,127,127,143,0,0,48,255,255,255,255,255,255,0,0,51,38,38,124,25
5,255,
  255,255,255,255,255,255,255,255,255,255,217,217,233,0,0,89,11,11,110,17,17,112,17,
17,112,
  17,17,115,0,0,94,0,0,67,0,0,71,0,0,97,13,13,112,21,21,113,9,9,111,0,
  0,77,0,0,99,0,0,83,0,0,67,21,21,117,0,0,105,0,0,69,0,0,73,0,0,
  105,0,0,71,0,0,85,19,19,118,255,255,255,219,219,235,45,45,129,0,0,87,177,177,209,
```

```
  255,255,255,195,195,220,210,210,231,235,235,237,77,77,110,214,214,226,255,255,255,
242,242,242,90,
  90,131,104,104,148,245,245,253,255,255,255,255,255,255,255,255,255,194,194,225,102
,102,141,142,142,
  182,255,255,255,255,255,255,255,255,255,255,255,255,177,177,209,88,88,122,200,200,
221,156,156,177,
  0,0,61,255,255,255,255,255,255,0,0,69,87,87,150,230,230,248,102,102,141,169,169,19
4,255,
  255,255,227,227,241,0,0,89,11,11,110,17,17,115,15,15,116,0,0,67,13,13,112,169,169,
  194,149,149,182,72,72,145,0,0,83,0,0,105,39,39,126,122,122,172,0,0,105,170,170,180
,
  157,157,189,0,0,74,31,31,111,132,132,177,208,208,209,7,7,113,101,101,159,71,71,144
,0,
  0,99,255,255,255,66,66,142,0,0,61,11,11,117,0,0,38,177,177,194,78,78,135,0,0,
  74,0,0,97,0,0,22,238,238,241,244,244,237,0,0,51,0,0,80,0,0,67,0,0,65,
  217,217,233,255,255,255,97,97,164,0,0,26,5,5,84,0,0,44,0,0,71,255,255,255,255,
  255,255,113,113,166,0,0,43,0,0,71,0,0,77,1,1,100,0,0,93,255,255,255,255,255,
  255,0,0,80,22,22,114,0,0,74,0,0,67,0,0,31,127,127,167,255,255,255,0,0,94,
  11,11,110,16,16,109,0,0,87,113,113,175,255,255,255,255,255,255,255,255,255,255,255
,255,133,
  133,159,0,0,55,49,49,135,255,255,255,255,255,255,250,250,254,255,255,255,0,0,87,25
5,255,
  255,255,255,255,234,234,246,255,255,255,255,255,255,243,243,252,0,0,69,255,255,255
,255,255,255,
  72,72,145,0,0,65,214,214,235,255,255,255,96,96,162,0,0,43,148,148,174,255,255,255,
255,
  255,255,0,0,93,0,0,83,255,255,255,255,255,255,0,0,94,25,25,117,255,255,255,0,0,
  47,134,134,182,255,255,255,255,255,255,0,0,55,124,124,154,255,255,255,0,0,67,26,26
,119,
  255,255,255,255,255,255,0,0,71,0,0,83,255,255,255,255,255,255,0,0,74,0,0,78,255,
  255,255,255,255,255,26,26,120,0,0,69,222,222,230,2,2,104,13,13,111,0,0,99,8,8,
  76,255,255,255,247,247,255,0,0,69,0,0,62,221,221,222,255,255,255,0,0,83,18,18,117,
  255,255,255,255,255,255,0,0,103,0,0,65,113,113,175,255,255,255,107,107,168,0,0,26,
127,
  127,153,255,255,255,201,201,228,0,0,74,255,255,255,255,255,255,101,101,160,0,0,55,
255,255,
  255,255,255,255,77,77,137,0,0,57,255,255,255,255,255,255,250,250,254,0,0,57,41,41,
124,
  90,90,141,116,116,148,0,0,80,0,0,74,234,234,215,0,0,77,12,12,109,110,110,141,103,
  103,150,0,0,77,16,16,104,236,236,250,0,0,26,170,170,180,255,255,255,255,255,255,15
8,158,
  166,0,0,59,255,255,255,255,255,255,0,0,67,66,66,137,255,255,255,255,255,255,156,15
6,177,
  0,0,77,205,205,229,7,7,107,15,15,116,0,0,85,89,89,157,255,255,255,118,118,141,0,
  0,62,0,0,69,105,105,165,255,255,255,75,75,149,0,0,87,255,255,255,167,167,201,0,0,
  67,0,0,59,213,213,211,255,255,255,0,0,97,0,0,74,0,0,87,255,255,255,210,210,231,
  0,0,73,255,255,255,255,255,255,84,84,150,0,0,57,234,234,246,255,255,255,77,77,137,
0,
  0,59,255,255,255,255,255,255,243,243,247,0,0,59,23,23,111,117,117,178,106,106,173,
132,132,
  185,179,179,204,211,211,201,0,0,71,80,80,158,122,122,180,99,99,169,142,142,182,200
,200,221,
  236,236,250,0,0,48,115,115,159,255,255,255,255,255,255,111,111,155,0,0,65,255,255,
255,255,
  255,255,0,0,65,77,77,145,255,255,255,255,255,255,177,177,194,0,0,71,214,214,235,0,
0,
  93,0,0,73,0,0,77,96,96,158,255,255,255,135,135,166,0,0,57,0,0,71,93,93,159,
  255,255,255,87,87,152,0,0,85,255,255,255,180,180,209,0,0,89,0,0,69,218,218,209,255
,
  255,255,0,0,87,0,0,38,28,28,99,255,255,255,208,208,231,0,0,73,255,255,255,255,255,
  255,88,88,152,0,0,57,236,236,248,255,255,255,81,81,136,0,0,52,255,255,255,255,255,
255,
  255,255,255,31,31,92,0,0,80,255,255,255,255,255,255,13,13,102,137,137,188,255,255,
255,0,
  0,41,63,63,139,255,255,255,255,255,252,0,0,73,184,184,215,255,255,255,0,0,77,0,0,
  69,255,255,255,245,245,255,0,0,69,0,0,78,255,255,255,255,255,255,0,0,80,0,0,74,
  221,221,222,255,255,255,0,0,78,0,0,74,194,194,225,17,17,102,213,213,211,74,74,153,
0,
  0,62,255,255,255,255,255,255,0,0,87,16,16,106,255,255,255,255,255,255,0,0,93,18,18
,
  113,255,255,255,179,179,208,0,0,83,0,0,89,142,142,184,255,255,255,183,183,211,7,7,
103,
  225,225,231,255,255,255,199,199,226,0,0,74,255,255,255,255,255,255,46,46,127,0,0,2
5,223,
  223,238,255,255,255,31,31,111,0,0,43,255,255,255,255,255,255,255,255,255,215,215,2
37,0,0,
  69,0,0,43,0,0,51,0,0,48,167,167,203,255,255,255,167,167,201,0,0,38,0,0,51,
  0,0,38,0,0,90,248,248,255,255,255,255,168,168,208,0,0,57,0,0,41,0,0,74,25,
  25,113,0,0,44,255,255,255,255,255,255,0,0,48,12,12,95,0,0,78,0,0,43,0,0,
  32,206,206,226,221,221,234,41,41,129,255,255,255,234,234,215,0,0,38,69,69,141,255,
255,255,
  255,255,255,255,255,255,255,255,255,91,91,151,0,0,47,58,58,140,255,255,255,223,223
,232,0,
  0,83,9,9,111,0,0,69,143,143,172,255,255,255,255,255,255,202,202,221,255,255,255,21
7,217,
  237,0,0,73,255,255,255,255,255,255,200,200,221,152,152,196,248,248,252,255,255,255
,191,191,218,
  181,181,211,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,132,132,18
0,120,120,169,251,
  251,247,255,255,255,255,255,255,255,255,255,199,199,226,127,127,171,138,138,184,25
5,255,255,255,255,
  255,255,255,255,255,255,255,226,226,238,100,100,157,235,235,237,255,255,255,148,14
8,189,255,255,255,
  255,255,255,152,152,196,255,255,250,244,244,240,111,111,163,181,181,211,255,255,25
5,210,210,228,0,
  0,90,110,110,170,60,60,130,4,4,105,0,0,87,19,19,102,132,132,185,180,180,209,47,47,
  133,0,0,80,4,4,105,24,24,120,102,102,162,64,64,141,0,0,103,7,7,110,80,80,142,
  20,20,99,74,74,156,15,15,95,0,0,55,255,255,255,191,191,218,0,0,77,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,223,223,234,0,0,85,0,0,80,0,0,
99,
  21,21,117,20,20,115,0,0,99,0,0,85,0,0,90,0,0,90,17,17,115,21,21,117,7,
  7,107,0,0,85,0,0,94,21,21,113,0,0,90,148,148,189,255,255,255,255,255,255,11,11,
  115,208,208,231,255,255,255,101,101,142,0,0,80,255,255,255,255,255,255,255,255,255
,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,219,219,233,0,0,90,14,14,113,19,19,114,16,16,112,16,16,112,19,
19,
  114,17,17,112,13,13,111,20,20,115,16,16,112,16,16,112,17,17,112,20,20,115,19,19,11
4,
  17,17,112,4,4,108,0,0,94,121,121,166,227,227,241,255,255,255,255,255,255,111,111,1
73,0,
  0,90,10,10,106,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255
,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,255,255,255,
  255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,25
5,255,219,219,233,
```

```
  0,0,89,11,11,110,17,17,112,16,16,112,16,16,112,16,16,112,16,16,112,17,17,112,16,
  16,112,16,16,112,16,16,112,16,16,112,16,16,112,16,16,112,21,21,113,13,13
,
  111,0,0,87,0,0,73,8,8,81,0,0,73,0,0,83,12,12,114,19,19,114
};

const char* Icons::verifyData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 2 1",
  "  c None",
  "a c #21A121",
  /* pixels */
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "            a   ",
  "           aa   ",
  "    a     aaa   ",
  "    aa   aaa    ",
  "    aaa aaa     ",
  "     aaaaa      ",
  "      aaa       ",
  "       a        ",
  "                ",
  "                ",
  "                "
};

const char* Icons::tapData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 17 1",
  "  c None",
  "a c Black",
  "b c #131313",
  "c c #00AEEF",
  "d c #E8D1A0",
  "e c #1B1B1B",
  "f c #070707",
  "g c #101010",
  "h c #232323",
  "i c #060606",
  "k c #030303",
  "l c #161616",
  "m c #0C0C0C",
  "n c #020202",
  "o c #151515",
  "p c #010101",
  "q c #0A0A0A",
  /* pixels */
  "                ",
  "   mmmmm        ",
  "   mcccm        ",
  "   mcacm        ",
  "   madam        ",
  "   madam        ",
  "    mdaa a a    ",
  "  aaidfdadada   ",
  "  admdgdidpdb   ",
  "  adadldodndk   ",
  "  adhddddddq    ",
  "  adaddddddda   ",
  "   aqddddddn    ",
  "    fddddddda   ",
  "     aaqefia    ",
  "                "
};

const char* Icons::emptyData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 1 1",
  "  c None",
  /* pixels */
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                ",
  "                "
};

const char* Icons::speaker32Data[] = {
  /* columns rows colors chars-per-pixel */
  "32 32 11 1",
  "  c None",
  "a c Black",
  "b c #DDDDDD",
  "c c #555555",
  "d c #AAAAAA",
  "e c #222222",
  "g c #FFFFFF",
  "h c #BBBBBB",
  "i c #777777",
  "j c #888888",
  "k c #444444",
  /* pixels */
  "                aaa             ",
  "              aagggaa           ",
  "           aaggbbbbbaa          ",
  "         aaggbbbbbbbbbaa        ",
  "        aggbbbbbbbbbbbbaa       ",
  "      ahhhbbbbbbbbbbbbbbba      ",
  "      ahbhhhbbbbbbbbbbbjjka     ",
  "      ahbddhhhbbbbbbbjjkkka     ",
  "      ahcddddhhhbbbjjkkkkka     ",
  "      accddddddhdjjkkkkkkka     ",
  "      ahbdeeedddjickkkkkkka     ",
  "      ahbdecccddcickkkkkkka     ",
  "      ahbdeheadccickkkkkkka     ",
  "      ahbdeeeaddjickkkkkkka     ",
  "      ahbddcaaddjickkkkkkka     ",
  "      ahbddddddjickkkkkkka      ",
  "      ahbeeeadddjickkkkkkka     ",
  "      aheecchaddjickkkkkkka     ",
  "      ahecddihadjickkkkkkka     ",
  "      ahecdhehadjickkkkkkka     ",
  "      ahecjkehadjickkkkkkka     ",
  "      ahecceehadjickkkkkkka     ",
  "      ahcajhhhadjickkkkkkka     ",
  "      accdcccaddjickkkkkkkac    ",
  "       abddddddjickkkkkkkakc    ",
  "        aaddddddjickkkkkkakkc   ",
  "          aaddddcickkkkaakkcc   ",
  "            aadccickkaakkcc     ",
  "              aajicaakkcc       ",
```

```
  "                aaakkcc         ",
  "                                ",
  "                                "
};

const char* Icons::speaker16Data[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 10 1",
  "   c None",
  "a c Black",
  "b c #DDDDDD",
  "c c #555555",
  "d c #AAAAAA",
  "e c #222222",
  "g c #BBBBBB",
  "h c #777777",
  "i c #888888",
  "j c #444444",
  /* pixels */
  "      aaaa      ",
  "    aabbbbaa    ",
  "   agbbbbbbba   ",
  "   addgbbbbjja  ",
  "   acdddijjjja  ",
  "   adeedcjjjja  ",
  "   adeaccjjjja  ",
  "   addddhjjjja  ",
  "   aeeadhjjjja  ",
  "   aedeahjjjja  ",
  "   aeeeahjjjja  ",
  "   accaahjjjjai ",
  "   aadddhjjjjac ",
  "     aaccjjaajc ",
  "       aaaajc   ",
  "                "
};

const char* Icons::editData[] = {
  /* columns rows colors chars-per-pixel */
  "16 16 8 1",
  "   c None",
  "a c Black",
  "b c #ACACAC",
  "c c #9DBE7D",
  "d c #B6D698",
  "e c #CCD4D5",
  "g c #FFFFFF",
  "h c #9AAD4C",
  /* pixels */
  "                ",
  "    aaaaaaaa    ",
  "   adagggggaa   ",
  "  ahcdaggggaga  ",
  "   ahcdagggaaaa ",
  "    ahcdaggggga ",
  "     ahcdabbbga ",
  "     aahcdaggga ",
  "     agahcabbga ",
  "     aggaaaggga ",
  "     agbbbbbbga ",
  "     agggggggga ",
  "     agggggggae ",
  "     aaaaaaaaaee",
  "                ",
  "                "
};

wxBitmap Icons::addFileBM;
wxBitmap Icons::addCDBM;
wxBitmap Icons::addLineInBM;
wxBitmap Icons::deleteBM;
wxBitmap Icons::filterBM;
wxBitmap Icons::uncheckedBoxBM;
wxBitmap Icons::checkedBoxBM;
wxBitmap Icons::refreshBM;
wxBitmap Icons::freeDBBM;
wxBitmap Icons::audioFileBM;
wxBitmap Icons::verifyBM;
wxBitmap Icons::tapBM;
wxBitmap Icons::emptyBM;
wxBitmap Icons::speaker32BM;
wxBitmap Icons::speaker16BM;
wxBitmap Icons::editBM;

wxColour Icons::red(160, 40, 40);

/*******************************************************************************
  Access methods

  All of these create the static wxBitmaps in request and return a reference
  to it from then on.
*******************************************************************************/

wxBitmap& Icons::AddFile() {

  if (!addFileBM.Ok()) {
    addFileBM = wxBitmap(addFileData);
  }
  return addFileBM;

}

wxBitmap& Icons::AddCD() {

  if (!addCDBM.Ok()) {
    addCDBM = wxBitmap(addCDData);
  }
  return addCDBM;

}

wxBitmap& Icons::AddLineIn() {

  if (!addLineInBM.Ok()) {
    addLineInBM = wxBitmap(addLineInData);
  }
  return addLineInBM;

}

wxBitmap& Icons::Delete() {

  if (!deleteBM.Ok()) {
    deleteBM = wxBitmap(deleteData);
  }
  return deleteBM;

}

wxBitmap& Icons::Filter() {

  if (!filterBM.Ok()) {
    wxImage image(16, 11, const_cast<unsigned char*>(filterData), TRUE);
    image.SetMaskColour(236, 233, 216);
    filterBM = wxBitmap(image);
  }
```

```
  return filterBM;

}

wxBitmap& Icons::UncheckedBox() {

  if (!uncheckedBoxBM.Ok()) {
    uncheckedBoxBM = wxBitmap(uncheckedBoxData);
  }
  return uncheckedBoxBM;

}

wxBitmap& Icons::CheckedBox() {

  if (!checkedBoxBM.Ok()) {
    wxImage image(16, 16, const_cast<unsigned char*>(checkedBoxData), TRUE);
    image.SetMaskColour(236, 233, 216);
    checkedBoxBM = wxBitmap(image);
  }
  return checkedBoxBM;

}

wxBitmap& Icons::Refresh() {

  if (!refreshBM.Ok()) {
    refreshBM = wxBitmap(refreshData);
  }
  return refreshBM;

}

wxBitmap& Icons::FreeDB() {

  if (!freeDBBM.Ok()) {
    wxImage image(64, 16, const_cast<unsigned char*>(freeDBData), TRUE);
    freeDBBM = wxBitmap(image);
  }
  return freeDBBM;

}

wxBitmap& Icons::AudioFile() {

  if (!audioFileBM.Ok()) {
    audioFileBM = wxBitmap(audioFileData);
  }
  return audioFileBM;

}

wxBitmap& Icons::Verify() {

  if (!verifyBM.Ok()) {
    verifyBM = wxBitmap(verifyData);
  }
  return verifyBM;

}

wxBitmap& Icons::Tap() {

  if (!tapBM.Ok()) {
    tapBM = wxBitmap(tapData);
  }
  return tapBM;

}

wxBitmap& Icons::Empty() {

  if (!emptyBM.Ok()) {
    emptyBM = wxBitmap(emptyData);
  }
  return emptyBM;

}

wxBitmap& Icons::Speaker32() {

  if (!speaker32BM.Ok()) {
    speaker32BM = wxBitmap(speaker32Data);
  }
  return speaker32BM;

}

wxBitmap& Icons::Speaker16() {

  if (!speaker16BM.Ok()) {
    speaker16BM = wxBitmap(speaker16Data);
  }
  return speaker16BM;

}

wxBitmap& Icons::Edit() {

  if (!editBM.Ok()) {
    editBM = wxBitmap(editData);
  }
  return editBM;

}

wxColour& Icons::Red() {
  return red;
}
```

```
/**************************************************************************

  InStream.h

  Abstract base class for all audio input streams.

  Author: Erik Jälevik

**************************************************************************/

#ifndef INSTREAM_H
#define INSTREAM_H

#include "Globals.h"

#include <string>

class InStream {

  public:

    /**************************************************************************
      Destructor
    **************************************************************************/
    virtual ~InStream() { }

    /**************************************************************************
      Exception class used for anything that can go wrong when reading input.
    **************************************************************************/
    class InStreamException : public std::runtime_error {
      public:
        InStreamException(const std::string& msg) : std::runtime_error(msg) {}
    };

    /**************************************************************************
      Set the sample rate at which values should be returned by the stream.
    **************************************************************************/
    virtual void SetRate(uint rate) { readRate = rate; }

    /**************************************************************************
      Opens the named file.
    **************************************************************************/
    virtual void Open(const std::string& file) = 0; // pure virtual

    /**************************************************************************
      Closes the stream.
    **************************************************************************/
    virtual void Close() = 0; // pure virtual

    /**************************************************************************
      Returns true if there is more data to read from the stream.
    **************************************************************************/
    virtual bool HasMore() = 0; // pure virtual

    /**************************************************************************
      Read one sample. If the file is a stereo file, the average of the two
      channels is returned.
    **************************************************************************/
    virtual double Read() = 0; // pure virtual

    /**************************************************************************
      Returns length of stream in seconds if it's a file, otherwise 0.
    **************************************************************************/
    virtual int GetLength() = 0; // pure virtual

    /**************************************************************************
      Returns a number between 0 and 1 indicating how much of the stream
      has been read.
    **************************************************************************/
    virtual float GetProgress() = 0; // pure virtual

  protected:

    /**************************************************************************
      Constructor sets initial read rate to 44100 kHz. Use SetRate to change
      it.
    **************************************************************************/
    InStream::InStream() : readRate(44100) { }


    /**************************************************************************
      Variables
    **************************************************************************/
    uint readRate; // rate at which values should be returned,
                   // not rate of underlying audio

};

#endif
```

```
/*****************************************************************************

  LogDialog.h

  GUI dialog window for displaying a log of error messages after batch
  operations.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef LOGDIALOG_H
#define LOGDIALOG_H

class LogDialog : public wxDialog {

  public:

    /*************************************************************************
      Parameters are: parent window, window ID, caption string, message
      string and log string.
    *************************************************************************/
    LogDialog(wxWindow*, wxWindowID, const wxString&, const wxString&,
      const wxString&);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~LogDialog() { }

    /*************************************************************************
      Handler for OK button.
    *************************************************************************/
    void OnOK(wxCommandEvent&);


  private:

    /*************************************************************************
      Autogenerated method for building dialog.
    *************************************************************************/
    void BuildMe();


    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxStaticText* messageST;
    wxTextCtrl* logTC;

    wxButton* okBut;

    enum {
      ID_MESSAGE_ST,
      ID_LOG_TC
    };

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/*****************************************************************************

  LogDialog.cpp

  GUI list control object for displaying the music collection.

  Author: Erik Jälevik

*****************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include "wx/wx.h"
#endif

#include "LogDialog.h"
#include "Icons.h"

#include <wx/statline.h>
#include <wx/artprov.h>

/*****************************************************************************
  Constructor
*****************************************************************************/
LogDialog::LogDialog(wxWindow* parent,
                     wxWindowID id,
                     const wxString& caption,
                     const wxString& msg,
                     const wxString& log) :
        wxDialog(parent, id, caption, wxDefaultPosition, wxDefaultSize,
          wxCAPTION | wxSYSTEM_MENU | wxRESIZE_BORDER) {

  BuildMe();

  messageST->SetLabel(msg);
  logTC->SetValue(log);

}

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(LogDialog, wxDialog)
  EVT_BUTTON(wxID_OK, LogDialog::OnOK)
END_EVENT_TABLE()

/*****************************************************************************
  OnOK
*****************************************************************************/
void LogDialog::OnOK(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
          SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*****************************************************************************
```

```
  BuildMe
*****************************************************************************/
void LogDialog::BuildMe() {

  wxBoxSizer *item1 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item2 = new wxBoxSizer( wxHORIZONTAL );

  // Ugly, ugly, ugly! Icon returned by wxArtProvider is not anti-aliased.
  wxStaticBitmap *item3 = new wxStaticBitmap(this, -1,
    wxArtProvider::GetBitmap(wxART_ERROR, wxART_OTHER, wxSize(32, 32)),
    wxDefaultPosition, wxDefaultSize);
  item2->Add( item3, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 5 );

  messageST = new wxStaticText(this, ID_MESSAGE_ST, "",
    wxDefaultPosition, wxSize(100, 28));
  item2->Add(messageST, 1, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxBoxSizer *item5 = new wxBoxSizer( wxVERTICAL );

  okBut = new wxButton(this, wxID_OK, "OK", wxDefaultPosition,
    wxDefaultSize, 0);
  item5->Add(okBut, 0, wxALIGN_CENTER|wxALL, 5);

  item2->Add( item5, 0, wxALIGN_CENTER|wxALL, 0 );

  item1->Add(item2, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  wxStaticLine *item7 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL);
  item1->Add(item7, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 5);

  logTC = new wxTextCtrl(this, ID_LOG_TC, "", wxDefaultPosition,
    wxSize(500, 180), wxTE_MULTILINE|wxTE_READONLY);
  item1->Add(logTC, 1, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  SetAutoLayout(TRUE);
  SetSizer(item1);
  item1->Fit(this);
  item1->SetSizeHints(this);

}
```

```
/*****************************************************************************

  MadWrapper.h

  A C++ wrapper around the MAD MP3 decoding library providing just the
  functions needed by FileInStream.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef MADWRAPPER_H
#define MADWRAPPER_H

#include "Globals.h"

#include "mad/mad.h"

#include <vector>
#include <fstream> // for reading from file

class MadWrapper {

  public:

    /*************************************************************************
      Constructor
    *************************************************************************/
    MadWrapper();

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~MadWrapper() { }

    /*************************************************************************
      Opens the named file.
    *************************************************************************/
    virtual void Open(const std::string&);

    /*************************************************************************
      Closes the stream opened by a call to Open. Must be called before
      Open is called again.
    *************************************************************************/
    virtual void Close();

    /*************************************************************************
      Has all data been read?
    *************************************************************************/
    bool HasMore() { return !finished; }

    /*************************************************************************
      Is a file open?
    *************************************************************************/
    bool IsOpen() { return open; }

    /*************************************************************************
      Read a sample from MP3.
    *************************************************************************/
    double Read();

    /*************************************************************************
      Read len number of samples into the array of doubles.
    *************************************************************************/
    uint Read(double*, uint);

    /*************************************************************************
      Set the sample rate at which values should be returned.
```

```
  ***************************************************************************/
  virtual void SetRate(uint r) { readRate = r; }

  /***************************************************************************
    Returns length of stream in seconds.
  ***************************************************************************/
  virtual uint GetLength() { return length; }

  /***************************************************************************
    Returns a number between 0 and 1 indicating how much of the file
    has been read.
  ***************************************************************************/
  virtual float GetProgress();


private:

  /***************************************************************************
    Calculates the length of the MP3 based on its bitrate and size in kB.
    Returns true if successful. This return value can be taken to indicate
    whether the file is a valid MP3 or not.
  ***************************************************************************/
  bool CalculateLength();

  /***************************************************************************
    Starts decoding process to fill PCM buffers.
  ***************************************************************************/
  void Decode();

  /***************************************************************************
    Read more encoded MP3 input from disk and give it to MAD.
  ***************************************************************************/
  void FillInputBuffer();

  /***************************************************************************
    Converts a sample from MADs internal 24-bit format to a double in the
    range of -1.0 to 1.0.
  ***************************************************************************/
  double Scale(mad_fixed_t);

  /***************************************************************************
    For throwing exceptions.
  ***************************************************************************/
  void Error(const std::string&);


  /***************************************************************************
    Variables
  ***************************************************************************/
  static const uint inputBufSizeKb = 500; // encoded file buffer size (kB)
    // The absolute theoretical maximum MP3 frame size is 2881 bytes:
    // MPEG 2.5 Layer II, 8000 Hz @ 160 kbps, with padding

  static const uint outputBufSizeKb = 500; //  decoded PCM buffer size (kB)

  mad_stream stream; // MAD
  mad_frame frame;   // decoder
  mad_synth synth;   // structs

  char inputBuf[inputBufSizeKb * 1000 + MAD_BUFFER_GUARD]; // input buffer
    // Adding MAD_BUFFER_GUARD ensures that buffer can't overflow when the
    // buffer guard is added after reading the last piece of data.

  size_t inputBufLen; // length of input buffer

  std::vector<mad_pcm> outputBuf; // output buffer for PCM frames
  std::vector<mad_pcm>::size_type outputBufIdx; // index into outputBuf
  uint frameIdx; // index into current frame in outputBuf
  uint outputBufMax; // max allowable frames in outputBuf

  std::string filename;     // name of currently open MP3
  std::filebuf file;        // filebuf for reading in MP3
  std::streamsize fileSize; // length of MP3 in bytes
  uint length;              // length of MP3 in seconds

  uint readRate;  // the rate we should return values at
  float readStep; // number of samples to advance per call to Read
  float readCtr;  // read step counter

  ulong totalBytesRead; // keeps track of buffering to
  long decodedBytes;    // enable progress reporting

  uint frameCount;  // frames read

  uint avgBitRate;  // running average of bitrate (used for VBRs)
  ulong bitRateSum; // running sum of bitrate (used for VBRs)

  bool open;       // is a file open?
  bool allRead;    // has all audio has been read from MP3 file?
  bool allDecoded; // has all audio been decoded?
  bool finished;   // is there any more decoded audio to return?

};

#endif
```

```
/*****************************************************************************

  MadWrapper.cpp

  A C++ wrapper around the MAD MP3 decoding library providing just the
  functions needed by FileInStream.

  One MP3 frame contains 1152 PCM samples.

  Uses crude point sampling for up- and downsampling. The results don't
  sound too great but it shouldn't matter for beat tracking as the rhythmic
  structure is completely preserved.

  Author: Erik Jälevik

*****************************************************************************/

#include "MadWrapper.h"
#include "InStream.h"   // InstreamException

#include <sstream>      // ostringstream
#include <cmath>        // floorf

using namespace std;

/*****************************************************************************
  Constructor
*****************************************************************************/
MadWrapper::MadWrapper() :
  avgBitRate(0),
  bitRateSum(0),
  frameCount(0),
  length(0),
  inputBufLen(0),
  outputBufIdx(0),
  totalBytesRead(0),
  decodedBytes(0),
  frameIdx(0),
  readRate(44100),
  readStep(1.0),
  readCtr(0.0),
  allRead(true),
  allDecoded(true),
  finished(true),
  open(false) {

  // Reserve enough space in outputBuffer to hold the number of frames closest
  // to outputBufSizeKb.
  uint frameSize = sizeof(mad_pcm); // 9224 bytes
  outputBufMax = (outputBufSizeKb * 1000) / frameSize;
  outputBuf.reserve(outputBufMax);

  LOG("frames in MadWrapper output buffer: " << outputBufMax << endl);

}

/*****************************************************************************
  Open
*****************************************************************************/
void MadWrapper::Open(const string& fname) {

  filename = fname;

  // Open file
  if (file.open(filename.c_str(), ios_base::in | ios_base::binary) == 0) {
    string s = "File '" + filename + "' not found or could not be opened.";
    // Can't use Error because there is no MAD error string yet
    throw InStream::InStreamException(s);
  }

  allRead = false;
  allDecoded = false;
  finished = false;
  open = true;

  if (!CalculateLength()) {
    string s = "File '" + filename + "' doesn't appear to be a valid MP3 "
      "file.";
    // Can't use Error because there is no MAD error string yet
    throw InStream::InStreamException(s);
  }

  // Initialise MAD structures
  mad_stream_init(&stream);
  mad_frame_init(&frame);
  mad_synth_init(&synth);

  // Push an empty pcm frame into outputBuf so that the Read logic
  // works correctly, i.e. goes away and buffers on first call.
  mad_pcm dummy;
  dummy.length = 0;
  outputBuf.push_back(dummy);

}

/*****************************************************************************
  Close
*****************************************************************************/
void MadWrapper::Close() {

  // Reset buffers and variables
  inputBufLen = inputBufSizeKb * 1000;
  outputBuf.clear();
  outputBufIdx = 0;
  avgBitRate = 0;
  bitRateSum = 0;
  frameCount = 0;
  length = 0;
  frameIdx = 0;
  readCtr = 0.0;
  totalBytesRead = 0;
  decodedBytes = 0;
  open = false;

  // Close any previously read file
  file.close();

  // Clean up MAD structs
  mad_synth_finish(&synth);
  mad_frame_finish(&frame);
  mad_stream_finish(&stream);

}

/*****************************************************************************
  Read (one sample)
*****************************************************************************/
inline double MadWrapper::Read() {

  // Check if reached end of current frame
  if (frameIdx >= outputBuf[outputBufIdx].length) {

    ++outputBufIdx;

    // Check if reached end of output buffer.
    // Using while guards against the situation when Decode returns 0 frames
```

```
    // which can happen at the end of a file.
    while (outputBufIdx >= outputBuf.size()) {
      if (!allDecoded) {
        Decode();
      }
      else {
        finished = true;
        return 0.0;
      }
    }

    // Calculate read step for new frame
    uint freq = outputBuf[outputBufIdx].samplerate;
    readStep = static_cast<float>(freq) / readRate;

    //LOG("new frame, readCtr: " << readCtr <<
    //    ", readStep: " << readStep <<
    //    ", freq: " << freq <<
    //    ", length: " << outputBuf[outputBufIdx].length <<
    //    ", channels: " << outputBuf[outputBufIdx].channels << endl);

    // Reset readCtr preserving decimals
    readCtr = readCtr - floorf(readCtr);
    frameIdx = 0;

  }

  // Extract left channel sample
  double sample = Scale(outputBuf[outputBufIdx].samples[0][frameIdx]);

  // If stereo, take average of left and right channels
  if (outputBuf[outputBufIdx].channels == 2) {
    double right = Scale(outputBuf[outputBufIdx].samples[1][frameIdx]);
    sample = (sample + right) / 2;
  }

  readCtr += readStep;
  frameIdx = static_cast<uint>(readCtr);

  return sample;

}

/*****************************************************************************
  Read (many samples)
*****************************************************************************/
uint MadWrapper::Read(double* buffer, uint len) {

  uint i;
  for (i = 0; (i < len) && !finished; ++i) {
    buffer[i] = Read();
  }

  return i;

}

/*****************************************************************************
  GetProgress
*****************************************************************************/
float MadWrapper::GetProgress() {

  float ratio = (static_cast<float>(outputBufIdx) / outputBuf.size());
  ulong pos = totalBytesRead - (stream.bufend - stream.next_frame) -
    ((1 - ratio) * decodedBytes);

  //LOG("totalBytesRead: " << totalBytesRead <<
        ", left in stream: " << (stream.bufend - stream.next_frame) <<
  //    ", ratio: " << ratio << ", decoded chunk: " << decodedBytes <<
  //    ", fileSize: " << fileSize << endl);

  return static_cast<float>(pos) / fileSize;

}

/*****************************************************************************
  CalculateLength
*****************************************************************************/
bool MadWrapper::CalculateLength() {

  mad_frame_init(&frame);
  mad_stream_init(&stream);

  // Open file with flag ate (at end)
  ifstream in(filename.c_str(),
              ios_base::in | ios_base::ate | ios_base::binary);

  // Pos of pointer = file size
  fileSize = in.tellg();

  // Reset pointer to start of file
  in.seekg(0);

  // Get 25k of file
  streamsize nBytes = in.rdbuf()->sgetn(inputBuf, 25000);

  // Give buffer to MAD
  mad_stream_buffer(&stream,
                    reinterpret_cast<const unsigned char*>(inputBuf),
                    nBytes);

  // See if we can decode 3 frames
  for (int count = 0; count < 3; ++count) {
    while (mad_frame_decode(&frame, &stream) != 0){

      LOG("mad error: " << mad_stream_errorstr(&stream) << endl);
      //LOG("this_frame: " << stream.this_frame - stream.buffer << endl);
      //LOG("next_frame: " << stream.next_frame - stream.buffer << endl);

      // This means we've read all the buffer without finding a frame
      if (stream.error == MAD_ERROR_BUFLEN || stream.error == MAD_ERROR_BUFPTR) {
        LOG("end of buf, returning false: " << endl);
        return false;
      }

      stream.error = MAD_ERROR_NONE;
    }

    LOG("MadWrapper, got one frame, count: " << count << endl);
    avgBitRate += frame.header.bitrate;

  }

  avgBitRate /= 3;

  // OK, we got 3 frames. Also check that bitrate is not 0 which can happen
  // if it's a bogus frame.
  if (avgBitRate == 0) {
    return false;
  }

  // This will not be correct for VBR files, but avgBitRate will keep getting
  // adjusted as we progress further into the file and should stabilise after
  // a while.
  length = fileSize / (avgBitRate / 8);
```

```
  mad_frame_finish(&frame);
  mad_stream_finish(&stream);

  return true;

}

/*****************************************************************************
  Decode
*****************************************************************************/
void MadWrapper::Decode() {

  // Reset output buffer
  outputBuf.clear();
  outputBufIdx = 0;
  frameIdx = 0;

  // Save value of old next_frame for later use
  unsigned char const* nfB4 = stream.next_frame;

  // Keep on decoding until the output buffer has reached its desired size
  // or until all data has been decoded
  while (outputBuf.size() <= outputBufMax) {

    // Check if we need to fill input buffer
    if (stream.buffer == 0 || stream.error == MAD_ERROR_BUFLEN) {

      // If allRead is true here, it means all input has been read and decoded.
      if (allRead) {
        allDecoded = true;
        break;
      }

      FillInputBuffer();
      stream.error = MAD_ERROR_NONE;

    }

    // Decode next frame and check for errors
    if (mad_frame_decode(&frame, &stream) != 0) {

      // An error occurred
      LOG("MadWrapper decode error: " << mad_stream_errorstr(&stream) << endl);

      // If it's MAD_ERROR_BUFLEN, just need to fill buffer which will happen
      // next time round loop. If it's a recoverable error we also loop again.
      if (stream.error == MAD_ERROR_BUFLEN || MAD_RECOVERABLE(stream.error)) {
        continue;
      }
      else {
        // Stop decoding
        string s = "An unrecoverable error occurred while decoding the MP3 '" +
          filename + "'.";
        Error(s);
      }
    }

    // Update running average bitrate
    ++frameCount;
    bitRateSum += frame.header.bitrate;
    avgBitRate = bitRateSum / frameCount;

    // Synthesize frame
    mad_synth_frame(&synth, &frame);

    // Put PCM frame in output buffer
    outputBuf.push_back(synth.pcm);

  }

  // Update length
  length = fileSize / (avgBitRate / 8);

  // Used for progress reporting
  long lastDecodedBytes = decodedBytes;
  if (nfB4 == 0) { nfB4 = stream.buffer; } // to get correct value first time
  decodedBytes = stream.next_frame - nfB4;
  if (decodedBytes < 0) { decodedBytes = lastDecodedBytes; } // if wrapped

  //LOG("MadWrapper decoded " << decodedBytes << " bytes" << endl);

}

/*****************************************************************************
  FillInputBuffer
*****************************************************************************/
void MadWrapper::FillInputBuffer() {

  // Any data in inputBuf from address stream->next_frame will not have been
  // consumed and needs copying to the start of the new inputBuf.
  size_t remaining = 0;
  if (stream.next_frame != 0) {
    remaining = stream.bufend - stream.next_frame;
    memmove(inputBuf, stream.next_frame, remaining);

    //LOG("MadWrapper, remaining " << remaining <<
    //    " bytes moved to new inputbuf\n");

  }

  // Fill inputBuf from file.
  // Not sure how to check for failure or end of file here.
  inputBufLen = inputBufSizeKb * 1000;
  streamsize nBytes = file.sgetn(&inputBuf[remaining], inputBufLen - remaining);

  //LOG("MadWrapper read " << nBytes << " bytes from MP3" << endl);

  totalBytesRead += nBytes;

  // If the number of bytes read was less than that asked to read, we have
  // reached the end of the file. We need to pad the data with MAD_BUFFER_GUARD
  // zeros.
  if (nBytes < inputBufLen - remaining) {
    int endIdx = nBytes + remaining;
    memset(&inputBuf[endIdx], 0, MAD_BUFFER_GUARD);
    inputBufLen = endIdx + MAD_BUFFER_GUARD;
    allRead = true;
  }

  // Give MAD compressed MP3 data
  mad_stream_buffer(&stream,
                    reinterpret_cast<const unsigned char*>(inputBuf),
                    inputBufLen);

}

/*****************************************************************************
  Scale
*****************************************************************************/
double MadWrapper::Scale(mad_fixed_t sample) {

  // Round (can skip this)
  // sample += (1L << (MAD_F_FRACBITS - 16));

  // Clip
  if (sample >= MAD_F_ONE)
```

```
    sample = MAD_F_ONE - 1;
  else if (sample < -MAD_F_ONE)
    sample = -MAD_F_ONE;

  // Quantize
  sample = sample >> (MAD_F_FRACBITS + 1 - 16);

  // Convert to double
  return sample / 32768.0;

}

/*****************************************************************************
  Error
*****************************************************************************/
void MadWrapper::Error(const string& msg) {

  ostringstream tmp;
  tmp << msg;

  #ifdef _DEBUG
    tmp << "\n\nMAD error: " << mad_stream_errorstr(&stream);
  #endif

  throw InStream::InStreamException(tmp.str());

}
```

```
/*****************************************************************************

  MusicCollection.h

  Class representing a music collection. All interfacing with the database
  is done from this class.

  It's implemented as a Singleton which means it's constructor is private
  and an instance of the class can only be retrieved by calling the
  Instance method.

  It's also implementing the Observable interface which means clients
  can register as observers of the collection and be informed every time
  the collection changes.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef MUSICCOLLECTION_H
#define MUSICCOLLECTION_H

#include "MusicTrack.h"
#include "Observable.h"
#include "Globals.h"

#include "sqlite.h"

#include <string>
#include <sstream>  // sql ostringstream
#include <vector>
#include <memory>   // auto_ptr

class MusicCollection : public Observable {

  public:

    /*************************************************************************
      Exception class used for anything that can go wrong when reading or
      writing to the database.
    *************************************************************************/
    class DatabaseException : public std::runtime_error {
      public:
        DatabaseException(const std::string& msg) : std::runtime_error(msg) {}
    };

    /*************************************************************************
      Call Instance to get a pointer to the collection. If the collection
      has not yet been created, it will be created. If it has, a reference
      to the collection will be returned ensuring only one instance is ever
      created.

      A pointer is returned instead of a reference to enable other classes
      to hold member pointers to the collection. It's not possible to hold
      a reference as a member.
    *************************************************************************/
    static MusicCollection* Instance();

    /*************************************************************************
      Open the passed database file. Must be called at least once after the
      first call to Instance before any other methods can be called.
    *************************************************************************/
    void Open(const std::string&);

    /*************************************************************************
      Get the track corresponding to the passed index. Indices correspond to
      the position in the current sort order as displayed by the collection
      list control in the GUI. As the track is returned by reference the
```

```
  caller needs to make a copy if it needs it to persist past the next
  call to this function.
*****************************************************************************/
MusicTrack& GetTrack(uint index);

/*****************************************************************************
  Add a new track to the database.
*****************************************************************************/
void AddTrack(const MusicTrack&);

/*****************************************************************************
  Delete the track corresponding to the passed index from the database,
  or delete several tracks at once with the second variant.
*****************************************************************************/
void DeleteTrack(uint index);
void DeleteTracks(std::vector<uint>& indices);

/*****************************************************************************
  Permanently update passed in track with info stored in it.
*****************************************************************************/
void UpdateTrack(const MusicTrack&);

/*****************************************************************************
  Add a new genre to the database.
*****************************************************************************/
void AddGenre(const std::string&);

/*****************************************************************************
  Delete a genre from the database.
*****************************************************************************/
void DeleteGenre(const std::string&);

/*****************************************************************************
  Returns a sorted vector of all genres present in the database.
*****************************************************************************/
const std::vector<std::string>& GetGenres();

/*****************************************************************************
  Checks if passed genre is used by any tracks.
*****************************************************************************/
bool GenreInUse(const std::string&);

/*****************************************************************************
  Add a new format to the database.
*****************************************************************************/
void AddFormat(const std::string&);

/*****************************************************************************
  Delete a format from the database.
*****************************************************************************/
void DeleteFormat(const std::string&);

/*****************************************************************************
  Returns a vector of all formats present in the database.
*****************************************************************************/
const std::vector<std::string>& GetFormats();

/*****************************************************************************
  Checks if passed format is used by any tracks.
*****************************************************************************/
bool FormatInUse(const std::string&);

/*****************************************************************************
  Return number of tracks currently in database.
*****************************************************************************/
uint GetCount() { return index2ID.size(); }

/*****************************************************************************
```

```
  Sort collection. Argument string should be an SQL ORDER BY clause in
  the form "column1 ASC, column2 DESC ...".
*****************************************************************************/
void Sort(const std::string&);

/*****************************************************************************
  Set filter according to fields in the passed MusicTrack.
  For example, if MusicTrack.artist = "ap", all artists starting with "ap"
  will be included. The TempoRange needs to contain at least two tempi
  for the filtering to work, a low tempo and a high tempo.
*****************************************************************************/
void SetFilter(const MusicTrack&);

/*****************************************************************************
  Turn filter on/off.
*****************************************************************************/
void Filter(bool);

/*****************************************************************************
  Export database contents to the filename passed as argument.
*****************************************************************************/
void ExportCSV(const std::string&);


// These callback functions have to be static for SQLite to be able to
// call them. The first void* paramater will be used to pass a pointer
// to this MusicCollection object through.

/*****************************************************************************
  Callback function for getting track ids, signature prescribed by sqlite.
*****************************************************************************/
static int CBIndex2ID(void*, int, char**, char**);

/*****************************************************************************
  Callback function for getting a track, signature prescribed by sqlite.
*****************************************************************************/
static int CBGetTrack(void*, int, char**, char**);

/*****************************************************************************
  Callback function for getting a track's tempi, signature prescribed by
  sqlite.
*****************************************************************************/
static int CBGetTrackTempi(void*, int, char**, char**);

/*****************************************************************************
  Callback function for getting genres, signature prescribed by sqlite.
*****************************************************************************/
static int CBGetGenres(void*, int, char**, char**);

/*****************************************************************************
  Callback function for getting formats, signature prescribed by sqlite.
*****************************************************************************/
static int CBGetFormats(void*, int, char**, char**);

/*****************************************************************************
  Callback function for checking genres and formats in use, signature
  prescribed by sqlite.
*****************************************************************************/
static int CBInUse(void*, int, char**, char**);

/*****************************************************************************
  Callback function for getting the data needed for exporting to CSV.
*****************************************************************************/
static int CBExport(void*, int, char**, char**);


protected:
```

```
  /*************************************************************************
    Constructor will load collection into memory from database. Protected
    to ensure only singleton is created.
  *************************************************************************/
  MusicCollection();

  /*************************************************************************
    Destructor
  *************************************************************************/
  virtual ~MusicCollection();


private:

  /*************************************************************************
    Copy construction prohibited
  *************************************************************************/
  MusicCollection(const MusicCollection&) { }

  /*************************************************************************
    For escaping quotes in strings
  *************************************************************************/
  std::string Escape(const std::string&);

  /*************************************************************************
    For removing commas from strings
  *************************************************************************/
  std::string RemoveComma(const std::string&);

  /*************************************************************************
    For querying the db for a new list of IDs
  *************************************************************************/
  void QueryIDs();

  /*************************************************************************
    For querying the db for a new list of genres
  *************************************************************************/
  void QueryGenres();

  /*************************************************************************
    For querying the db for a new list of formats
  *************************************************************************/
  void QueryFormats();

  /*************************************************************************
    For throwing exceptions
  *************************************************************************/
  void Error(const std::string&, const char* err = "");


  /*************************************************************************
    Variables
  *************************************************************************/
  sqlite* db;   // the dabatase handle
  char* sqlErr; // error string for passing to sqlite

  std::vector<uint> index2ID;       // maps list ctrl indexes to track ids
  std::vector<std::string> genres;  // all genre names
  std::vector<std::string> formats; // all formats

  std::string sortClause; // current sort clause

  MusicTrack lastTrack; // used for temp storage when querying db
  bool inUse; // set by InUse callback
  std::vector<std::vector<std::string>* > results; // returned results

  bool filtered; // is filter on?

    std::ostringstream sql; // temporary stream for writing SQL statements

    // SQL statement strings
    static const std::string sqlSelectID;
    std::string sqlSelectIDFiltered; // not const since it will be changing
    static const std::string sqlOrderBy;
    static const std::string sqlSelectAllGenres;
    static const std::string sqlSelectAllFormats;
    static const std::string sqlSelectTrack;
    static const std::string sqlSelectTrackTempi1;
    static const std::string sqlSelectTrackTempi2;
    static const std::string sqlInsertTrack;
    static const std::string sqlInsertTrackTempi;
    static const std::string sqlDeleteTrack;
    static const std::string sqlDeleteTrackTempi;
    static const std::string sqlUpdateTrack1;
    static const std::string sqlUpdateTrack2;
    static const std::string sqlInsertGenre;
    static const std::string sqlDeleteGenre;
    static const std::string sqlInsertFormat;
    static const std::string sqlDeleteFormat;
    static const std::string sqlGenreInUse;
    static const std::string sqlFormatInUse;
    static const std::string sqlExport;
    static const std::string sqlBegin;  // begin transaction
    static const std::string sqlCommit; // commit transaction
    static const std::string sqlSC;     // semicolon
    static const std::string sqlBSC;    // bracket & semicolon

};

#endif
```

```
/*****************************************************************************

  MusicCollection.cpp

  Class representing a music collection. All interfacing with the database
  is done from this class.

  It's implemented as a Singleton which means it's constructor is private
  and an instance of the class can only be retrieved by calling the
  Instance method.

  It's also implementing the Observable interface which means clients
  can register as observers of the collection and be informed every time
  the collection changes.

  Author: Erik Jälevik

*****************************************************************************/

#include "MusicCollection.h"
#include "Globals.h"

#include <ctime>
#include <algorithm>
#include <sstream>

using namespace std;

const string MusicCollection::sqlSelectID =
  "SELECT mt.id "
  "FROM   music_track mt "
         "LEFT OUTER JOIN "
         "tempo_map tm "
         "ON mt.id = tm.track_id "
         "INNER JOIN ("
           "SELECT   track_id, MIN(time) AS min_time "
           "FROM     tempo_map "
           "GROUP BY track_id) AS "
         "tmd "
         "ON tm.track_id = tmd.track_id AND tm.time = tmd.min_time ";

const string MusicCollection::sqlOrderBy =
  " ORDER BY ";

const string MusicCollection::sqlSelectAllGenres =
  "SELECT genre FROM genre ORDER BY genre";

const string MusicCollection::sqlSelectAllFormats =
  "SELECT format FROM format ORDER BY format";

const string MusicCollection::sqlSelectTrack =
  "SELECT mt.id, mt.artist, mt.title, mt.length, mt.format, mt.source, "
         "mt.genre, mt.verified, mt.added_on "
  "FROM music_track mt "
  "WHERE mt.id = ";

const string MusicCollection::sqlSelectTrackTempi1 =
  "SELECT time, tempo FROM tempo_map WHERE track_id = ";
const string MusicCollection::sqlSelectTrackTempi2 =
  " ORDER BY time;";

const string MusicCollection::sqlDeleteTrack =
  "DELETE FROM music_track WHERE id = ";

const string MusicCollection::sqlDeleteTrackTempi =
  "DELETE FROM tempo_map WHERE track_id = ";

const string MusicCollection::sqlUpdateTrack1 =
  "UPDATE music_track SET ";
const string MusicCollection::sqlUpdateTrack2 =
  " WHERE id = ";

const string MusicCollection::sqlInsertTrack =
  "INSERT INTO music_track "
  "(id, artist, title, length, format, source, genre, verified, added_on) "
  "VALUES (";

const string MusicCollection::sqlInsertTrackTempi =
  "INSERT INTO tempo_map (track_id, time, tempo) VALUES (";

const string MusicCollection::sqlInsertGenre =
  "INSERT INTO genre (id, genre) VALUES (";

const string MusicCollection::sqlDeleteGenre =
  "DELETE FROM genre WHERE genre = ";

const string MusicCollection::sqlInsertFormat =
  "INSERT INTO format (id, format) VALUES (";

const string MusicCollection::sqlDeleteFormat =
  "DELETE FROM format WHERE format = ";

const string MusicCollection::sqlGenreInUse =
  "SELECT id FROM music_track WHERE genre = ";

const string MusicCollection::sqlFormatInUse =
  "SELECT id FROM music_track WHERE format = ";

const string MusicCollection::sqlExport =
  "SELECT id, artist, title, length, format, "
       "genre, source, verified, added_on "
  "FROM music_track ORDER BY artist, title";

const string MusicCollection::sqlBegin = "BEGIN;";
const string MusicCollection::sqlCommit = "COMMIT;";

const string MusicCollection::sqlSC = ";";
const string MusicCollection::sqlBSC = ");";


/*****************************************************************************
  Instance
*****************************************************************************/
MusicCollection* MusicCollection::Instance() {

  static MusicCollection instance;
  return &instance;

}

/*****************************************************************************
  Constructor
*****************************************************************************/
MusicCollection::MusicCollection() :
  db(0),
  sqlErr(0),
  sqlSelectIDFiltered(sqlSelectID), // will change on first filtering action
  filtered(false),
  sortClause("artist ASC") {

  results.reserve(1000);

}

/*****************************************************************************
  Destructor
```

```
/*****************************************************************************/
MusicCollection::~MusicCollection() {

  if (db != 0) {
    sqlite_close(db);
  }

  // Delete vectors in results
  vector<vector<string>* >::iterator pos;
  for (pos = results.begin(); pos != results.end(); ++pos) {
    delete *pos;
  }

}

/*****************************************************************************
  Open
*****************************************************************************/
void MusicCollection::Open(const string& database) {

  if (db != 0) {
    sqlite_close(db);
  }

  // 2nd argument is r/w mode but is ignored in current implementation
  db = sqlite_open(database.c_str(), 0, &sqlErr);

  if (db != 0) {
    lastTrack.Clear();
    QueryIDs();
    Notify();
  }
  else {
    string s = "Could not open database '" + database + "'.";
    Error(s, sqlErr);
  }

}

/*****************************************************************************
  GetTrack
*****************************************************************************/
MusicTrack& MusicCollection::GetTrack(uint index) {

  uint id = index2ID.at(index);

  // TODO: Not very good, this is copied every time
  if (id == lastTrack.GetID()) {
    return lastTrack;
  }

  lastTrack.Clear();

  // Get track itself
  sql.str("");
  sql << sqlSelectTrack << id << sqlSC;

  //LOG("About to query: " << sql.str() << endl);

  int stTrk = sqlite_exec(
    db, sql.str().c_str(), &MusicCollection::CBGetTrack, this, &sqlErr);

  // Get its tempi
  sql.str("");
  sql << sqlSelectTrackTempi1 << id << sqlSelectTrackTempi2;

  //LOG("About to query: " << sql.str() << endl);

  int stTmp = sqlite_exec(
    db, sql.str().c_str(), &MusicCollection::CBGetTrackTempi, this, &sqlErr);

  if (stTrk != SQLITE_OK || stTmp != SQLITE_OK) {
    LOG("stTrack: " << stTrk << ", stTempo: " << stTmp << endl);
    Error("Could not retrieve track from database.", sqlErr);
  }

  // lastTrack is now fully built
  return lastTrack;

}

/*****************************************************************************
  AddTrack
*****************************************************************************/
void MusicCollection::AddTrack(const MusicTrack& trk) {

  // Format current date and time
  time_t t = time(NULL);
  tm* now = localtime(&t);
  char addedOn[16];
  strftime(addedOn, 16, "%Y%m%d%H%M%S", now);

  MusicTrack& track = const_cast<MusicTrack&>(trk);

  // Insert track
  sql.str("");
  sql << sqlInsertTrack <<
    "NULL"                    << ", '"  <<
    Escape(track.GetArtist())  << "', '" <<
    Escape(track.GetTitle())   << "', "  <<
    track.GetLength()          << ", '"  <<
    Escape(track.GetFormat())   << "', '" <<
    Escape(track.GetSource())  << "', '" <<
    Escape(track.GetGenre())   << "', "  <<
    track.GetVerified()        << ", '"  <<
    addedOn                    << "')"    << sqlBSC;

  LOG("About to query: " << sql.str() << endl);

  // Execute to receive an id
  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not add track.", sqlErr);
  }

  uint id = sqlite_last_insert_rowid(db);

  // Insert tempi
  sql.str("");
  sql << sqlBegin;
  IntShortMap& range = track.GetTempoRange().GetMap();
  IntShortMap::iterator it;
  for (it = range.begin(); it != range.end(); ++it) {
    sql << sqlInsertTrackTempi <<
      id         << ", " <<
      it->first  << ", " <<
      it->second << sqlBSC;
  }
  sql << sqlCommit;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not add track.", sqlErr);
  }

  // Also add it to index
```

```
  index2ID.push_back(id);

  Notify();

}

/*****************************************************************************
  DeleteTrack
*****************************************************************************/
void MusicCollection::DeleteTrack(uint index) {

  vector<uint> v;
  v.push_back(index);
  DeleteTracks(v);

}

/*****************************************************************************
  DeleteTracks
*****************************************************************************/
void MusicCollection::DeleteTracks(vector<uint>& indices) {

  sql.str("");
  uint id;
  vector<uint>::const_iterator it;
  sql << sqlBegin;
  for (it = indices.begin(); it != indices.end(); ++it) {
    id = index2ID.at(*it);
    sql << sqlDeleteTrack << id << sqlSC;
    sql << sqlDeleteTrackTempi << id << sqlSC;
  }
  sql << sqlCommit;

  LOG("About to query: " << sql.str() << endl);

  // Passing in 0 for a callback, so no callback will be called.
  int status = sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr);

  if (status != SQLITE_OK) {
    Error("Could not delete track(s).", sqlErr);
  }

  // Removal from index2ID needs to be done in reverse order
  // to not corrupt position of subsequent indices
  sort(indices.begin(), indices.end());
  vector<uint>::reverse_iterator it2;
  for (it2 = indices.rbegin(); it2 != indices.rend(); ++it2) {
    index2ID.erase(index2ID.begin() + *it2);
  }

  Notify();

}

/*****************************************************************************
  UpdateTrack
*****************************************************************************/
void MusicCollection::UpdateTrack(const MusicTrack& trk) {

  MusicTrack& track = const_cast<MusicTrack&>(trk);

  // Update music_track
  sql.str("");
  sql << sqlBegin;
  sql << sqlUpdateTrack1 <<
    "artist = '"   << Escape(track.GetArtist())   << "', " <<
    "title  = '"   << Escape(track.GetTitle())    << "', " <<
    "length = "    << track.GetLength()           << ",  "  <<
    "format = '"   << Escape(track.GetFormat())   << "', " <<
    "source = '"   << Escape(track.GetSource())   << "', " <<
    "genre = '"    << Escape(track.GetGenre())    << "',  " <<
    "verified = " << track.GetVerified()         <<
    sqlUpdateTrack2 << track.GetID() << sqlSC;

  // Update tempo_map
  sql << sqlDeleteTrackTempi << track.GetID() << sqlSC;
  IntShortMap& range = track.GetTempoRange().GetMap();
  IntShortMap::iterator it;
  for (it = range.begin(); it != range.end(); ++it) {
    sql << sqlInsertTrackTempi <<
      track.GetID() << ", " <<
      it->first     << ", " <<
      it->second    << sqlBSC;
  }
  sql << sqlCommit;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not update track.", sqlErr);
  }

  Notify();

}

/*****************************************************************************
  AddGenre
*****************************************************************************/
void MusicCollection::AddGenre(const string& g) {

  sql.str("");
  sql << sqlInsertGenre << "NULL" << ", '" << Escape(g) << "'" << sqlBSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not add genre.", sqlErr);
  }

  QueryGenres();
  Notify();

}

/*****************************************************************************
  DeleteGenre
  TODO : need to think about referential integrity in connection with deleting
  genres.
*****************************************************************************/
void MusicCollection::DeleteGenre(const string& g) {

  if (GenreInUse(g)) {
    string s = "The genre '" + g + "' can't be deleted because there are "
      "tracks in the database of that genre.";
    Error(s);
  }

  sql.str("");
  sql << sqlDeleteGenre << "'" << Escape(g) << "'" << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not delete genre.", sqlErr);
  }
```

```
  QueryGenres();
  Notify();

}

/*****************************************************************************
  GetGenres
*****************************************************************************/
const vector<string>& MusicCollection::GetGenres() {

  // Should only be empty first time method is called
  if (genres.empty()) {
    QueryGenres();
  }

  return genres;

}

/*****************************************************************************
  GenreInUse
*****************************************************************************/
bool MusicCollection::GenreInUse(const string& g) {

  inUse = false;

  sql.str("");
  sql << sqlGenreInUse << "'" << Escape(g) << "'" << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBInUse, this,
    &sqlErr) != SQLITE_OK) {
    Error("Could not check if genre is in use.", sqlErr);
  }

  if (inUse) {
    inUse = false;
    return true;
  }

  return false;

}

/*****************************************************************************
  AddFormat
*****************************************************************************/
void MusicCollection::AddFormat(const string& f) {

  sql.str("");
  sql << sqlInsertFormat << "NULL" << ", '" << Escape(f) << "'" << sqlBSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not add format.", sqlErr);
  }

  QueryFormats();
  Notify();

}

/*****************************************************************************
  DeleteFormat
*****************************************************************************/
void MusicCollection::DeleteFormat(const string& f) {

  if (FormatInUse(f)) {
    string s = "The format '" + f + "' can't be deleted because there are "
      "tracks in the database of that format.";
    Error(s);
  }

  sql.str("");
  sql << sqlDeleteFormat << "'" << Escape(f) << "'" << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), 0, 0, &sqlErr) != SQLITE_OK) {
    Error("Could not delete format.", sqlErr);
  }

  QueryFormats();
  Notify();

}

/*****************************************************************************
  GetFormats
*****************************************************************************/
const vector<string>& MusicCollection::GetFormats() {

  // Should only be empty first time method is called
  if (formats.empty()) {
    QueryFormats();
  }

  return formats;

}

/*****************************************************************************
  FormatInUse
*****************************************************************************/
bool MusicCollection::FormatInUse(const string& f) {

  inUse = false;

  sql.str("");
  sql << sqlFormatInUse << "'" << Escape(f) << "'" << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBInUse, this,
    &sqlErr) != SQLITE_OK) {
    Error("Could not check if format is in use.", sqlErr);
  }

  if (inUse) {
    inUse = false;
    return true;
  }

  return false;

}

/*****************************************************************************
  Sort
*****************************************************************************/
void MusicCollection::Sort(const string& sc) {

  sortClause = sc;
```

```
  QueryIDs();

  Notify();

}

/*****************************************************************************
  Filter
*****************************************************************************/
void MusicCollection::SetFilter(const MusicTrack& flt) {

  // The LIKE operator is case sensitive for 8-bit iso8859 characters
  // or UTF-8 characters. For example, the expression 'a' LIKE 'A' is TRUE
  // but 'æ' LIKE 'Æ' is FALSE.

  MusicTrack& filter = const_cast<MusicTrack&>(flt);

  ostringstream os;
  os << sqlSelectID << " WHERE "
    "mt.artist LIKE '" << Escape(filter.GetArtist()) << "%' AND "
    "mt.title LIKE '"  << Escape(filter.GetTitle())  << "%' AND "
    "mt.format LIKE '" << Escape(filter.GetFormat()) << "%'";

  // Check if filter contains a genre
  string& g = filter.GetGenre();
  if (g != "") {
     os << " AND mt.genre = '" << Escape(g) << "'";
  }

  // Check if filter contains a tempo range
  TempoRange& range = filter.GetTempoRange();
  if (!range.IsEmpty()) {
    os << " AND tm.tempo BETWEEN " <<
      range.GetFirst() << " AND " << range.GetLast();
  }

  sqlSelectIDFiltered = os.str();

}

/*****************************************************************************
  Filter
*****************************************************************************/
void MusicCollection::Filter(bool status) {

  // TODO: should not be able to switch filter on before having called Filter

  filtered = status;

  QueryIDs();

  Notify();

}

/*****************************************************************************
  ExportCSV
*****************************************************************************/
void MusicCollection::ExportCSV(const string& file) {

  sql.str("");
  sql << sqlExport << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  // Delete old vectors in results in case this isn't the first export
  vector<vector<string>* >::iterator pos;
  for (pos = results.begin(); pos != results.end(); ++pos) {
    delete *pos;
  }
  results.clear();

  if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBExport, this,
    &sqlErr) != SQLITE_OK) {
    Error("Could not retrieve data for exporting.", sqlErr);
  }

  ofstream csvFile(file.c_str());
  csvFile << "Artist,Title,Tempo,Length,Format,Genre,Source,Verified,"
    "Added On" << endl;

  for (pos = results.begin(); pos != results.end(); ++pos) {

    // Output artist, title
    csvFile << RemoveComma((*(*pos))[1]) << ",";
    csvFile << RemoveComma((*(*pos))[2]) << ",";

    // Get tempi
    sql.str("");
    sql << sqlSelectTrackTempi1 << (*(*pos))[0] << sqlSelectTrackTempi2;

    LOG("About to query: " << sql.str() << endl);

    lastTrack.Clear();
    if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBGetTrackTempi,
      this, &sqlErr) != SQLITE_OK) {
      Error("Could not retrieve tempo data for exporting.", sqlErr);
    }

    csvFile << lastTrack.GetTempoRange().GetRangeString() << ",";
    csvFile << MusicTrack::Secs2Time(atoi((*(*pos))[3].c_str())) << ",";
    csvFile << RemoveComma((*(*pos))[4]) << ",";
    csvFile << RemoveComma((*(*pos))[5]) << ",";
    csvFile << RemoveComma((*(*pos))[6]) << ",";
    csvFile << ((*(*pos))[7] == "1" ? "true" : "false") << ",";
    csvFile << MusicTrack::Date2DateString((*(*pos))[8]) << endl;

  }

}

/*****************************************************************************
  Escape
*****************************************************************************/
string MusicCollection::Escape(const string& s) {

  string str(s);
  string::size_type index = str.find("'");

  while (index != string::npos) {
    str.replace(index, 1, "''");
    index = str.find("'", index + 2);
  }

  return str;

}

/*****************************************************************************
  RemoveComma
*****************************************************************************/
string MusicCollection::RemoveComma(const string& s) {

  string str(s);
  string::size_type index = str.find(",");
```

```
  while (index != string::npos) {
    str.replace(index, 1, "");
    index = str.find(",", index);
  }

  return str;

}

/*****************************************************************************
  QueryIDs
*****************************************************************************/
void MusicCollection::QueryIDs() {

  index2ID.clear();

  sql.str("");
  if (filtered) {
    sql << sqlSelectIDFiltered << sqlOrderBy << sortClause << sqlSC;
  }
  else {
    sql << sqlSelectID << sqlOrderBy << sortClause << sqlSC;
  }

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBIndex2ID,
    this, &sqlErr)) {
    Error("Could not read collection data from database.", sqlErr);
  }

}

/*****************************************************************************
  QueryGenres
*****************************************************************************/
void MusicCollection::QueryGenres() {

  genres.clear();

  sql.str("");
  sql << sqlSelectAllGenres << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBGetGenres, this,
    &sqlErr) != SQLITE_OK) {
    Error("Could not get genres.", sqlErr);
  }

}


/*****************************************************************************
  QueryFormats
*****************************************************************************/
void MusicCollection::QueryFormats() {

  formats.clear();

  sql.str("");
  sql << sqlSelectAllFormats << sqlSC;

  LOG("About to query: " << sql.str() << endl);

  if (sqlite_exec(db, sql.str().c_str(), &MusicCollection::CBGetFormats, this,
    &sqlErr) != SQLITE_OK) {
    Error("Could not get format.", sqlErr);
  }

}

/*****************************************************************************
  Error
*****************************************************************************/
void MusicCollection::Error(const string& msg, const char* err) {

  ostringstream tmp;
  tmp << msg;

  #ifdef _DEBUG
    if (err != 0) {
      tmp << "\n\nSQLite error: " << err;
    }
  #endif

  throw DatabaseException(tmp.str());

}




/*****************************************************************************
  CallBack Index2ID
*****************************************************************************/
int MusicCollection::CBIndex2ID(void* p, int argc, char** argv, char** cols) {

  MusicCollection* me = static_cast<MusicCollection*>(p);

  if (argc > 0) {
    uint id = atoi(argv[0]);
    if (id != 0) {
      me->index2ID.push_back(id);
    }
    else {
      LOG("ERROR! CBIndex2ID id could not be parsed.\n");
      return -1;
      // No exception thrown, ctor does that when function returns.
    }
  }
  else {
    LOG("ERROR! CBIndex2ID argc = 0\n");
    return -1;
    // No exception thrown, ctor does that when function returns.
  }

  return 0;

}

/*****************************************************************************
  CallBack GetTrack
*****************************************************************************/
int MusicCollection::CBGetTrack(void* p, int argc, char** argv, char** cols) {

  MusicCollection* me = static_cast<MusicCollection*>(p);

  // Retrieve fields and put them in lastTrack
  if (argc == 9) {
    me->lastTrack.SetID(argv[0] ? atoi(argv[0]) : 0);
    me->lastTrack.SetArtist(argv[1] ? argv[1] : "");
    me->lastTrack.SetTitle(argv[2] ? argv[2] : "");
    // If length can't be parsed by atoi it will be set to 0
```

```
    me->lastTrack.SetLength(argv[3] ? atoi(argv[3]) : 0);
    me->lastTrack.SetFormat(argv[4] ? argv[4] : "");
    me->lastTrack.SetSource(argv[5] ? argv[5] : "");
    me->lastTrack.SetGenre(argv[6] ? argv[6] : "");
    me->lastTrack.SetVerified(argv[7] ? atoi(argv[7]) == 1 : false);
    me->lastTrack.SetAddedOn(argv[8] ? argv[8] : "");
  }
  else {
    LOG("ERROR! CBGetTrack argc = " << argc << endl);
    return -1;
  }

  return 0;

}

/*****************************************************************************
  CallBack GetTrackTempi
*****************************************************************************/
int MusicCollection::CBGetTrackTempi(void* p, int argc, char** argv,
  char** cols) {

  MusicCollection* me = static_cast<MusicCollection*>(p);

  // Retrieve fields and put them in lastTrack
  if (argc == 2) {
    uint time = argv[0] ? atoi(argv[0]) : 0;
    ushort tempo = argv[1] ? atoi(argv[1]) : 0;
    me->lastTrack.GetTempoRange().Add(time, tempo);
  }
  else {
    LOG("ERROR! CBGetTrackTempi argc = " << argc << endl);
    return -1;
  }

  return 0;

}

/*****************************************************************************
  CallBack GetGenres
*****************************************************************************/
int MusicCollection::CBGetGenres(void* p, int argc, char** argv, char** cols) {

  if (argc == 1) {
    static_cast<MusicCollection*>(p)->genres.push_back(argv[0] ? argv[0] : " ");
  }
  else {
    LOG("ERROR! CBGetGenres argc = 0\n");
    return -1;
  }

  return 0;

}

/*****************************************************************************
  CallBack GetFormats
*****************************************************************************/
int MusicCollection::CBGetFormats(void* p, int argc, char** argv, char** cols) {

  if (argc == 1) {
    static_cast<MusicCollection*>(p)->formats.push_back(argv[0] ? argv[0] : " ");
  }
  else {
    LOG("ERROR! CBGetFormats argc = 0\n");
    return -1;
  }

  return 0;

}

/*****************************************************************************
  CallBack InUse
*****************************************************************************/
int MusicCollection::CBInUse(void* p, int argc, char** argv, char** cols) {

  MusicCollection* me = static_cast<MusicCollection*>(p);

  if (argc > 0) {
    me->inUse = true;
  }

  return 0;

}

/*****************************************************************************
  CallBack Export
*****************************************************************************/
int MusicCollection::CBExport(void* p, int argc, char** argv, char** cols) {

  MusicCollection* me = static_cast<MusicCollection*>(p);

  // Retrieve fields and put them in results
  if (argc == 9) {

    vector<string>* v = new vector<string>();
    v->reserve(argc);

    for (int i = 0; i < argc; ++i) {
      v->push_back(argv[i] ? argv[i] : "");
    }

    me->results.push_back(v);

  }
  else {
    LOG("ERROR! CBGetExport argc = " << argc << endl);
    return -1;
  }

  return 0;

}
```

```
/*****************************************************************************

  MusicTrack.h

  Class for holding tracks that are part of the collection in memory.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef MUSICTRACK_H
#define MUSICTRACK_H

#include "TempoRange.h"
#include "Globals.h"

#include <string>
#include <vector>

class MusicTrack {

  public:

    /*************************************************************************
      Default constructor creates empty MusicTrack.
    *************************************************************************/
    MusicTrack() { Init(); }
    MusicTrack(const MusicTrack&);
    virtual ~MusicTrack() { }
    MusicTrack& operator=(const MusicTrack&);

    /*************************************************************************
      Getters and setters for members.
    *************************************************************************/
    void SetID(const uint i) { id = i; }
    uint GetID() { return id; }

    void SetArtist(const std::string& a) { artist = a; }
    std::string& GetArtist() { return artist; }

    void SetTitle(const std::string& t) { title = t; }
    std::string& GetTitle() { return title; }

    void SetLength(const uint l) { length = l; }
    uint GetLength() { return length; }
    std::string GetLengthString() { return Secs2Time(length); }

    void SetFormat(const std::string& f) { format = f; }
    std::string& GetFormat() { return format; }

    void SetSource(const std::string& s) { source = s; }
    std::string& GetSource() { return source; }

    void SetGenre(const std::string& g) { genre = g; }
    std::string& GetGenre() { return genre; }

    void SetTempoRange(const TempoRange& t) { tempoRange = t; }
    TempoRange& GetTempoRange() { return tempoRange; }

    void SetVerified(bool v) { verified = v; }
    bool GetVerified() { return verified; }

    void SetAddedOn(const std::string& a) { addedOn = a; }
    std::string& GetAddedOn() { return addedOn; }
    std::string GetAddedOnString() { return Date2DateString(addedOn); }

    void SetAccess(const std::string& a) { access = a; }
    std::string& GetAccess() { return access; }

    /*************************************************************************
      Clears the track of all data.
    *************************************************************************/
    void Clear();

    /*************************************************************************
      Takes a length in seconds and converts it to a string of the format
      mm:ss.
    *************************************************************************/
    static std::string Secs2Time(int);

    /*************************************************************************
      Takes a string of the format mm:ss and converts it to a length in
      seconds. Returns -1 if the string is not a valid time.
    *************************************************************************/
    static int Time2Secs(const std::string&);

    /*************************************************************************
      Takes a date in the format it's stored in the database and converts it
      into a more readable format. Returns the string unchanged if it's not
      in the right format.
    *************************************************************************/
    static std::string Date2DateString(const std::string&);


  protected:

    /*************************************************************************
      Initiliases everything to 0.
    *************************************************************************/
    void Init();


  private:

    /*************************************************************************
      Variables
    *************************************************************************/
    uint id;
    std::string artist;
    std::string title;
    uint length;
    std::string format;
    std::string source;
    std::string genre;
    TempoRange tempoRange;
    bool verified;
    std::string addedOn;
    std::string access; // used internally for specifying how to access track

};

#endif
```

```
/*****************************************************************************

  MusicTrack.cpp

  Class for holding tracks that are part of the collection in memory.

  Author: Erik Jälevik

*****************************************************************************/

#include "MusicTrack.h"
#include "FileInStream.h"

#include <sstream>

using namespace std;

/*****************************************************************************
  Copy constructor
*****************************************************************************/
MusicTrack::MusicTrack(const MusicTrack& that) :
  id(that.id),
  artist(that.artist),
  title(that.title),
  length(that.length),
  format(that.format),
  source(that.source),
  genre(that.genre),
  tempoRange(that.tempoRange),
  verified(that.verified),
  addedOn(that.addedOn),
  access(that.access) {

}

/*****************************************************************************
  Operator=
*****************************************************************************/
MusicTrack& MusicTrack::operator=(const MusicTrack& that) {

  if(&that != this) {
    id = that.id;
    artist = that.artist;
    title = that.title;
    length = that.length;
    format = that.format;
    source = that.source;
    genre = that.genre;
    tempoRange = that.tempoRange;
    verified = that.verified;
    addedOn = that.addedOn;
    access = that.access;
  }
  return *this;

}

/*****************************************************************************
  Init
*****************************************************************************/
void MusicTrack::Init() {

  id = 0;
  artist = "";
  title = "";
  length = 0;
  format = "";
  source = "";
  genre = "";
  verified = false;
  addedOn = "";
  access = "";

}

/*****************************************************************************
  Clear
*****************************************************************************/
void MusicTrack::Clear() {

  Init();
  tempoRange.Clear();

}

/*****************************************************************************
  Secs2Time
*****************************************************************************/
string MusicTrack::Secs2Time(int l) {

  int mins = l / 60;
  int secs = l % 60;

  ostringstream os;
  if (secs < 10) {
    os << mins << ":0" << secs;
  }
  else {
    os << mins << ":" << secs;
  }
  string s = os.str();

  return s;

}

/*****************************************************************************
  Time2Secs
*****************************************************************************/
int MusicTrack::Time2Secs(const string& t) {

  istringstream is(t);
  int mins, secs, len;
  if (is >> mins) {
    is.get(); // discard colon
    if (is >> secs) {
      len = (mins * 60) + secs;
      return len;
    }
  }

  return -1;

}

/*****************************************************************************
  Date2DateString
*****************************************************************************/
string MusicTrack::Date2DateString(const string& d) {

  if (d.size() >= 12) {
    ostringstream os;
    os << d.substr(6, 2) << "/"
       << d.substr(4, 2) << "/"
       << d.substr(0, 4) << " "
       << d.substr(8, 2) << ":"
```

```
      << d.substr(10, 2);
    string s = os.str();
    return s;
  }
  else {
    return d;
  }

}
```

```
/*****************************************************************************

  MusicTrackDialog.h

  GUI dialog window for editing music track details.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef MUSICTRACKDIALOG_H
#define MUSICTRACKDIALOG_H

#include "MusicTrack.h"

#include <wx/grid.h>

class GridKeyHandler;

class MusicTrackDialog : public wxDialog {

  public:

    /*************************************************************************
      Constructor takes parent window, id, position and the track to
      display/edit as parameters.
    *************************************************************************/
    MusicTrackDialog(wxWindow*, wxWindowID, const wxPoint&, MusicTrack&);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~MusicTrackDialog();

    /*************************************************************************
      Handler for Format drop down. Enables/Disables Play button.
    *************************************************************************/
    void OnFormatChange(wxCommandEvent&);

    /*************************************************************************
      Handler for Play button.
    *************************************************************************/
    void OnPlay(wxCommandEvent&);

    /*************************************************************************
      Handler for Verify button.
    *************************************************************************/
    void OnVerify(wxCommandEvent&);

    /*************************************************************************
      Handler for Delete button.
    *************************************************************************/
    void OnDelete(wxCommandEvent&);

    /*************************************************************************
      Handler for Tapper button.
    *************************************************************************/
    void OnTapper(wxCommandEvent&);

    /*************************************************************************
      Handler for OK button.
    *************************************************************************/
    void OnOK(wxCommandEvent&);

    /*************************************************************************
      Handler for Cancel button.
    *************************************************************************/
    void OnCancel(wxCommandEvent&);
```

```
  /**************************************************************************
    Handler for Apply button.
  **************************************************************************/
  void OnApply(wxCommandEvent&);

  /**************************************************************************
    Handler for cell change.
  **************************************************************************/
  void OnCellChange(wxGridEvent&);

  /**************************************************************************
    OnChar, not used
  **************************************************************************/
  void OnChar(wxKeyEvent&);


private:

  /**************************************************************************
    Validate time entry made in a tempo range time cell.
  **************************************************************************/
  bool ValidateTime(wxString&);

  /**************************************************************************
    Validate tempo entry made in a tempo cell.
  **************************************************************************/
  bool ValidateTempo(wxString&);

  /**************************************************************************
    Used by the OK and Apply handlers to store away track details
  **************************************************************************/
  void SaveTrack();

  /**************************************************************************
    Autogenerated method for building dialog
  **************************************************************************/
  void BuildMe();


  /**************************************************************************
    Variables
  **************************************************************************/

  // Controls
  wxTextCtrl* artistTC;
  wxTextCtrl* titleTC;
  wxTextCtrl* lengthTC;
  wxChoice* formatDD;
  wxTextCtrl* sourceTC;
  wxChoice* genreDD;
  wxTextCtrl* addedTC;

  wxButton* playBut;

  wxBitmapButton* verifyBut;
  wxBitmapButton* deleteBut;
  wxBitmapButton* tapBut;
  wxGrid* tempoGrid;

  wxButton* okBut;
  wxButton* cancelBut;
  wxButton* applyBut;

  GridKeyHandler* gridKeyHandler; // event handler for the tempo grid

  // Model objects
  MusicTrack track;

    enum {
      ID_ARTIST_TC,
      ID_TITLE_TC,
      ID_LENGTH_TC,
      ID_FORMAT_DD,
      ID_SOURCE_TC,
      ID_GENRE_DD,
      ID_ADDED_TC,
      ID_PLAY_BUT,
      ID_PLOT_WIN,
      ID_VERIFY_BUT,
      ID_DELETE_BUT,
      ID_TAP_BUT,
      ID_TEMPO_GRID,
      ID_OK_NOENTER,
      ID_CANCEL_NOESC
    };

    DECLARE_EVENT_TABLE()

};


/*****************************************************************************

  GridKeyHandler

  A key handler that will be pushed into the wxGrid owned by
  MusicTrackDialog. It's needed to capture key presses.

  Author: Erik Jälevik

*****************************************************************************/

class GridKeyHandler : public wxEvtHandler {

  public:

    /**************************************************************************
      Constructor takes the dialog that owns the grid.
    **************************************************************************/
    GridKeyHandler(MusicTrackDialog* mtd) : dialog(mtd) { }

    /**************************************************************************
      Key event handler.
    **************************************************************************/
    void OnChar(wxKeyEvent&);


  private:

    /**************************************************************************
      Variables
    **************************************************************************/
    MusicTrackDialog* dialog; // the MusicTrackDialog of which the grid is part

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/*****************************************************************************

  MusicTrackDialog.cpp

  GUI dialog window for editing music track details.

  Author: Erik Jälevik

*****************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "MusicTrackDialog.h"
#include "MusicCollection.h"
#include "TapperDialog.h"
#include "Icons.h"
#include "Globals.h"

#include <wx/statline.h>
#include <wx/confbase.h>

#include <map>
#include <string>

using namespace std;


/*****************************************************************************
  Constructor
*****************************************************************************/
MusicTrackDialog::MusicTrackDialog(wxWindow* parent,
                                   wxWindowID id,
                                   const wxPoint& pos,
                                   MusicTrack& trk) :
  wxDialog(parent, -1, "Track Information", pos, wxDefaultSize,
    wxCAPTION | wxSYSTEM_MENU | wxRESIZE_BORDER),
  track(trk) {

  BuildMe();

}

/*****************************************************************************
  Destructor
*****************************************************************************/
MusicTrackDialog::~MusicTrackDialog() {

  tempoGrid->GetGridWindow()->PopEventHandler(TRUE);

}

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(MusicTrackDialog, wxDialog)
  EVT_CHOICE(ID_FORMAT_DD, MusicTrackDialog::OnFormatChange)
  EVT_BUTTON(ID_PLAY_BUT, MusicTrackDialog::OnPlay)
  EVT_BUTTON(ID_VERIFY_BUT, MusicTrackDialog::OnVerify)
  EVT_BUTTON(ID_DELETE_BUT, MusicTrackDialog::OnDelete)
  EVT_BUTTON(ID_TAP_BUT, MusicTrackDialog::OnTapper)
  EVT_BUTTON(ID_OK_NOENTER, MusicTrackDialog::OnOK)
  EVT_BUTTON(ID_CANCEL_NOESC, MusicTrackDialog::OnCancel)
  EVT_BUTTON(wxID_APPLY, MusicTrackDialog::OnApply)
  EVT_GRID_CELL_CHANGE(MusicTrackDialog::OnCellChange)
  EVT_CHAR(MusicTrackDialog::OnChar)
END_EVENT_TABLE()

/*****************************************************************************
  OnFormatChange
*****************************************************************************/
void MusicTrackDialog::OnFormatChange(wxCommandEvent& event) {

  // Enable Play button if format is a playable file, disable otherwise
  wxString format = formatDD->GetStringSelection();
  if (format == "WAV" ||
      format == "MP3" ||
      format == "OGG") { // ogg not yet implemented
    playBut->Enable();
  }
  else {
   playBut->Disable();
  }

}

/*****************************************************************************
  OnPlay
*****************************************************************************/
void MusicTrackDialog::OnPlay(wxCommandEvent& event) {

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString player = cfg->Read("/Options/General/player", "");
  wxString file = sourceTC->GetValue();

  // Check that program and file exists
  if (!::wxFileExists(player)) {
    ::wxMessageBox("Could not find the player program. Please check that the right "
      "program is specified under Tools, Options, General.",
      "Player Error", wxOK | wxICON_ERROR | wxCENTRE);
    return;
  }

  if (!::wxFileExists(file)) {
    ::wxMessageBox("Could not find the file at the location specified in the "
      "Source field. Please check that it's correct.",
      "Player Error", wxOK | wxICON_ERROR | wxCENTRE);
    return;
  }

  file.Prepend("\"");
  file.Append("\"");

  const char* argv[3];
  argv[0] = player.c_str();
  argv[1] = file.c_str();
  argv[2] = NULL;

  long retCode = ::wxExecute(const_cast<char**>(argv), wxEXEC_ASYNC);

  // Return codes:
  // pid if file not found
  // 0 if program not found
  // -1 if connected to running process
  if (retCode == 0) {
    ::wxMessageBox("Could not launch the player program. Please check that "
      "the right program is specified under Tools, Options, General.",
      "Player Error", wxOK | wxICON_ERROR | wxCENTRE);
  }
```

```
}

/*****************************************************************************
  OnVerify
*****************************************************************************/
void MusicTrackDialog::OnVerify(wxCommandEvent& event) {

  // Verifies tempo and changes all cells from red to black
  track.SetVerified(true);

  wxColour c = wxSystemSettings::GetColour(wxSYS_COLOUR_WINDOWTEXT);
  tempoGrid->SetDefaultCellTextColour(c);
  tempoGrid->BeginBatch();
  for (int i = 0; i < tempoGrid->GetNumberCols(); ++i) {
    for (int j = 0; j < tempoGrid->GetNumberRows(); ++j) {
      tempoGrid->SetCellTextColour(i, j, c);
    }
  }
  tempoGrid->EndBatch();

}

/*****************************************************************************
  OnDelete
*****************************************************************************/
void MusicTrackDialog::OnDelete(wxCommandEvent& event) {

  // If multiple selection, delete all
  if (tempoGrid->IsSelection()) {

    tempoGrid->BeginBatch();

    for (int i = 0; i < tempoGrid->GetNumberRows(); ) {

      // Only delete if row is in selection and it's not the last row
      if ((tempoGrid->IsInSelection(i, 0) ||
          tempoGrid->IsInSelection(i, 1)) &&
          i != tempoGrid->GetNumberRows() - 1 ) {

        tempoGrid->DeleteRows(i, 1);
      }
      else {
        ++i; // this is because deletion causes all indices to decrease
      }
    }

    tempoGrid->EndBatch();

  }

  // If no multisel, delete row under cursor
  else {
    int row = tempoGrid->GetGridCursorRow();
    if (row != tempoGrid->GetNumberRows() - 1 ) {
      tempoGrid->DeleteRows(row, 1);
    }
  }

}

/*****************************************************************************
  OnTapper
*****************************************************************************/
void MusicTrackDialog::OnTapper(wxCommandEvent& event) {

  // Get selected row
  int row = tempoGrid->GetGridCursorRow();

  // Tapper can't throw
  TapperDialog tapper(this, -1);
  if (tapper.ShowModal() == wxID_OK) {

    // Get tempo from tapper and update grid
    int tempo = tapper.GetTempo();
    wxString t;
    t << tempo;
    tempoGrid->SetCellValue(row, 1, t);

  }

}

/*****************************************************************************
  OnOK
*****************************************************************************/
void MusicTrackDialog::OnOK(wxCommandEvent& event) {

  try {
    SaveTrack();
  }
  catch (MusicCollection::DatabaseException& e) {
    wxString msg = e.what();
    msg += "\n\nDo you want to close the window anyway?";
    wxMessageDialog dlg(this, msg, "Database Error",
      wxYES_NO | wxICON_ERROR | wxCENTRE);
    int ans = dlg.ShowModal();

    if (ans != wxID_YES) {
      return;
    }

  }

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*****************************************************************************
  OnCancel
*****************************************************************************/
void MusicTrackDialog::OnCancel(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_CANCEL);
  }
  else {
    SetReturnCode(wxID_CANCEL);
    Show(FALSE);
    Destroy();
  }

}

/*****************************************************************************
  OnApply
*****************************************************************************/
void MusicTrackDialog::OnApply(wxCommandEvent& event) {
```

```
  try {
    SaveTrack();
  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

}

/*********************************************************************************
  OnChange
*********************************************************************************/
void MusicTrackDialog::OnCellChange(wxGridEvent& event) {

  // Check that value is numeric
  int row = event.GetRow();
  int col = event.GetCol();
  wxString val = tempoGrid->GetCellValue(row, col);

  if (row == tempoGrid->GetNumberRows() - 1) {
    tempoGrid->AppendRows(1);
  }

  if (col == 0 && !ValidateTime(val)) {
    event.Veto();
    ::wxMessageBox("A time must be in the format mm:ss.\n\nThat is, a number "
      "of minutes followed by a colon followed by a number of seconds.",
      "Input Error", wxOK | wxICON_ERROR | wxCENTRE);
    tempoGrid->SetGridCursor(row, col);
  }
  else if (col == 1 && !ValidateTempo(val)) {
    event.Veto();
    ::wxMessageBox("A tempo must be a number.",
      "Input Error", wxOK | wxICON_ERROR | wxCENTRE);
    tempoGrid->SetGridCursor(row, col);
  }
  else {
    // Should only get here if validation succeeded
    event.Skip();
  }

}

/*********************************************************************************
  OnChar
*********************************************************************************/
void MusicTrackDialog::OnChar(wxKeyEvent& event) {

  event.Skip();

}

/*********************************************************************************
  ValidateTime
*********************************************************************************/
bool MusicTrackDialog::ValidateTime(wxString& val) {

  return MusicTrack::Time2Secs(val.c_str()) != -1;

}

/*********************************************************************************
  ValidateTempo
*********************************************************************************/
bool MusicTrackDialog::ValidateTempo(wxString& val) {

  // Can't use wxString::IsNumber cause it doesn't work on chars above 127
  // Should be an int
  const char* s = val.c_str();
  while (*s) {
    if (!isdigit((unsigned char)*s)) {
      return false;
    }
    ++s;
  }
  return true;

}

/*********************************************************************************
  SaveTrack
*********************************************************************************/
void MusicTrackDialog::SaveTrack() {

  wxBusyCursor busy;

  // Ensure that currently edited cell, if any, is validated and saved
  tempoGrid->SaveEditControlValue();

  // Store away data
  track.SetArtist(artistTC->GetValue().c_str());
  track.SetTitle(titleTC->GetValue().c_str());
  track.SetFormat(formatDD->GetStringSelection().c_str());
  track.SetSource(sourceTC->GetValue().c_str());
  // If no genre is selected, this will be set to the empty string
  track.SetGenre(genreDD->GetStringSelection().c_str());

  // Returns a reference to actual range in track so just need to change it
  // directly
  TempoRange& range = track.GetTempoRange();
  range.Clear();
  for (int i = 0; i < tempoGrid->GetNumberRows() - 1; ++i) {
    wxString timeStr = tempoGrid->GetCellValue(i, 0);
    int time = MusicTrack::Time2Secs(timeStr.c_str());
    int tempo = atoi(tempoGrid->GetCellValue(i, 1).c_str());
    range.Add(time, tempo);
  }
  // Make sure a range can't be empty
  if (range.IsEmpty()) {
    range.Add(0, 0);
  }

  MusicCollection::Instance()->UpdateTrack(track);

}

/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void MusicTrackDialog::BuildMe() {

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxBoxSizer* item26 = new wxBoxSizer( wxVERTICAL );

  wxStaticBox *item3 = new wxStaticBox( this, -1, "Track details" );
  wxStaticBoxSizer *item2 = new wxStaticBoxSizer( item3, wxVERTICAL );

  wxFlexGridSizer *item4 = new wxFlexGridSizer( 2, 0, 0 );

  wxStaticText *item5 = new wxStaticText( this, -1, "Artist",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
```

```
  item4->Add( item5, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  artistTC = new wxTextCtrl( this, ID_ARTIST_TC,
    track.GetArtist().c_str(), wxDefaultPosition, wxSize(240,-1), 0 );
  item4->Add( artistTC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxStaticText *item7 = new wxStaticText( this, -1, "Title",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item4->Add( item7, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  titleTC = new wxTextCtrl( this, ID_TITLE_TC, track.GetTitle().c_str(),
    wxDefaultPosition, wxSize(160,-1), 0 );
  item4->Add( titleTC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxStaticText *item9 = new wxStaticText( this, -1, "Length",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item4->Add( item9, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxBoxSizer *item10 = new wxBoxSizer( wxHORIZONTAL );

  lengthTC = new wxTextCtrl( this, ID_LENGTH_TC,
    track.GetLengthString().c_str(), wxDefaultPosition, wxSize(38,-1), 0 );
  lengthTC->SetEditable(FALSE);
  wxColour shaded = wxSystemSettings::GetColour(wxSYS_COLOUR_BTNFACE);
  lengthTC->SetBackgroundColour(shaded);
  item10->Add( lengthTC, 0, wxALIGN_CENTER|wxALL, 5 );

//  item10->Add( 10, 10, 1, wxALIGN_CENTER|wxALL, 5 );

  wxStaticText *item12 = new wxStaticText( this, -1, "Format",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item10->Add( item12, 0, wxALIGN_RIGHT|wxALIGN_CENTER|wxALL, 5 );

  // Get all formats and put into drop down box
  const vector<string>& formatVec = MusicCollection::Instance()->GetFormats();
  size_t nFormats = formatVec.size();
  wxString* formatArr = new wxString[nFormats];
  for (uint i = 0; i < nFormats; ++i) {
    formatArr[i] = formatVec[i].c_str();
  }
  formatDD = new wxChoice( this, ID_FORMAT_DD, wxDefaultPosition,
    wxSize(50,-1), nFormats + 1, formatArr, 0 );
  wxString format(track.GetFormat().c_str());
  if (!format.IsEmpty()) {
    formatDD->SetStringSelection(format);
  }
  item10->Add( formatDD, 0, wxALIGN_CENTER|wxALL, 5 );
  delete[] formatArr;

  wxStaticText *item16 = new wxStaticText( this, -1, "Genre",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item10->Add( item16, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  // Get all genres and put into drop down box
  const vector<string>& genreVec =
    MusicCollection::Instance()->GetGenres();
  size_t nGenres = genreVec.size();
  wxString* genreArr = new wxString[nGenres];
  for (uint i = 0; i < nGenres; ++i) {
    genreArr[i] = genreVec[i].c_str();
  }
  genreDD = new wxChoice( this, ID_GENRE_DD, wxDefaultPosition,
    wxSize(85,-1), nGenres, genreArr, 0 );
  wxString genre(track.GetGenre().c_str());
  if (!genre.IsEmpty()) {
    genreDD->SetStringSelection(genre);
  }
  item10->Add( genreDD, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );
  delete[] genreArr;

  item4->Add( item10, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 0 );

  wxStaticText *item14 = new wxStaticText( this, -1, "Source",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item4->Add( item14, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  sourceTC = new wxTextCtrl( this, ID_SOURCE_TC,
    track.GetSource().c_str(), wxDefaultPosition, wxSize(160,-1), 0 );
  item4->Add( sourceTC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxStaticText *item18 = new wxStaticText( this, -1, wxT("Added"),
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT );
  item4->Add( item18, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  addedTC = new wxTextCtrl( this, ID_ADDED_TC,
     track.GetAddedOnString().c_str(), wxDefaultPosition, wxSize(160,-1));
  addedTC->SetEditable(FALSE);
  addedTC->SetBackgroundColour(shaded);
  item4->Add( addedTC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  item2->Add( item4, 0, wxALIGN_CENTER|wxALL, 5 );

  item26->Add( item2, 0, wxALIGN_LEFT|wxALIGN_TOP|wxALL, 5 );

  // Get hold of tempo range for track
  TempoRange range = track.GetTempoRange();

  item1->Add(item26, 0, wxGROW|wxALIGN_LEFT|wxALIGN_TOP|wxALL, 0);

  wxStaticBox *item21 = new wxStaticBox( this, -1, "Tempo range" );
  wxStaticBoxSizer *item20 = new wxStaticBoxSizer( item21, wxVERTICAL );

  // Get map out of tempo range
  IntShortMap& rangeMap = range.GetMap();
  size_t nTempi = rangeMap.size();

  // Create controls for tempo range display
  tempoGrid = new wxGrid(this, ID_TEMPO_GRID, wxDefaultPosition,
    wxSize(96, 150), wxWANTS_CHARS|wxSUNKEN_BORDER|wxTAB_TRAVERSAL);
  tempoGrid->CreateGrid(nTempi + 1, 2);
  tempoGrid->DisableDragColSize();
  tempoGrid->DisableDragRowSize();
  tempoGrid->DisableDragGridSize();
  tempoGrid->SetColFormatNumber(0);
  tempoGrid->SetColFormatNumber(1);
  tempoGrid->SetColLabelValue(0, "Time");
  tempoGrid->SetColLabelValue(1, "Tempo");
  tempoGrid->SetColLabelSize(19);
  tempoGrid->SetRowLabelSize(0);
  wxFont font = tempoGrid->GetLabelFont();
  font.SetWeight(wxNORMAL);
  tempoGrid->SetLabelFont(font);
  wxColour highlight = wxSystemSettings::GetColour(wxSYS_COLOUR_HIGHLIGHT);
  tempoGrid->SetGridLineColour(highlight);
  tempoGrid->SetCellHighlightColour(highlight);
  if (!track.GetVerified()) {
    tempoGrid->SetDefaultCellTextColour(Icons::Red());
  }
  // tempoGrid->SetExtraStyle(wxWS_EX_BLOCK_EVENTS);
  gridKeyHandler = new GridKeyHandler(this);
  tempoGrid->GetGridWindow()->PushEventHandler(gridKeyHandler);

  // Insert tempi in control
  IntShortMap::iterator it = rangeMap.begin();
  for (int i = 0; it != rangeMap.end(); ++it, ++i) {
    wxString time = MusicTrack::Secs2Time(it->first).c_str();
```

```
    tempoGrid->SetCellValue(i, 0, time);
    wxString tempo;
    tempo << it->second;
    tempoGrid->SetCellValue(i, 1, tempo);
  }
  tempoGrid->AutoSizeColumns(true);

  item20->Add(tempoGrid, 1, wxGROW|wxALIGN_CENTER_HORIZONTAL|wxLEFT|wxRIGHT|wxTOP,
10);

  // Grid toolbar
  wxBoxSizer *item24 = new wxBoxSizer( wxHORIZONTAL );
  item24->Add(4, 1, 0);

  verifyBut = new wxBitmapButton(this, ID_VERIFY_BUT,
    Icons::Verify(), wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  verifyBut->SetToolTip("Verify tempo range");
  item24->Add(verifyBut, 0, wxALIGN_LEFT|wxALL, 1);

  deleteBut = new wxBitmapButton(this, ID_DELETE_BUT,
    Icons::Delete(), wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  deleteBut->SetToolTip("Delete selected tempo rows");
  item24->Add(deleteBut, 0, wxALIGN_LEFT|wxALL, 1);

  item24->Add(4, 1, 0);

  tapBut = new wxBitmapButton(this, ID_TAP_BUT,
    Icons::Tap(), wxDefaultPosition, wxDefaultSize, wxBU_AUTODRAW);
  tapBut->SetToolTip("Tap selected tempo");
  item24->Add(tapBut, 0, wxALIGN_LEFT|wxALL, 1);

  item20->Add(item24, 0, wxALIGN_LEFT|wxALL, 5);

  item1->Add( item20, 1, wxGROW|wxALIGN_CENTER|wxALL, 5 );

  item0->Add( item1, 1, wxGROW|wxALIGN_CENTER|wxALL, 5 );

  wxStaticLine *item25 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL);
  item0->Add(item25, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  wxBoxSizer *item23 = new wxBoxSizer( wxHORIZONTAL );

  playBut = new wxButton(this, ID_PLAY_BUT, "Play", wxDefaultPosition,
    wxDefaultSize, 0);
  playBut->SetToolTip("Play track in default player program");
  string f = track.GetFormat();
  if (f != "WAV" && f != "MP3" && f != "OGG") {
    playBut->Disable();
  }
  item23->Add(playBut, 0, wxALIGN_LEFT|wxALL, 5);

  wxStaticLine *item27 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(-1,-1), wxLI_VERTICAL);
  item23->Add(item27, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  okBut = new wxButton( this, ID_OK_NOENTER, "OK", wxDefaultPosition,
    wxDefaultSize, 0 );
  okBut->SetDefault();
  item23->Add( okBut, 0, wxALIGN_CENTER|wxALL, 5 );

  cancelBut = new wxButton( this, ID_CANCEL_NOESC, "Cancel",
    wxDefaultPosition, wxDefaultSize, 0 );
  item23->Add( cancelBut, 0, wxALIGN_CENTER|wxALL, 5 );

  applyBut = new wxButton( this, wxID_APPLY, "Apply",
    wxDefaultPosition, wxDefaultSize, 0 );
  item23->Add( applyBut, 0, wxALIGN_CENTER|wxALL, 5 );

  item0->Add( item23, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  SetAutoLayout( TRUE );
  SetSizer( item0 );

  item0->Fit( this );
  item0->SetSizeHints( this );

}



/*****************************************************************************

  GridKeyHandler

  A key handler that will be pushed into the wxGrid owned by
  MusicTrackDialog. It's needed to capture key presses.

  Author: Erik Jälevik

*****************************************************************************/

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(GridKeyHandler, wxEvtHandler)
  EVT_CHAR(GridKeyHandler::OnChar)
END_EVENT_TABLE()

/*****************************************************************************
  OnChar
*****************************************************************************/
void GridKeyHandler::OnChar(wxKeyEvent& event) {

  // At the moment, this only seems to get called for keys that aren't caught
  // by the grid itself. Delete gets here but not Escape, Enter or arrows.
  // Normal key presses get here but they also open the cell editor even though
  // I don't call skip. Must mean the events don't get here first.

  if (event.GetKeyCode() == WXK_DELETE) {
    wxCommandEvent e;
    dialog->OnDelete(e);
  }

}
```

```
/*****************************************************************************

  Observable.h

  The abstract base class for all Observable objects.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef OBSERVABLE_H
#define OBSERVABLE_H

#include "Observer.h"

#include <list>

class Observable {

  public:

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~Observable() { }

    /*************************************************************************
      Register an observer.
    *************************************************************************/
    virtual void Attach(Observer* obs) { observers.push_back(obs); }

    /*************************************************************************
      Unregister an observer.
    *************************************************************************/
    virtual void Detach(Observer* obs) { observers.remove(obs); }

    /*************************************************************************
      Notify each registered observer by calling its Update method.
    *************************************************************************/
    virtual void Notify();


  protected:

    /*************************************************************************
      Constructor
    *************************************************************************/
    Observable() { }


  private:

    /*************************************************************************
      Variables
    *************************************************************************/
    std::list<Observer*> observers;

};

#endif
```

```
/*****************************************************************************

  Observable.cpp

  The abstract base class for all Observable objects.

  Author: Erik Jälevik

*****************************************************************************/

#include "Observable.h"

using namespace std;

/*****************************************************************************
  Notify
*****************************************************************************/
void Observable::Notify() {

  list<Observer*>::iterator iter = observers.begin();
  while (iter != observers.end()) {
    (*iter)->Update();
    ++iter;
  }

}
```

```
/**************************************************************************

  Observer.h

  The abstract base class for all Observer objects.

  Author: Erik Jälevik

**************************************************************************/

#ifndef OBSERVER_H
#define OBSERVER_H

class Observable; // just tell Observer that an Observable class exists

class Observer {

  public:

    /************************************************************************
      Notify this observer to update its display.
    ************************************************************************/
    virtual void Update() = 0;

  protected:

    /************************************************************************
      Constructor
    ************************************************************************/
    Observer() { };

    /************************************************************************
      Destructor
    ************************************************************************/
    virtual ~Observer() { };

};

#endif
```

```
/**************************************************************************

  OptionsDialog.h

  Tabbed options window. Contains the class for the main notebook dialog
  (OptionsDialog) plus individual classes for the different panels that make
  up the tabs.

  Author: Erik Jälevik

**************************************************************************/

#ifndef OPTIONSDIALOG_H
#define OPTIONSDIALOG_H

#include "MusicCollection.h"

#include <wx/notebook.h>

class GeneralOptionsPanel;
class BeatTrackingOptionsPanel;
class DatabaseOptionsPanel;
class FreeDBOptionsPanel;

class OptionsDialog : public wxDialog {

  public:

    /************************************************************************
      Parameters are parent window and ID.
    ************************************************************************/
    OptionsDialog(wxWindow*, wxWindowID);

    /************************************************************************
      Destructor
    ************************************************************************/
    virtual ~OptionsDialog();

    /************************************************************************
      Handler for OK button.
    ************************************************************************/
    void OnOK(wxCommandEvent&);

    /************************************************************************
      Handler for Reset button.
    ************************************************************************/
    void OnApply(wxCommandEvent&);

    /************************************************************************
      Handler for OK button.
    ************************************************************************/
    void OnCancel(wxCommandEvent&);

    /************************************************************************
      Save current values in all individual tabs.
    ************************************************************************/
    void Save();


  private:

    /************************************************************************
      Autogenerated method for building dialog
    ************************************************************************/
    void BuildMe();


    /************************************************************************
```

```
    Variables
  *****************************************************************************/

  // Controls
  wxNotebook* noteBook;
  wxButton* okBut;
  wxButton* cancelBut;
  wxButton* applyBut;

  GeneralOptionsPanel* generalTab;
  BeatTrackingOptionsPanel* beatTrackingTab;
  DatabaseOptionsPanel* databaseTab;
  FreeDBOptionsPanel* freeDBTab;

  DECLARE_EVENT_TABLE()

};


/*****************************************************************************

  GeneralOptionsPanel

  Panel for the general options.

*****************************************************************************/
class GeneralOptionsPanel : public wxPanel {

  public:

    /*************************************************************************
      Parameters are parent window and ID.
    *************************************************************************/
    GeneralOptionsPanel(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~GeneralOptionsPanel() { }

    /*************************************************************************
      Handler for Browse button.
    *************************************************************************/
    void OnBrowse(wxCommandEvent&);

    /*************************************************************************
      Save current values in panel fields.
    *************************************************************************/
    void Save();


  private:

    /*************************************************************************
      Autogenerated method for building panel
    *************************************************************************/
    void BuildMe();


    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxTextCtrl* playerTC;
    wxButton* browseBut;

    enum {
      ID_PLAYER_TC,
      ID_BROWSE_BUT
    };

    DECLARE_EVENT_TABLE()

};


/*****************************************************************************

  BeatTrackingOptionsPanel

  Panel for the beat tracking options.

*****************************************************************************/
class BeatTrackingOptionsPanel : public wxPanel {

  public:

    /*************************************************************************
      Parameters are parent window and ID.
    *************************************************************************/
    BeatTrackingOptionsPanel(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructors
    *************************************************************************/
    virtual ~BeatTrackingOptionsPanel() { }

    /*************************************************************************
      Handler for checking whether to disable the tempo interval option.
    *************************************************************************/
    void OnRadioChange(wxCommandEvent&);

    /*************************************************************************
      Save current values in panel fields.
    *************************************************************************/
    void Save();


  private:

    /*************************************************************************
      Autogenerated method for building panel
    *************************************************************************/
    void BuildMe();


    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxChoice* algorithmDD;
    wxSlider* accuracySl;
    wxRadioButton* trackChangesRB;
    wxRadioButton* useAverageRB;

    enum {
      ID_ALGORITHM_DD,
      ID_ACCURACY_SL,
      ID_TRACKCHANGES_RB,
      ID_USEAVERAGE_RB
    };

    DECLARE_EVENT_TABLE()
```

```
};

/*****************************************************************************

  DatabaseOptionsPanel

  Panel for the database options.

*****************************************************************************/
class DatabaseOptionsPanel : public wxPanel {

  public:

    /*************************************************************************
      Parameters are parent window and ID.
    *************************************************************************/
    DatabaseOptionsPanel(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~DatabaseOptionsPanel() { }

    /*************************************************************************
      Handler for browsing to a database file.
    *************************************************************************/
    void OnBrowse(wxCommandEvent&);

    /*************************************************************************
      Handler for adding new genre.
    *************************************************************************/
    void OnNewGenre(wxCommandEvent&);

    /*************************************************************************
      Handler for deleting genre.
    *************************************************************************/
    void OnDeleteGenre(wxCommandEvent&);

    /*************************************************************************
      Handler for adding new formats.
    *************************************************************************/
    void OnNewFormat(wxCommandEvent&);

    /*************************************************************************
      Handler for deleting format.
    *************************************************************************/
    void OnDeleteFormat(wxCommandEvent&);

    /*************************************************************************
      Save current values in panel fields.
    *************************************************************************/
    void Save();


  private:

    /*************************************************************************
      Autogenerated method for building panel
    *************************************************************************/
    void BuildMe();

    /*************************************************************************
      Does the work of changing the database file.
    *************************************************************************/
    void SetDatabase(wxString&);

    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxTextCtrl* dbFileTC;
    wxButton* browseBut;
    wxListBox* genreLB;
    wxButton* newGenreBut;
    wxButton* deleteGenreBut;
    wxListBox* formatLB;
    wxButton* newFormatBut;
    wxButton* deleteFormatBut;

    // Model objects
    MusicCollection* coll;

    wxString dbFile; // active database file name

    std::vector<wxString> newGenres;      // vectors keeping
    std::vector<wxString> deletedGenres;  // track of
    std::vector<wxString> newFormats;     // changes to
    std::vector<wxString> deletedFormats; // genres and formats

    enum {
      ID_DBFILE_TC,
      ID_BROWSE_BUT,
      ID_GENRE_LB,
      ID_NEWGENRE_BUT,
      ID_DELETEGENRE_BUT,
      ID_FORMAT_LB,
      ID_NEWFORMAT_BUT,
      ID_DELETEFORMAT_BUT
    };

    DECLARE_EVENT_TABLE()

};


/*****************************************************************************

  FreeDBOptionsPanel

  Panel for the FreeDB options.

*****************************************************************************/
class FreeDBOptionsPanel : public wxPanel {

  public:

    /*************************************************************************
      Parameters are parent window and ID.
    *************************************************************************/
    FreeDBOptionsPanel(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~FreeDBOptionsPanel() { }

    /*************************************************************************
      Handler for updating fields on server change.
    *************************************************************************/
    void OnServerChange(wxCommandEvent&);

    /*************************************************************************
      Handler for Get list button.
```

```
    ***************************************************************************/
    void OnGetList(wxCommandEvent&);

    /***************************************************************************
      Save current values in panel fields.
    ***************************************************************************/
    void Save();


  private:

    /***************************************************************************
      Autogenerated method for building panel
    ***************************************************************************/
    void BuildMe();


    /***************************************************************************
      Variables
    ***************************************************************************/

    // Controls
    wxTextCtrl* serverTC;
    wxButton* getListBut;
    wxTextCtrl* urlTC;
    wxTextCtrl* portTC;

    enum {
      ID_SERVER_TC,
      ID_GETLIST_BUT,
      ID_URL_TC,
      ID_PORT_TC
    };

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/***************************************************************************

  OptionsDialog.cpp

  Tabbed options window. Contains the class for the main notebook dialog
  (OptionsDialog) plus individual classes for the different panels that make
  up the tabs.

  Author: Erik Jälevik

***************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "OptionsDialog.h"
#include "FreeDB.h"

#include <wx/confbase.h>

#include <vector>

using namespace std;

/***************************************************************************
  Constructor
***************************************************************************/
OptionsDialog::OptionsDialog(wxWindow* parent, wxWindowID id) :
  wxDialog(parent, id, "Options", wxDefaultPosition, wxDefaultSize,
    wxCAPTION | wxSYSTEM_MENU | wxRESIZE_BORDER) {

  BuildMe();

}

/***************************************************************************
  Destructor
***************************************************************************/
OptionsDialog::~OptionsDialog() {

  wxConfigBase* cfg = wxConfigBase::Get();
  cfg->Write("/Options/tab", noteBook->GetSelection());

}

/***************************************************************************
  Event table
***************************************************************************/
BEGIN_EVENT_TABLE(OptionsDialog, wxDialog)
  EVT_BUTTON(wxID_OK, OptionsDialog::OnOK)
  EVT_BUTTON(wxID_CANCEL, OptionsDialog::OnCancel)
  EVT_BUTTON(wxID_APPLY, OptionsDialog::OnApply)
END_EVENT_TABLE()

/***************************************************************************
  OnOK
***************************************************************************/
void OptionsDialog::OnOK(wxCommandEvent& event) {

  Save();
```

```
  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************************
  OnCancel
*********************************************************************************/
void OptionsDialog::OnCancel(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************************
  OnApply
*********************************************************************************/
void OptionsDialog::OnApply(wxCommandEvent& event) {

  Save();

}

/*********************************************************************************
  Save
*********************************************************************************/
void OptionsDialog::Save() {

  wxBusyCursor busy;

  generalTab->Save();
  beatTrackingTab->Save();
  databaseTab->Save();
  freeDBTab->Save();

}

/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void OptionsDialog::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();
  long tabVal = cfg->Read("/Options/tab", 0.0);

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  noteBook = new wxNotebook(this, -1, wxDefaultPosition,
    wxSize(200, 160), 0);
  wxNotebookSizer *item1 = new wxNotebookSizer(noteBook);

  generalTab = new GeneralOptionsPanel(noteBook, -1);
  noteBook->AddPage(generalTab, "General");

  beatTrackingTab = new BeatTrackingOptionsPanel(noteBook, -1);
  noteBook->AddPage(beatTrackingTab, "Beat tracking");

  databaseTab = new DatabaseOptionsPanel(noteBook, -1);
  noteBook->AddPage(databaseTab, "Database");

  freeDBTab = new FreeDBOptionsPanel(noteBook, -1);
  noteBook->AddPage(freeDBTab, "FreeDB");

  noteBook->SetSelection(tabVal);
  item0->Add(item1, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT|wxTOP, 5);

  wxBoxSizer *item6 = new wxBoxSizer( wxHORIZONTAL );

  okBut = new wxButton(this, wxID_OK, "OK", wxDefaultPosition,
    wxDefaultSize, 0);
  item6->Add(okBut, 0, wxALIGN_CENTER|wxALL, 5);

  cancelBut = new wxButton(this, wxID_CANCEL, "Cancel", wxDefaultPosition,
    wxDefaultSize, 0);
  item6->Add(cancelBut, 0, wxALIGN_CENTER|wxALL, 5);

  applyBut = new wxButton(this, wxID_APPLY, "Apply", wxDefaultPosition,
    wxDefaultSize, 0);
  item6->Add(applyBut, 0, wxALIGN_CENTER|wxALL, 5);

  item0->Add(item6, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}


/*********************************************************************************

  GeneralOptionsPanel

  Panel for the general options.

*********************************************************************************/

/*********************************************************************************
  Constructor
*********************************************************************************/
GeneralOptionsPanel::GeneralOptionsPanel(wxWindow* parent,
                                         wxWindowID id) :
  wxPanel(parent, id, wxDefaultPosition, wxDefaultSize) {

  BuildMe();

}

/*********************************************************************************
  Event table
*********************************************************************************/
BEGIN_EVENT_TABLE(GeneralOptionsPanel, wxPanel)
  EVT_BUTTON(ID_BROWSE_BUT, GeneralOptionsPanel::OnBrowse)
END_EVENT_TABLE()

/*********************************************************************************
  OnBrowse
*********************************************************************************/
void GeneralOptionsPanel::OnBrowse(wxCommandEvent& event) {
```

```
  wxString plr = ::wxFileSelector("Select player", "", "", "",
    "Programs (*.exe)|*.exe|All files (*.*)|*.*",
    wxOPEN | wxFILE_MUST_EXIST | wxHIDE_READONLY);

  if (!plr.IsEmpty()) {
    playerTC->SetValue(plr);
  }

}

/*********************************************************************************
  Save
*********************************************************************************/
void GeneralOptionsPanel::Save() {

  wxConfigBase* cfg = wxConfigBase::Get();

  cfg->Write("/Options/General/player", playerTC->GetValue());

}

/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void GeneralOptionsPanel::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString player = cfg->Read("/Options/General/player", "");

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  item0->Add( 20, 5, 0, wxALIGN_CENTER|wxALL, 0 );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticText *item2 = new wxStaticText(this, -1, "Player program",
    wxDefaultPosition, wxDefaultSize, 0);
  item2->SetToolTip("The program to launch when Play is pressed in the "
    "Track Info window");
  item1->Add(item2, 0, wxALIGN_CENTER|wxALL, 5);

  playerTC = new wxTextCtrl(this, ID_PLAYER_TC, player, wxDefaultPosition,
    wxSize(200,-1), 0);
  item1->Add(playerTC, 0, wxALIGN_CENTER|wxALL, 5);

  browseBut = new wxButton(this, ID_BROWSE_BUT, "Browse...",
    wxDefaultPosition, wxDefaultSize, 0);
  item1->Add(browseBut, 0, wxALIGN_CENTER|wxALL, 5);

  item0->Add(item1, 0, wxALIGN_CENTER|wxLEFT|wxRIGHT|wxTOP, 5);

  SetAutoLayout( TRUE );
  SetSizer( item0 );
  item0->Fit( this );
  item0->SetSizeHints( this );

}


/*********************************************************************************

  BeatTrackingOptionsPanel

  Panel for the beat tracking options.

*********************************************************************************/

/*********************************************************************************
  Constructor
*********************************************************************************/
BeatTrackingOptionsPanel::BeatTrackingOptionsPanel(wxWindow* parent,
                                                   wxWindowID id) :
  wxPanel(parent, id, wxDefaultPosition, wxDefaultSize) {

  BuildMe();

}

/*********************************************************************************
  Event table
*********************************************************************************/
BEGIN_EVENT_TABLE(BeatTrackingOptionsPanel, wxPanel)
  EVT_RADIOBUTTON(ID_TRACKCHANGES_RB, BeatTrackingOptionsPanel::OnRadioChange)
  EVT_RADIOBUTTON(ID_USEAVERAGE_RB, BeatTrackingOptionsPanel::OnRadioChange)
END_EVENT_TABLE()

/*********************************************************************************
  OnRadioChange
*********************************************************************************/
void BeatTrackingOptionsPanel::OnRadioChange(wxCommandEvent& event) {

}

/*********************************************************************************
  Save
*********************************************************************************/
void BeatTrackingOptionsPanel::Save() {

  wxConfigBase* cfg = wxConfigBase::Get();

  cfg->Write("/Options/BeatTracking/algorithm", algorithmDD->GetSelection());
  cfg->Write("/Options/BeatTracking/accuracy", accuracySl->GetValue());
  cfg->Write("/Options/BeatTracking/trackChanges", trackChangesRB->GetValue());

}

/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void BeatTrackingOptionsPanel::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();

  long algorithmVal = cfg->Read("/Options/BeatTracking/algorithm", 0L);
  long accuracyVal = cfg->Read("/Options/BeatTracking/accuracy", 3L);
  bool trackChangesVal;
  cfg->Read("/Options/BeatTracking/trackChanges", &trackChangesVal,
    true);

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxVERTICAL );

  item1->Add( 20, 5, 0, wxALIGN_CENTER|wxALL, 0 );

  wxBoxSizer *item2 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticText *item3 = new wxStaticText(this, -1, "Algorithm",
    wxDefaultPosition, wxDefaultSize, 0);
  item2->Add(item3, 0, wxALIGN_CENTER|wxALL, 5);

  wxString strs4[] = {"Scheirer"};
  algorithmDD = new wxChoice(this, ID_ALGORITHM_DD, wxDefaultPosition,
```

```
    wxSize(160, -1), 1, strs4, 0);
  algorithmDD->SetSelection(algorithmVal);
  item2->Add(algorithmDD, 0, wxALIGN_CENTER|wxALL, 5);

  item1->Add(item2, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 0);

  wxBoxSizer *item5 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticText *item6 = new wxStaticText(this, -1, "Accuracy",
    wxDefaultPosition, wxDefaultSize, 0);
  item5->Add(item6, 0, wxALIGN_CENTER|wxALL, 5);

  accuracySl = new wxSlider(this, ID_ACCURACY_SL, accuracyVal, 1, 5,
    wxDefaultPosition, wxSize(100, -1),
    wxSL_HORIZONTAL | wxSL_AUTOTICKS);
  item5->Add(accuracySl, 0, wxALIGN_CENTER|wxALL, 5);

  item1->Add(item5, 0, wxALIGN_CENTER_VERTICAL|wxALL, 0);

  item1->Add(20, 5, 0, wxALIGN_CENTER_VERTICAL|wxALL, 0);

  trackChangesRB = new wxRadioButton(this, ID_TRACKCHANGES_RB,
    "Track tempo changes", wxDefaultPosition, wxDefaultSize, wxRB_GROUP);
  trackChangesRB->SetValue(trackChangesVal);
  item1->Add(trackChangesRB, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

  useAverageRB = new wxRadioButton(this, ID_USEAVERAGE_RB,
    "Use overall tempo", wxDefaultPosition, wxDefaultSize, 0);
  useAverageRB->SetValue(!trackChangesVal);
  item1->Add(useAverageRB, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

  item1->Add(20, 5, 0, wxALIGN_CENTER_VERTICAL|wxALL, 0);

  item0->Add(item1, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5);

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}


/*********************************************************************************

  DatabaseOptionsPanel

  Panel for the database options.

*********************************************************************************/

/*********************************************************************************
  Constructor
*********************************************************************************/
DatabaseOptionsPanel::DatabaseOptionsPanel(wxWindow* parent,
                                           wxWindowID id) :
  wxPanel(parent, id, wxDefaultPosition, wxDefaultSize),
  dbFile("") {

  coll = MusicCollection::Instance();

  BuildMe();

}

/*********************************************************************************
  Event table
*********************************************************************************/
BEGIN_EVENT_TABLE(DatabaseOptionsPanel, wxPanel)
  EVT_BUTTON(ID_BROWSE_BUT, DatabaseOptionsPanel::OnBrowse)
  EVT_BUTTON(ID_NEWGENRE_BUT, DatabaseOptionsPanel::OnNewGenre)
  EVT_BUTTON(ID_DELETEGENRE_BUT, DatabaseOptionsPanel::OnDeleteGenre)
  EVT_BUTTON(ID_NEWFORMAT_BUT, DatabaseOptionsPanel::OnNewFormat)
  EVT_BUTTON(ID_DELETEFORMAT_BUT, DatabaseOptionsPanel::OnDeleteFormat)
END_EVENT_TABLE()

/*********************************************************************************
  OnBrowse
*********************************************************************************/
void DatabaseOptionsPanel::OnBrowse(wxCommandEvent& event) {

  wxString newDB = ::wxFileSelector("Select database", "", "", "",
    "Database files (*.db)|*.db|All files (*.*)|*.*",
    wxOPEN | wxFILE_MUST_EXIST | wxHIDE_READONLY);

  if (!newDB.IsEmpty()) {
    dbFileTC->SetValue(newDB);
  }

}

/*********************************************************************************
  OnNewGenre
*********************************************************************************/
void DatabaseOptionsPanel::OnNewGenre(wxCommandEvent& event) {

  wxString genre = ::wxGetTextFromUser("Please enter a genre", "New Genre");
  if (!genre.IsEmpty()) {
    genreLB->Append(genre);
    newGenres.push_back(genre);
  }

}

/*********************************************************************************
  OnDeleteGenre
*********************************************************************************/
void DatabaseOptionsPanel::OnDeleteGenre(wxCommandEvent& event) {

  // This way of deletion is necessary since indices change when deleting
  for (int i = 0; i < genreLB->GetCount(); ) {

    if (genreLB->Selected(i)) {

      wxString genre = genreLB->GetString(i);

      if (coll->GenreInUse(genre.c_str())) {
        wxString msg;
        msg << "The genre '" << genre << "' can't be deleted because there "
          "are tracks in the collection of that genre. If you want to delete "
          "a genre you need to make sure there are no tracks in the "
          "collection using that genre.";
        ::wxMessageBox(msg, "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
        break;
      }
      else {
        deletedGenres.push_back(genre);
        genreLB->Delete(i);
      }

    }
    else {
      ++i;
    }

  }
```

```
}

/*****************************************************************************
  OnNewFormat
*****************************************************************************/
void DatabaseOptionsPanel::OnNewFormat(wxCommandEvent& event) {

  wxString format = ::wxGetTextFromUser("Please enter a format", "New Format");
  if (!format.IsEmpty()) {
    formatLB->Append(format);
    newFormats.push_back(format);
  }

}

/*****************************************************************************
  OnDeleteFormat
*****************************************************************************/
void DatabaseOptionsPanel::OnDeleteFormat(wxCommandEvent& event) {

  // This way of deletion is necessary since indices change when deleting
  for (int i = 0; i < formatLB->GetCount(); ) {

    if (formatLB->Selected(i)) {

      wxString format = formatLB->GetString(i);

     if (coll->FormatInUse(format.c_str())) {
        wxString msg;
        msg << "The format '" << format << "' can't be deleted because there "
          "are tracks in the collection of that format. If you want to delete "
          "a format you need to make sure there are no tracks in the "
          "collection using that format.";
        ::wxMessageBox(msg, "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
        break;
      }
      else {
        deletedFormats.push_back(format);
        formatLB->Delete(i);
      }
    }

    else {
      ++i;
    }

  }

}

/*****************************************************************************
  Save
*****************************************************************************/
void DatabaseOptionsPanel::Save() {

  wxString newDB = dbFileTC->GetValue();

  // Check if db file has changed
  if (newDB != dbFile) {
    SetDatabase(newDB);
  }

  try {

    // Add genres
    if (!newGenres.empty()) {
      for (size_t i = 0; i < newGenres.size(); ++i) {
        coll->AddGenre(newGenres.at(i).c_str());
      }
    }

    // Delete genres
    if (!deletedGenres.empty()) {
      for (size_t i = 0; i < deletedGenres.size(); ++i) {
        coll->DeleteGenre(deletedGenres.at(i).c_str());
      }
    }

    // Add formats
    if (!newFormats.empty()) {
      for (size_t i = 0; i < newFormats.size(); ++i) {
        coll->AddFormat(newFormats.at(i).c_str());
      }
    }

    // Delete formats
    if (!deletedFormats.empty()) {
      for (size_t i = 0; i < deletedFormats.size(); ++i) {
        coll->DeleteFormat(deletedFormats.at(i).c_str());
      }
    }

  }
  catch (MusicCollection::DatabaseException& e) {
    ::wxMessageBox(e.what(), "Database Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

  newGenres.clear();
  deletedGenres.clear();
  newFormats.clear();
  deletedFormats.clear();

}

/*****************************************************************************
  SetDatabase
*****************************************************************************/
void DatabaseOptionsPanel::SetDatabase(wxString& newDB) {

  // Check that name isn't empty
  if (!newDB.IsEmpty()) {

    try {

      coll->Open(newDB.c_str());

      // If successfully opened, we save new db in config
      wxConfigBase* cfg = wxConfigBase::Get();
      cfg->Write("/Options/Database/file", newDB);
      dbFile = newDB;

    }
    catch (MusicCollection::DatabaseException& e) {

      // Immediately restore old database
      // TODO: this could throw as well, what would happen then?
      coll->Open(dbFile.c_str());

      ::wxMessageBox(e.what(), "Database Error",
        wxOK | wxICON_ERROR | wxCENTRE);

    }

  }
```

```
}

/*****************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*****************************************************************************/
void DatabaseOptionsPanel::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();
  dbFile = cfg->Read("/Options/Database/file");

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  item0->Add( 20, 5, 0, wxALIGN_CENTER|wxALL, 0 );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticText *item2 = new wxStaticText(this, -1, "Database file",
    wxDefaultPosition, wxDefaultSize, 0);
  item1->Add(item2, 0, wxALIGN_CENTER|wxALL, 5);

  dbFileTC = new wxTextCtrl(this, ID_DBFILE_TC, dbFile,
    wxDefaultPosition, wxSize(200,-1), 0);
  item1->Add(dbFileTC, 1, wxALIGN_CENTER|wxALL, 5);

  browseBut = new wxButton(this, ID_BROWSE_BUT, "Browse...",
    wxDefaultPosition, wxDefaultSize, 0);
  item1->Add(browseBut, 0, wxALIGN_CENTER|wxALL, 5);
  browseBut->SetFocus(); // this is to stop the dbFileTC's weird behaviour of
                         // only showing the last char when receiving focus

  item0->Add(item1, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT|wxTOP, 5);

  wxBoxSizer *item5 = new wxBoxSizer( wxHORIZONTAL );

  wxStaticBox *item7 = new wxStaticBox(this, -1, "Genres");
  wxStaticBoxSizer *item6 = new wxStaticBoxSizer( item7, wxHORIZONTAL );

  // Get all genres and put into list box
  const vector<string>& genreVec =
    MusicCollection::Instance()->GetGenres();
  size_t nGenres = genreVec.size();
  wxString* genreArr = new wxString[nGenres];
  for (uint i = 0; i < nGenres; ++i) {
    genreArr[i] = genreVec[i].c_str();
  }
  genreLB = new wxListBox(this, ID_GENRE_LB, wxDefaultPosition,
    wxSize(100, 140), nGenres, genreArr, wxLB_EXTENDED);
  item6->Add(genreLB, 0, wxALIGN_CENTER|wxALL, 5);
  delete[] genreArr;

  wxBoxSizer *item9 = new wxBoxSizer( wxVERTICAL );

  newGenreBut = new wxButton(this, ID_NEWGENRE_BUT, "New...",
    wxDefaultPosition, wxSize(50, -1), 0);
  item9->Add(newGenreBut, 0, wxALIGN_CENTER|wxALL, 5);

  deleteGenreBut = new wxButton(this, ID_DELETEGENRE_BUT, "Delete",
    wxDefaultPosition, wxSize(50, -1), 0);
  item9->Add(deleteGenreBut, 0, wxALIGN_CENTER|wxALL, 5);

  item6->Add( item9, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 0 );

  item5->Add( item6, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticBox *item13 = new wxStaticBox(this, -1, "Formats");
  wxStaticBoxSizer *item12 = new wxStaticBoxSizer(item13, wxHORIZONTAL);

  // Get all formats and put into list box
  const vector<string>& formatVec =
    MusicCollection::Instance()->GetFormats();
  size_t nFormats = formatVec.size();
  wxString* formatArr = new wxString[nFormats];
  for (uint i = 0; i < nFormats; ++i) {
    formatArr[i] = formatVec[i].c_str();
  }
  formatLB = new wxListBox(this, ID_FORMAT_LB, wxDefaultPosition,
    wxSize(100, 140), nFormats, formatArr, wxLB_EXTENDED);
  item12->Add(formatLB, 0, wxALIGN_CENTER|wxALL, 5);
  delete[] formatArr;

  wxBoxSizer *item15 = new wxBoxSizer( wxVERTICAL );

  newFormatBut = new wxButton(this, ID_NEWFORMAT_BUT, "New...",
    wxDefaultPosition, wxSize(50, -1), 0);
  item15->Add(newFormatBut, 0, wxALIGN_CENTER|wxALL, 5);

  deleteFormatBut = new wxButton(this, ID_DELETEFORMAT_BUT, "Delete",
    wxDefaultPosition, wxSize(50, -1), 0);
  item15->Add(deleteFormatBut, 0, wxALIGN_CENTER|wxALL, 5);

  item12->Add( item15, 0, wxALIGN_CENTER_HORIZONTAL|wxALL, 0 );

  item5->Add( item12, 0, wxALIGN_CENTER|wxALL, 5 );

  item0->Add( item5, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}


/*****************************************************************************

  FreeDBOptionsPanel

  Panel for the FreeDB options.

*****************************************************************************/

/*****************************************************************************
  Constructor
*****************************************************************************/
FreeDBOptionsPanel::FreeDBOptionsPanel(wxWindow* parent,
                                       wxWindowID id) :
  wxPanel(parent, id, wxDefaultPosition, wxDefaultSize) {

  BuildMe();

}

/*****************************************************************************
  Event table
*****************************************************************************/
BEGIN_EVENT_TABLE(FreeDBOptionsPanel, wxPanel)
  EVT_BUTTON(ID_GETLIST_BUT, FreeDBOptionsPanel::OnGetList)
END_EVENT_TABLE()

/*****************************************************************************
  OnGetList
*****************************************************************************/
void FreeDBOptionsPanel::OnGetList(wxCommandEvent& event) {
```

```
  ::wxBeginBusyCursor();

  auto_ptr<vector<FreeDBSite> > sites;
  try {
    FreeDB db;
    sites = db.GetSites();
  }
  catch (FreeDB::FreeDBException& e) {
    ::wxEndBusyCursor();
    ::wxMessageBox(e.what(), "FreeDB Error", wxOK | wxICON_ERROR | wxCENTRE);
  }

  size_t nSites = sites->size();

  wxString* sitesArr = new wxString[nSites];
  for (uint i = 0; i < nSites; ++i) {
    sitesArr[i] = sites->at(i).server.c_str();
  }

  wxSingleChoiceDialog dlg(this,
    "The following servers were found. Please select one.",
    "FreeDB Servers",
    nSites, sitesArr, NULL,
    wxOK | wxCANCEL | wxCENTRE | wxCAPTION);
  dlg.SetSelection(0);
  delete[] sitesArr;

  ::wxEndBusyCursor();

  // If OK, fill fields from chosen server
  if (dlg.ShowModal() == wxID_OK) {
    int choice = dlg.GetSelection();
    serverTC->SetValue(sites->at(choice).server.c_str());
    urlTC->SetValue(sites->at(choice).url.c_str());
    wxString port;
    port << sites->at(choice).port;
    portTC->SetValue(port);
  }

}

/*********************************************************************************
  Save
*********************************************************************************/
void FreeDBOptionsPanel::Save() {

  wxConfigBase* cfg = wxConfigBase::Get();
  cfg->Write("Options/FreeDB/server", serverTC->GetValue());
  cfg->Write("Options/FreeDB/url", urlTC->GetValue());
  long port;
  portTC->GetValue().ToLong(&port);
  cfg->Write("Options/FreeDB/port", port);

}

/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void FreeDBOptionsPanel::BuildMe() {

  wxConfigBase* cfg = wxConfigBase::Get();
  wxString serverVal = cfg->Read("/Options/FreeDB/server", "www.freedb.org");
  wxString urlVal = cfg->Read("/Options/FreeDB/url",
    "www.freedb.org/~cddb/cddb.cgi");
  long portVal = cfg->Read("/Options/FreeDB/port", 80L);

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  item0->Add( 20, 5, 0, wxALIGN_CENTER|wxALL, 0 );

  wxFlexGridSizer *item1 = new wxFlexGridSizer( 2, 0, 0 );

  wxStaticText *item2 = new wxStaticText(this, -1, "Server",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT);
  item1->Add(item2, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  wxBoxSizer *item3 = new wxBoxSizer( wxHORIZONTAL );

  serverTC = new wxTextCtrl(this, ID_SERVER_TC, serverVal, wxDefaultPosition,
    wxSize(100, -1), 0);
  item3->Add(serverTC, 1, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  getListBut = new wxButton(this, ID_GETLIST_BUT, "Get list...",
    wxDefaultPosition, wxSize(69, -1), 0);
  getListBut->SetToolTip("Download list of available FreeDB servers");
  item3->Add(getListBut, 0, wxALIGN_CENTER|wxALL, 5);

  item1->Add( item3, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 0 );

  wxStaticText *item6 = new wxStaticText(this, -1, "URL",
    wxDefaultPosition, wxDefaultSize, wxALIGN_RIGHT);
  item1->Add( item6, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  wxBoxSizer *item7 = new wxBoxSizer( wxHORIZONTAL );

  urlTC = new wxTextCtrl(this, ID_URL_TC, urlVal, wxDefaultPosition,
    wxSize(200, -1), 0);
  item7->Add(urlTC, 0, wxALIGN_CENTER|wxALL, 5);

  wxStaticText *item9 = new wxStaticText(this, -1, "Port", wxDefaultPosition,
    wxDefaultSize, wxALIGN_RIGHT);
  item7->Add(item9, 0, wxALIGN_CENTER|wxALL, 5);

  wxString port;
  port << portVal;
  portTC = new wxTextCtrl(this, ID_PORT_TC, port, wxDefaultPosition,
    wxSize(40, -1), 0, wxTextValidator(wxFILTER_NUMERIC));
  item7->Add(portTC, 0, wxALIGN_CENTER|wxALL, 5);

  item1->Add(item7, 0, wxALIGN_CENTER|wxALL, 0);

  item0->Add( item1, 0, wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}
```

```
/*****************************************************************************

  PhaseResonator.h

  A resonator that keeps track of phase constancy.

  The phase constancy is implemented as a simple first order recursive
  filter with the equation:

  constancy[t] = alpha * constancy[t - period] + (1 - alpha) * (1 - drift)

  Drift is the difference in phase between two subsequent periods.
  The alpha parameter controls the speed of change for the constancy.


  Author: Erik Jälevik

*****************************************************************************/

#ifndef PHASERESONATOR_H
#define PHASERESONATOR_H

#include "Resonator.h"
#include "Globals.h"

class PhaseResonator : public Resonator {

  public:

    /*************************************************************************
      Parameters:
      delay - delay in number of samples
      halfTime - time until half energy is reached when sent impulse response
    *************************************************************************/
    PhaseResonator(uint delay, double halfTime) :
      Resonator(delay, halfTime),
      lastTimeToBeat(0),
      phaseConstancy(0.0),
      lastConstancy(0.0) {
      phaseAlpha = pow(0.5, delay / halfTime);
    }

    /*************************************************************************
      Default constructor creates a resonator with delay 1, alpha 0.5 and
      phaseAlpha 0.5.
    *************************************************************************/
    PhaseResonator() :
      Resonator(),
      phaseAlpha(0.5),
      lastTimeToBeat(0),
      phaseConstancy(0.0),
      lastConstancy(0.0) { }

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~PhaseResonator() { }

    /*************************************************************************
      Input one value. Output is not returned, it's just stored in internal
      buffer. Phase constancy is also updated once per period.
    *************************************************************************/
    virtual void Tick(double input);

    /*************************************************************************
      Returns the energy of the resonator. Energy is weighted against phase
      constancy so that a resonator with high constancy would return more
      energy than one with low constancy.
    *************************************************************************/
    virtual double GetEnergy();

    /*************************************************************************
      Returns the phase constancy as a value between 0 and 1.
    *************************************************************************/
    virtual float GetPhaseConstancy() { return phaseConstancy; }


  protected:

    /*************************************************************************
      Variables
    *************************************************************************/
    double phaseAlpha; // alpha constant

    int lastTimeToBeat; // no of samples to beat last cycle

    float phaseConstancy; // current constancy
    float lastConstancy;  // constancy last cycle

};

#endif
```

```
/*******************************************************************************

  PhaseResonator.cpp

  A resonator that keeps track of phase constancy.

  The phase constancy is implemented as a simple first order recursive
  filter with the equation:

  constancy[t] = alpha * constancy[t - period] + (1 - alpha) * (1 - drift)

  Drift is the difference in phase between two subsequent periods.
  The alpha parameter controls the speed of change for the constancy.


  Author: Erik Jälevik

*******************************************************************************/

#include "PhaseResonator.h"
#include "Globals.h"

#include <cmath>

/*******************************************************************************
  Tick
*******************************************************************************/
void PhaseResonator::Tick(double input) {

  buffer[index] = alpha * buffer[index] + beta * input;

  // Wrap around buffer and do phase constancy calculation
  if (++index == length) {
    index = 0;

    // Find peak in delay line, i.e. phase
    double max = 0.0;
    int winner = -1;
    for (uint i = 0; i < length; ++i) {
      if (buffer[i] > max) {
        max = buffer[i];
        winner = i;
      }
    }

    if (winner != -1) {

      // Compare phase to last phase. Since we're always at index 0 when
      // we're doing this, we can just use the index of the winner directly
      // as the time to next beat.
      int drift = abs(lastTimeToBeat - winner);

      // Since we're using a circular buffer, we need to make sure that a
      // phase difference that crosses the buffer ends is properly calculated.
      // I.e if the winner is index 49 out of 50 and last winner was 0, we
      // should have a phase drift of 1, not 49.

      // By doing this several times we can also make a phase drift of half the
      // period or a quarter of the period not count as a drift. The
      // resonator's phase constancy should not be penalised if it shifts
      // the downbeat from a whole note to a quarter note for example.
      int i;
      for (i = 2; i < 3; i *= 2) { // i values of 2
        if (drift > (length / i)) {
          drift -= 2 * (drift - length / i);
        }
      }

      // Can happen due to rounding errors
      if (drift < 0) { drift = 0; }

      // Convert drift to a range of 0 - 1.
      float driftf = (i / 2) * (static_cast<float>(drift) / length);

      // Square to make drastic changes count more. This gave worse
      // results. Seems to boost the bad ones instead.
      //driftf *= driftf;

      // Update phaseConstancy
      phaseConstancy = phaseAlpha * lastConstancy +
                       (1.0 - phaseAlpha) * (1.0 - driftf);

      lastTimeToBeat = winner;
      lastConstancy = phaseConstancy;

    }

  }

}

/*******************************************************************************
  GetEnergy
*******************************************************************************/
double PhaseResonator::GetEnergy() {

  double pwr = 0.0;
  double sum = 0.0;

  // Step through the delay line adding up its power
  for (uint i = 0; i < length ; ++i) {
    pwr += buffer[i] * buffer[i];
  }
  pwr = sqrt(pwr);

  for (uint i = 0; i < length; ++i) {
    sum += pwr - buffer[i];
  }

  sum = sum / length;

  return phaseConstancy * sum;

}
```

```
/*****************************************************************************

  ProgressDialog.h

  GUI progress dialog for beat tracking.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef PROGRESSDIALOG_H
#define PROGRESSDIALOG_H

#include "BeatTracker.h"
#include "Observer.h"
#include "MusicTrack.h"

#include <wx/listctrl.h>

#include <vector>
#include <memory>

class ProgressDialog : public wxDialog, public Observer {

  public:

    /*************************************************************************
      Parameters are parent window, beat tracker and a vector of tracks to
      schedule and pass to the tracker.
    *************************************************************************/
    ProgressDialog(wxWindow*,
                   BeatTracker&,
                   std::auto_ptr<std::vector<MusicTrack> >);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~ProgressDialog();

    /*************************************************************************
      Handler for Stop button.
    *************************************************************************/
    void OnStop(wxCommandEvent&);

    /*************************************************************************
      Start tracking.
    *************************************************************************/
    void Run();

    /*************************************************************************
      Observer method. Called by beat tracker.
    *************************************************************************/
    virtual void Update();


  protected:

    /*************************************************************************
      Updates display to show next track as current.
    *************************************************************************/
    void DisplayNext();

    /*************************************************************************
      Updates status of track in schedule list control.
    *************************************************************************/
    void DisplayStatus(const char*);

    /*************************************************************************
      Check if any errors occurred and display them to user as a log.
    *************************************************************************/
    void CheckErrors();


  private:

    /*************************************************************************
      Autogenerated method for building dialog
    *************************************************************************/
    void BuildMe();


    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxStaticText* currentST;
    wxGauge* gauge;
    wxListCtrl* scheduleLC;
    wxButton* stopBut;


    // Model objects
    BeatTracker* tracker;  // tracker to use
    std::auto_ptr<std::vector<MusicTrack> > tracks; // schedule to beat track
    int current; // index into list ctrl of currently beat tracked track

    wxString errorLog; // errors are logged to this string

    bool stopped; // gets set if user stops beat tracking

    enum {
      ID_CURRENT_ST,
      ID_GAUGE,
      ID_SCHEDULE_LC
    };

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/**********************************************************************

  ProgressDialog.cpp

  GUI progress dialog for when beat tracking.

  Author: Erik Jälevik

**********************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "ProgressDialog.h"
#include "MusicCollection.h"
#include "LogDialog.h"
#include "Globals.h"

#include <wx/statline.h>
#include <wx/confbase.h>

using namespace std;

/*********************************************************************
  Constructor
*********************************************************************/
ProgressDialog::ProgressDialog(wxWindow* parent,
                               BeatTracker& trker,
                               auto_ptr<vector<MusicTrack> > trks) :
  wxDialog(parent,
           -1,
           "Progress",
           wxDefaultPosition,
           wxDefaultSize,
           wxCAPTION | wxSYSTEM_MENU),
  tracker(&trker),
  tracks(trks),
  current(-1),
  stopped(false),
  errorLog("") {

  BuildMe();

  // Register as observer
  tracker->Attach(this);

}

/*********************************************************************
  Destructor
*********************************************************************/
ProgressDialog::~ProgressDialog() {

  wxConfigBase* cfg = wxConfigBase::Get();
  wxPoint pos = GetPosition();
  cfg->Write("/ProgressDialog/xpos", pos.x);
  cfg->Write("/ProgressDialog/ypos", pos.y);

  tracker->Detach(this);

}

/*********************************************************************
  Event table
*********************************************************************/
BEGIN_EVENT_TABLE(ProgressDialog, wxDialog)
  EVT_BUTTON(wxID_CANCEL, ProgressDialog::OnStop)
END_EVENT_TABLE()

/*********************************************************************
  OnStop
*********************************************************************/
void ProgressDialog::OnStop(wxCommandEvent& event) {

  wxString msg = "Are you sure you want to stop beat tracking?";
  int ans = ::wxMessageBox(msg, "Confirm Stop", wxYES_NO | wxICON_QUESTION);
  if (ans == wxYES) {
    tracker->Stop();
    stopped = true;
  }

}

/*********************************************************************
  Run
*********************************************************************/
void ProgressDialog::Run() {

  Show(true);

  // Disable all windows but this one for duration of method
  wxWindowDisabler disabler(this);

  MusicCollection* coll = MusicCollection::Instance();

  int idx = 0; // index counter

  // Iterate through all tracks in vector beat tracking each one
  vector<MusicTrack>::iterator it;
  for (it = tracks->begin(); it != tracks->end(); ++it, ++idx) {

    // Check if user pressed stop during last yield
    if (stopped) { break; }

    DisplayNext();

    MusicTrack& track = *it;
    try {

      auto_ptr<IntShortMap> range = tracker->TrackTempo(track.GetAccess());

      // The reason for setting the length here after having beat tracked the
      // track is that for VBR MP3s, the length is not correct until the entire
      // file has been stepped through. A slight hack but it will have to do
      // for now.
      track.SetLength(tracker->GetInStream().GetLength());

      if (!stopped) {
        track.GetTempoRange().SetMap(*range);
        coll->AddTrack(track);
        DisplayStatus("Complete");
      }

    }
    catch (MusicCollection::DatabaseException& e) {
      wxString msg = "Database error on track '";
      msg << track.GetArtist().c_str();
      msg << " - " << track.GetTitle().c_str() << "':\n";
      msg << e.what() << "\n\n";
```

```
      errorLog << msg;
      DisplayStatus("Failed");
    }
    catch (InStream::InStreamException& e) {
      wxString msg = "Audio reading error on track '";
      msg << track.GetArtist().c_str();
      msg << " - " << track.GetTitle().c_str() << "':\n";
      msg << e.what() << "\n\n";
      errorLog << msg;
      DisplayStatus("Failed");
    }

  }

  CheckErrors();

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************************
  Update
*********************************************************************************/
void ProgressDialog::Update() {

  int percent = static_cast<int>(tracker->GetProgress() * 100);
  gauge->SetValue(percent);

  wxString s;
  s << percent << "%";
  DisplayStatus(s.c_str());

  ::wxYield();

}

/*********************************************************************************
  DisplayNext
*********************************************************************************/
void ProgressDialog::DisplayNext() {

  ++current;

  // Update list box
  DisplayStatus("Tracking");

  // Update current track display
  wxString track = scheduleLC->GetItemText(current);

  // Escape any &s
  track.Replace("&", "&&");
  currentST->SetLabel(track);
  gauge->SetValue(0);

  ::wxYield();

}

/*********************************************************************************
  DisplayStatus
*********************************************************************************/
void ProgressDialog::DisplayStatus(const char* msg) {

  scheduleLC->SetItem(current, 1, msg);

}


/*********************************************************************************
  CheckErrors
*********************************************************************************/
void ProgressDialog::CheckErrors() {

  if (!errorLog.IsEmpty()) {
    LogDialog dlg(this, -1, "Error Log", "There were some "
      "errors. See below for details.", errorLog);
    dlg.ShowModal();
  }

}


/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void ProgressDialog::BuildMe() {

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxVERTICAL );

  wxStaticBox *item9 = new wxStaticBox(this, -1, "Current track");
  wxStaticBoxSizer *item2 = new wxStaticBoxSizer( item9, wxVERTICAL );

  currentST = new wxStaticText(this, ID_CURRENT_ST, "",
    wxDefaultPosition, wxDefaultSize, wxST_NO_AUTORESIZE);
  item2->Add( currentST, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  gauge = new wxGauge( this, ID_GAUGE, 100, wxDefaultPosition,
    wxSize(100,20), 0); // smooth doesn't work on XP
  item2->Add( gauge, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  item1->Add(item2, 0, wxGROW|wxALIGN_CENTER|wxALL, 5);

  wxStaticBox *item5 = new wxStaticBox(this, -1, "Schedule");
  wxStaticBoxSizer *item6 = new wxStaticBoxSizer( item5, wxVERTICAL );

  scheduleLC = new wxListCtrl( this, ID_SCHEDULE_LC, wxDefaultPosition,
    wxSize(460,200), wxLC_REPORT | wxSUNKEN_BORDER | wxLC_HRULES );
  scheduleLC->InsertColumn(0, "Track", wxLIST_FORMAT_LEFT);
  scheduleLC->InsertColumn(1, "Status", wxLIST_FORMAT_LEFT);
  scheduleLC->SetColumnWidth(0, 370);
  scheduleLC->SetColumnWidth(1, 68);

  // Fill schedule with tracks
  int i = 0;
  vector<MusicTrack>::iterator it;
  for (it = tracks->begin(); it != tracks->end(); ++it, ++i) {
    wxString art = it->GetArtist().c_str();
    wxString tit = it->GetTitle().c_str();
    wxString s;
    if (!art.IsEmpty()) {
      s << art << " - ";
    }
    s << tit;
    scheduleLC->InsertItem(i, s, -1);
    scheduleLC->SetItem(i, 1, "Queued");
  }
```

```
  item6->Add(scheduleLC, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  item1->Add(item6, 0, wxGROW|wxALIGN_CENTER|wxALL, 5);

  wxStaticLine *item8 = new wxStaticLine( this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL );
  item1->Add( item8, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  stopBut = new wxButton( this, wxID_CANCEL, "Stop",
    wxDefaultPosition, wxDefaultSize, 0 );
  item1->Add( stopBut, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5 );

  item0->Add( item1, 0, wxALIGN_CENTER|wxALL, 5 );

  wxConfigBase* cfg = wxConfigBase::Get();
  long xpos = cfg->Read("/ProgressDialog/xpos", 100L);
  long ypos = cfg->Read("/ProgressDialog/ypos", 100L);
  Move(xpos, ypos);

  SetAutoLayout( TRUE );
  SetSizer( item0 );
  item0->Fit( this );
  item0->SetSizeHints( this );

}
```

```
/*****************************************************************************

  RangeProcessor.h

  A kind of state machine for applying heuristics to a tempo range after
  it's been beat tracked. Tries to improve the tempo range by removing
  entries that are likely to be wrong.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef RANGEPROCESSOR_H
#define RANGEPROCESSOR_H

#include "Globals.h"

#include <vector>

class RangeProcessor {

  public:

    /*************************************************************************
      Constructor
    *************************************************************************/
    RangeProcessor() { }

    /*************************************************************************
      Destructor
    *************************************************************************/
    ~RangeProcessor() { }

    /*************************************************************************
      Process a range. The range will be modified by the method.
    *************************************************************************/
    void Process(IntShortMap&);


  private:

    /*************************************************************************
      Being in this method means we're in a searching state.
    *************************************************************************/
    void Searching();

    /*************************************************************************
      Being in this method means we're in a state where we've potentially
      found a continuous tempo stretch but it's not yet certain.
    *************************************************************************/
    void PotentialStretch();

    /*************************************************************************
      Being in this method means we're in a continuous tempo stretch.
    *************************************************************************/
    void DefiniteStretch();

    /*************************************************************************
      When we've been in PotentialStretch and it turned out we hadn't found
      a stretch, this method is used to fill the "gap" of the non-stretch
      with the previous tempo.
    *************************************************************************/
    void FillGap();

    /*************************************************************************
      For checking whether two subsequent tempi seem stable.
    *************************************************************************/
    bool Stable(short cur, short next);
```

```
    /*************************************************************************
      Variables
    *************************************************************************/
    static const int stretchLength = 5; // seconds a tempo should stay constant
                                        // for it to be considered a stretch

    static const int searchLimit = 10; // seconds a search can keep going
                                       // before resetting tempo to zero

    static const float stretchDeviation; // max allowed tempo change in percent
                                         // to be considered a stable stretch

    std::vector<ushort> tempi;  // current range
    std::vector<ushort>::iterator pos; // current position in range

    int searchCtr; // counts seconds we've been searching
    int stretchCtr; // counts seconds of stretch
    short prevTempo; // tempo of last found stretch

};

#endif
```

```
/*************************************************************************

  RangeProcessor.cpp

  A kind of state machine for applying heuristics to a tempo range after
  it's been beat tracked. Tries to improve the tempo range by removing
  entries that are likely to be wrong.

  Author: Erik Jälevik

*************************************************************************/

#include "RangeProcessor.h"

#include <cmath> // floor, abs

using namespace std;

const float RangeProcessor::stretchDeviation = 0.033F;

/*************************************************************************
  Process
*************************************************************************/
void RangeProcessor::Process(IntShortMap& range) {

  // If range empty on entry, put a single zero in and return
  if (range.empty()) {
    range[0] = 0;
    return;
  }

  tempi.clear();

  // Get map and copy it across to vector to enable random access
  IntShortMap::const_iterator it = range.begin();
  int startTime = it->first;
  for (; it != range.end(); ++it) {
    tempi.push_back(it->second);
  }

  // Initialise and start recursion
  pos = tempi.begin();
  prevTempo = 0;
  Searching();

  // Remove leading zeros
  pos = tempi.begin();
  while (*pos == 0) {
    range.erase(startTime++);
    ++pos;
  }

  // Copy rest of vector back into map
  for (; pos != tempi.end(); ++pos) {
    range[startTime++] = *pos;
  }

  // Put a single zero in if range empty
  if (range.empty()) {
    range[0] = 0;
  }

}

/*************************************************************************
  Searching
*************************************************************************/
void RangeProcessor::Searching() {
```

```
  if (pos != tempi.end() - 1) { ++pos; } else { *pos = prevTempo; return; }

  ++searchCtr;

  if (Stable(*(pos - 1), *pos)) {
    stretchCtr = 2; // found two consecutive tempi
    PotentialStretch();
  }
  else {

    if (searchCtr >= searchLimit) {

      prevTempo = 0;
      searchCtr = 0;
    }

    *(pos - 1) = prevTempo;
    Searching();
  }

}

/*****************************************************************************
  PotentialStretch
*****************************************************************************/
void RangeProcessor::PotentialStretch() {

  if (pos != tempi.end() - 1) {
    ++pos;
  }
  else {
    ++pos;
    FillGap();
    --pos;
    return;
  }

  ++searchCtr;

  if (Stable(*(pos - 1), *pos)) {
    if (++stretchCtr >= stretchLength) {
      DefiniteStretch();
    }
    else {
      PotentialStretch();
    }
  }

  else {
    // False alarm, not a stretch
    FillGap();

    // Check if new tempo is same as previous stretch
    if (prevTempo != 0 && Stable(prevTempo, *pos)) {

      DefiniteStretch();
    }
    else {
      Searching();
    }

  }

}

/*****************************************************************************
  DefiniteStretch
*****************************************************************************/
void RangeProcessor::DefiniteStretch() {

  if (pos != tempi.end() - 1) { ++pos; } else { return; }

  if (Stable(*(pos - 1), *pos)) {
    DefiniteStretch();
  }
  else {
    prevTempo = *(pos - 1);
    searchCtr = 0;
    Searching();
  }

}

/*****************************************************************************
  FillGap
*****************************************************************************/
void RangeProcessor::FillGap() {

  if (searchCtr >= searchLimit) {

    prevTempo = 0;
    searchCtr = 0;
  }

  // Set tempo to that of previous stretch
  for (; stretchCtr > 0; --stretchCtr) {
    *(pos - stretchCtr) = prevTempo;
  }

}

/*****************************************************************************
  Stable
*****************************************************************************/
bool RangeProcessor::Stable(short cur, short next) {

  short change = static_cast<short>(abs(next - cur));
  short allowance = static_cast<short>(floor(stretchDeviation * cur + 0.5));

  // Considered stable either if tempo is within allowance or tempo has
  // doubled or halved.
  if (change <= allowance) {
    return true;
  }
  else if (next >= (cur * 2) - (allowance * 2) &&
           next <= (cur * 2) + (allowance * 2)) {
    // halve tempo
    *pos = static_cast<short>(floor(*pos / 2.0 + 0.5));
    return true;
  }
  else if (next >= (cur / 2.0) - (allowance / 2.0) &&
           next <= (cur / 2.0) + (allowance / 2.0)) {
    // double tempo
    *pos = *pos * 2;
    return true;
  }
  else {
    return false;
  }

}
```

```
/*****************************************************************************

  Resonator.h

  Comb filter for use as a resonator. It uses a circular buffer to hold the
  current history of output values (y values), the pointer index points to
  the value output by the filter delay time steps ago. This values is used
  in the calculation of the new output.

  Equation for the filter:

  y[t] = alpha * y[t-delay] + (1 - alpha) * x[t]

  For efficiency, the functions have not been made virtual, so subclass
  members can't be accessed through a Resonator pointer.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef RESONATOR_H
#define RESONATOR_H

#include "Globals.h"

#include <cmath>    // pow

class Resonator {

  public:

    /*************************************************************************
      Parameters:
      delay - delay in number of samples
      halfTime - time until half energy is reached when sent impulse response
    *************************************************************************/
    Resonator(uint delay, double halfTime) {
      Init(delay, pow(0.5, delay / halfTime));
    }

    /*************************************************************************
      Default constructor creates a resonator with delay 1 and alpha 0.5.
    *************************************************************************/
    Resonator() { Init(1, 0.5); }

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~Resonator() { delete[] buffer; }

    /*************************************************************************
      Input one value. Output is not returned, it's just stored in internal
      buffer.
    *************************************************************************/
    virtual void Tick(double input) {
      buffer[index] = alpha * buffer[index] + beta * input;
      if (++index == length) { index = 0; }
    }

    /*************************************************************************
      Returns the energy of the resonator.
    *************************************************************************/
    virtual double GetEnergy();

    /*************************************************************************
      Returns the delay of the resonator in samples.
    *************************************************************************/
    virtual uint GetDelay() { return length; }

    /*************************************************************************
      Returns the entire delay line of the resonator.
    *************************************************************************/
    virtual double* GetDelayLine() { return buffer; }

    /*************************************************************************
      Returns the index of the current delay line.
    *************************************************************************/
    virtual uint GetIndex() { return index; }

    /*************************************************************************
      Returns the resonator's last output.
    *************************************************************************/
    virtual double GetLast() {
      if (index >= 1) { return buffer[index - 1]; }
      else            { return buffer[length - 1]; }
    }


  protected:

    /*************************************************************************
      Initialises buffer etc.
    *************************************************************************/
    void Init(uint, double);


    /*************************************************************************
      Variables
    *************************************************************************/
    uint length; // length of buffer, i.e. delay in samples
    double alpha; // alpha value, see Scheirer's paper
    double beta; // 1 - alpha
    double* buffer; // dynamically allocated array representing delay line
    uint index; // pointer to current index in buffer

};

#endif
```

```
/*****************************************************************************

  Resonator.cpp

  Comb filter for use as a resonator. It uses a circular buffer to hold the
  current history of output values (y values), the pointer index points to
  the value output by the filter delay time steps ago. This values is used
  in the calculation of the new output.

  Equation for the filter:

  y[t] = alpha * y[t-delay] + (1 - alpha) * x[t]

  Author: Erik Jälevik

*****************************************************************************/

#include "Resonator.h"

/*****************************************************************************
  Init
*****************************************************************************/
void Resonator::Init(uint delay, double alpha) {

  length = delay;
  this->alpha = alpha;
  beta = sqrt(1.0 - alpha * alpha);

  buffer = new double[length];

  // Initialise buffer to zero
  for(uint i = 0; i < length; ++i) {
    buffer[i] = 0.0;
  }

  index = 0;

}

/*****************************************************************************
  GetEnergy
*****************************************************************************/
double Resonator::GetEnergy() {

  double pwr = 0.0;
  double sum = 0.0;

  // Step through the delay line adding up its power
  for (uint i = 0; i < length ; ++i) {
    pwr += buffer[i] * buffer[i];
  }
  pwr = sqrt(pwr);

  for (uint i = 0; i < length; ++i) {
    sum += pwr - buffer[i];
  }

  return sum / length;

  // TODO: could keep a running total instead

}
```

```
/*****************************************************************************

  ScheirerTracker.h

  A BeatTracker implementing a version of the algorithm described in:

  Scheirer, E. D. "Tempo and Beat Analysis of Acoustic Musical Signals",
  J. Acoust. Soc. Am. 103:1, Jan 1998, pp. 588-601.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef SCHEIRERTRACKER_H
#define SCHEIRERTRACKER_H

#include "BeatTracker.h"
#include "EnvelopeFollower.h"
#include "HanningFollower.h"
#include "Resonator.h"
#include "PhaseResonator.h"
#include "RangeProcessor.h"

#include "Filter.h"
#include "WvOut.h"

#include <string>
#include <vector>

class ScheirerTracker : public BeatTracker {

  public:

    /*************************************************************************
      Parameters:
      stream - the InStream to use for audio input
      accuracy - accuracy of tracking, higher = better
      trackChanges - report tempo changes (true) or overall tempo (false)
      trackPhase - detect phase of beats (i.e. downbeat location) or not
    *************************************************************************/
    ScheirerTracker(std::auto_ptr<InStream> stream,
                    int accuracy = 3,
                    bool trackChanges = true,
                    bool trackPhase = false);
    virtual ~ScheirerTracker();

    /*************************************************************************
      Tracks the tempo of the named file.
    *************************************************************************/
    virtual std::auto_ptr<IntShortMap> TrackTempo(const std::string& file);

    /*************************************************************************
      Tracks the tempo using the passed stream.
    *************************************************************************/
    //virtual std::auto_ptr<IntShortMap> TrackTempo(InStream&);

    /*************************************************************************
      Sets the accuracy to use. The argument should be a value between 1
      and 5. A low accuracy means faster operation but less accurate beat
      tracking.
    *************************************************************************/
    virtual void SetAccuracy(int);

    /*************************************************************************
      Returns a value between 0 and 1 indicating how far the tracking of
      the current file has got.
    *************************************************************************/
    virtual float GetProgress() { return in->GetProgress(); }
```

```
  /*************************************************************************
    Call to stop an ongoing tracking operation.
  *************************************************************************/
  virtual void Stop() { stopped = true; }

  /*************************************************************************
    Call this from scheduler after each Notify to check whether tracker has
    been stopped.
  *************************************************************************/
  virtual bool WasStopped() { return stopped; }


protected:

  /*************************************************************************
    Initialises filter banks and resonators.
  *************************************************************************/
  void Init();

  /*************************************************************************
    Deletes all dynamically allocated memory.
  *************************************************************************/
  void DeleteMemory();

  /*************************************************************************
    Performs the actual beat tracking work.
  *************************************************************************/
  std::auto_ptr<IntShortMap> DoTrackTempo();

  /*************************************************************************
    Resets vectors and other variables used during beat tracking.
  *************************************************************************/
  void Reset();

  /*************************************************************************
    Performs the bandpass filtering of the signal and passes subbands on to
    envelope followers.
  *************************************************************************/
  void BandPassFilter();

  /*************************************************************************
    Extracts envelopes and derivatives and passes them on to resonators.
  *************************************************************************/
  void ExtractEnvelopes();

  /*************************************************************************
    Collects the resonator energy and returns the BPM corresponding to the
    winning resonator.
  *************************************************************************/
  std::pair<int, int> CheckResonators();

  /*************************************************************************
    Normalises array to the range 0-1.
  *************************************************************************/
  void Normalise(double*, int);

  /*************************************************************************
    Applies some heuristics to improve a tempo range.
  *************************************************************************/
  void PostProcess(IntShortMap&);

  /*************************************************************************
    Works out the most likely overall tempo.
  *************************************************************************/
  void OverallTempo(IntShortMap&);
```

```
private:

  /*************************************************************************
    Parameters
  *************************************************************************/
  static const uint lowBpm = 80;   // lowest detectable BPM, DO start at 80
  static const uint highBpm = 200; // highest detectable BPM
  static const uint startup = 5;   // seconds to wait before first BPM check
  static const uint halfTime = 2000; // time (ms) for resonator impulse
                                     // response to reach half energy (ms)

  static double bpB_8000_6_7[6][7];  // coefficients for x values for
  static double bpB_12000_6_7[6][7]; // bandpass filters for different
  static double bpB_16000_6_7[6][7]; // audio rates
  static double bpB_22000_6_7[6][7];

  static double bpA_8000_6_7[6][7];  // coefficients for y values for
  static double bpA_12000_6_7[6][7]; // bandpass filters for different
  static double bpA_16000_6_7[6][7]; // audio rates
  static double bpA_22000_6_7[6][7];

  double(* bpB)[7]; // actual coefficients used
  double(* bpA)[7]; // (depending on accuracy setting)

  uint nSubBands; // number of frequency subbands
  uint bpOrder;   // order of bandpass filters

  uint envRate; // envelope sample rate (Hz), must divide audio frequency
                // evenly, higher rate means more closely spaced
                // resonators

  bool phasing;   // whether we are detecting phase or not
  uint phaseRate; // phase check rate (Hz), i.e. how many times per second
                  // to check the resonators for phase information, must
                  // divide envRate evenly

  uint checkInterval; // how often we check energy of resonators if
                      // not phasing

  uint nResonators; // number of resonators, depends on envRate,
                    // lowBpm and highBpm

  const std::string drumFile; // file name of drum sound to insert at beats

  /*************************************************************************
    Filter banks and member objects
  *************************************************************************/

  // Can't store auto_ptrs in containers and can't store the objects
  // themselves because they contain dynamic memory so have to use
  // regular pointers.
  std::vector<Filter*> bpFilters; // bandpass filters

  std::vector<HanningFollower*> envFollowers; // envelope calculators

  // one resonator bank for each subband
  std::vector<std::vector<PhaseResonator*> > resonators;

  RangeProcessor processor;

  /*************************************************************************
    Working structures
  *************************************************************************/
  ulong sampleCtr; // number of samples of audio read

  // Keeping all of the following makes outputting debug info easier
  std::vector<double> subBands;  // current subband values
  std::vector<double> envelopes; // current envelope values per subband
```

```
    std::vector<double> prevEnvs;  // previous sample's envelope values
    std::vector<double> derivs;    // envelope derivatives per subband

    std::vector<std::vector<double> > energy; // resonator energies per subband
    std::vector<double*> delayLines; // resonator delay lines per subband
    std::vector<double> delaySums; // resonator delay line sums across subbands

    std::vector<double> drum; // the drum samples
    size_t drumIdx; // where in the drum sound we are
    bool boom; // true for as long as a drum sound is being output

    // debugging output switches
    static const bool outputSub = false;
    static const bool outputEnv = false;
    static const bool outputDeriv = false;
    static const bool outputEnergy = false;
    static const int outputReson = -1;
    static const bool outputPhase = false;

    // debugging output streams
    WvOut clickFile;
    WvOut subFiles[6];
    std::ofstream envFiles[6];
    std::ofstream derivFiles[6];
    std::ofstream energyFiles[6];
    std::ofstream energySumFile;
    std::ofstream resonFiles[6];
    std::ofstream resonSumFile;
    std::ofstream phaseFiles[6];
    std::ofstream phaseSumFile;

};

#endif
```

```
/*******************************************************************************

  ScheirerTracker.cpp

  A BeatTracker implementing a version of the algorithm described in:

  Scheirer, E. D. "Tempo and Beat Analysis of Acoustic Musical Signals",
  J. Acoust. Soc. Am. 103:1, Jan 1998, pp. 588-601.

  Author: Erik Jälevik

*******************************************************************************/

#include "ScheirerTracker.h"
#include "FileInStream.h"    // for reading in drum sound
#include "Globals.h"

#include <sstream>
#include <cmath>
#include <limits>            // num_limits

using namespace std;

/*******************************************************************************
  Static parameters
*******************************************************************************/

// B coefficients for 6th order 8000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpB_8000_6_7[6][7] = {
  { 0.01008682522332, -0.05843786306527, 0.14306909652920, -0.18943476125840,
0.14306909652920, -0.05843786306527, 0.01008682522332 },
  { 0.00376928598647, -0.01434858113252, 0.01739822358498, -0.00000000000000,
-0.01739822358498, 0.01434858113252, -0.00376928598647 },
  { 0.00776756168611, -0.02546100621495, 0.02788976327834, -0.00000000000000,
-0.02788976327834, 0.02546100621495, -0.00776756168611 },
  { 0.01817484295119, -0.03154252788024, 0.01600670766884, -0.00000000000000,
-0.01600670766884, 0.03154252788024, -0.01817484295119 },
  { 0.05976405774141, 0.02924547510000, -0.09155574792423, -0.00000000000000,
0.09155574792423, -0.02924547510000, -0.05976405774141 },
  { 0.01520507659478, 0.05332524154303, 0.10178400873026, 0.12225397039275,
0.10178400873026, 0.05332524154303, 0.01520507659478 }
};

// A coefficients for 6th order 8000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpA_8000_6_7[6][7] = {
  { 1.00000000000000, -5.88861081938169, 14.49588310428154, -19.09349058807506,
14.19205037922033, -5.64407613971332, 0.93824647522156 },
  { 1.00000000000000, -5.76896855144678, 14.02680354434945, -18.39400533071941,
13.71964251391261, -5.51914022431232, 0.93578302645826 },
  { 1.00000000000000, -5.22551196045376, 11.95518006796402, -15.25298365181883,
11.43667995399035, -4.78220118769562, 0.87572330374485 },
  { 1.00000000000000, -3.36659946598967, 6.32505067602225, -7.34336279131566,
5.79596577210699, -2.81740112357986, 0.76701335905224 },
  { 1.00000000000000, 1.68946070525993, 2.47056302162255, 2.34782158538899,
2.18652242095955, 1.18025717823480, 0.58547667319587 },
  { 1.00000000000000, 5.00683944081512, 11.11583966375599, 13.87960458972819,
10.25392999414134, 4.24944505642616, 0.77514191585016 }
};

// B coefficients for 6th order 12000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpB_12000_6_7[6][7] = {
  { 0.00996988494274, -0.05889189422908, 0.14585831934990, -0.19387249982323,
0.14585831934990, -0.05889189422908, 0.00996988494274 },
  { 0.00251032290148, -0.00982280967180, 0.01211584281646, -0.00000000000000,
-0.01211584281646, 0.00982280967180, -0.00251032290148 },
```

```
  { 0.00505409098459, -0.01851069178075, 0.02189608396762, 0.00000000000000,
-0.02189608396762, 0.01851069178075, -0.00505409098459 },
  { 0.01077854434217, -0.03017773778017, 0.02911070262561, -0.00000000000000,
-0.02911070262561, 0.03017773778017, -0.01077854434217 },
  { 0.02817581221592, -0.02121531001364, -0.01463240096341, 0.00000000000000,
0.01463240096341, 0.02121531001364, -0.02817581221592 },
  { 0.05290385932159, -0.07046928980191, 0.15845203583412, -0.14001657884509,
0.15845203583412, -0.07046928980191, 0.05290385932159 }
};

// A coefficients for 6th order 12000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpA_12000_6_7[6][7] = {
  { 1.00000000000000, -5.93627594122473, 14.70438241639539, -19.45374029649818,
14.49783465135632, -5.77059454156658, 0.95839392547171 },
  { 1.00000000000000, -5.88208950814071, 14.48880765746763, -19.12894798883716,
14.27658673195741, -5.71106102209029, 0.95671439866001 },
  { 1.00000000000000, -5.62099251710983, 13.44099237907855, -17.48579885439504,
13.04981497834374, -5.29869617860944, 0.91531365573818 },
  { 1.00000000000000, -4.70766357828481, 10.16650924835991, -12.66096828149142,
9.58304733600474, -4.18222808911066, 0.83786585835244 },
  { 1.00000000000000, -1.74207223701924, 3.20872384332820, -2.99982779709008,
2.88574663348879, -1.37268481387304, 0.70200316853124 },
  { 1.00000000000000, 1.78003464348789, 3.33154135927194, 3.27688979905778,
2.82425691797163, 1.40410635596955, 0.55654896777703 }
};

// B coefficients for 6th order 16000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpB_16000_6_7[6][7] = {
  { 0.00994432425081, -0.05914313256535, 0.14707987262513, -0.19576210709709,
0.14707987262513, -0.05914313256535, 0.00994432425081 },
  { 0.00188514714007, -0.00744787396541, 0.00924059137043, 0.00000000000000,
-0.00924059137043, 0.00744787396541, -0.00188514714007 },
  { 0.00376928598647, -0.01434858113252, 0.01739822358498, -0.00000000000000,
-0.01739822358498, 0.01434858113252, -0.00376928598647 },
  { 0.00776756168611, -0.02546100621495, 0.02788976327834, -0.00000000000000,
-0.02788976327834, 0.02546100621495, -0.00776756168611 },
  { 0.01817484295119, -0.03154252788024, 0.01600670766884, -0.00000000000000,
-0.01600670766884, 0.03154252788024, -0.01817484295119 },
  { 0.10178242254484, -0.33424422276021, 0.64406215470136, -0.77682701113027,
0.64406215470136, -0.33424422276020, 0.10178242254484 }
};

// A coefficients for 6th order 16000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpA_16000_6_7[6][7] = {
  { 1.00000000000000, -5.95617330871181, 14.79368331490100, -19.61259423719054,
14.63746254415515, -5.83100821683404, 0.96862994195611 },
  { 1.00000000000000, -5.92525925920999, 14.66980657584361, -19.42469776412128,
14.50837770317267, -5.79558163629372, 0.96735622234710 },
  { 1.00000000000000, -5.76896855144678, 14.02680354434945, -18.39400533071941,
13.71964251391261, -5.51914022431232, 0.93578302645826 },
  { 1.00000000000000, -5.22551196045376, 11.95518006796402, -15.25298365181883,
11.43667995399035, -4.78220118769562, 0.87572330374485 },
  { 1.00000000000000, -3.36659946598967, 6.32505067602225, -7.34336279131566,
5.79596577210699, -2.81740112357986, 0.76701335905224 },
  { 1.00000000000000, -0.32722973404339, 2.01377074337875, -0.05607604970283,
1.43193829121158, 0.09020948601332, 0.48400949486191 }
};

// B coefficients for 6th order 22000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpB_22000_6_7[6][7] = {
  { 0.00993988756862, -0.05936211785582, 0.14799124106247, -0.19713801835180,
0.14799124106247, -0.05936211785582, 0.00993988756862 },
  { 0.00137368003770, -0.00545888622938, 0.00679679061437, -0.00000000000000,
-0.00679679061437, 0.00545888622938, -0.00137368003770 },
  { 0.00273807509763, -0.01066926283980, 0.01312613759040, 0.00000000000000,
-0.01312613759040, 0.01066926283980, -0.00273807509763 },
  { 0.00553098218293, -0.01991967606159, 0.02330316937936, 0.00000000000000,
-0.02330316937936, 0.01991967606159, -0.00553098218293 },
  { 0.01197109968873, -0.03126190491753, 0.02826676618600, 0.00000000000000,
-0.02826676618600, 0.03126190491753, -0.01197109968873 },
  { 0.17251121640125, -0.78230118606458, 1.68480629041566, -2.13878362041441,
1.68480629041566, -0.78230118606458, 0.17251121640125 }
};

// A coefficients for 6th order 22000 Hz filter with passband ripple 5 dB and
// stopband attenuation 40 dB.
double ScheirerTracker::bpA_22000_6_7[6][7] = {
  { 1.00000000000000, -5.97049136039656, 14.85918617337241, -19.73168392774267,
14.74486129129179, -5.87895851823126, 0.97708634739454 },
  { 1.00000000000000, -5.95383754788791, 14.79199367743398, -19.62896113916824,
14.67344459636956, -5.85879090863442, 0.97615159600655 },
  { 1.00000000000000, -5.86416242362284, 14.41455295785746, -19.00921760743110,
14.18436645305754, -5.67839603582415, 0.95287391366199 },
  { 1.00000000000000, -5.55877106225240, 13.19988120438684, -17.11737786739901,
12.78128660912621, -5.21194238967738, 0.90798297166864 },
  { 1.00000000000000, -4.49183603000218, 9.47157982693885, -11.67709161628955,
8.88057776239659, -3.94752371532460, 0.82451325802423 },
  { 1.00000000000000, -2.12883719866612, 3.42064924422264, -2.92638969068209,
2.27858049700734, -0.97806973353410, 0.45878765440920 }
};

/*****************************************************************************
  Constructor
*****************************************************************************/
ScheirerTracker::ScheirerTracker(auto_ptr<InStream> stream,
                                 int accuracy,
                                 bool trackChanges,
                                 bool trackPhase) :
  BeatTracker(stream),
  phasing(trackPhase),
  checkInterval(1),
  nSubBands(6),
  bpOrder(6),
  drumFile("rimshot.wav") {

  // Can't call in->SetRate here because it changes in SetAccuracy

  SetTrackChanges(trackChanges);
  SetAccuracy(accuracy); // will call Init

}

/*****************************************************************************
  Destructor
*****************************************************************************/
ScheirerTracker::~ScheirerTracker() {

  DeleteMemory();
  in->Close();

  // Close debug files
  for (uint i = 0; i < nSubBands; ++i) {
    subFiles[i].closeFile();
    envFiles[i].close();
    derivFiles[i].close();
    energyFiles[i].close();
    resonFiles[i].close();
    phaseFiles[i].close();
  }
  energySumFile.close();
  resonSumFile.close();
  phaseSumFile.close();
```

```
  clickFile.closeFile();

}

/*****************************************************************************
  SetAccuracy
*****************************************************************************/
void ScheirerTracker::SetAccuracy(int acc) {

  // Could also incorporate envRate in a more direct way, i.e. of accuracy
  // is high, fractional bpms should be reported.

  switch (acc) {
    case 1: // lowest accuracy
      audioRate = 8000;
      envRate = 100;
      phaseRate = 5;
      bpA = bpA_8000_6_7;
      bpB = bpB_8000_6_7;
      break;
    case 2:
      audioRate = 8000;
      envRate = 200;
      phaseRate = 5;
      bpA = bpA_8000_6_7;
      bpB = bpB_8000_6_7;
      break;
    case 3:
      audioRate = 12000;
      envRate = 200;
      phaseRate = 5;
      bpA = bpA_12000_6_7;
      bpB = bpB_12000_6_7;
      break;
    case 4:
      audioRate = 16000;
      envRate = 400;
      phaseRate = 5;
      bpA = bpA_16000_6_7;
      bpB = bpB_16000_6_7;
      break;
    case 5: // highest accuracy
      audioRate = 22000;
      envRate = 550;
      phaseRate = 5;
      bpA = bpA_22000_6_7;
      bpB = bpB_22000_6_7;
      break;
    default: // use 3
      audioRate = 12000;
      envRate = 200;
      phaseRate = 5;
      bpA = bpA_12000_6_7;
      bpB = bpB_12000_6_7;
      break;
  }

  // Check that envelope rate divides audio rate.
  if (audioRate % envRate != 0) {
    throw logic_error("Envelope rate does not divide audio rate evenly.");
  }

  // Check that phase rate divides envelope rate.
  if (envRate % phaseRate != 0) {
    throw logic_error("Phase rate does not divide envelope rate evenly.");
  }

  Init();
}

/*****************************************************************************
  TrackTempo
*****************************************************************************/
auto_ptr<IntShortMap> ScheirerTracker::TrackTempo(const string& fname) {

  in->Open(fname);
  auto_ptr<IntShortMap> r = DoTrackTempo();
  return r;

}

/*****************************************************************************
  Init
*****************************************************************************/
void ScheirerTracker::Init() {

  DeleteMemory();

  in->SetRate(audioRate);

  // Set up debugging output
  ostringstream fname;
  for (uint i = 0; i < nSubBands; ++i) {

    if (outputSub) {
      fname.str("");
      fname << "output/sub" << i << ".wav";
      subFiles[i].openFile(fname.str().c_str(), 1,
        WvOut::WVOUT_WAV, Stk::STK_SINT16);
    }

    if (outputEnv) {
      fname.str("");
      fname << "output/env" << i << ".dat";
      envFiles[i].open(fname.str().c_str(), ios::trunc);
      if (!envFiles[i]) {
        LOG("envFiles[" << i << "] didn't open!\n");
      }
    }

    if (outputDeriv) {
      fname.str("");
      fname << "output/deriv" << i << ".dat";
      derivFiles[i].open(fname.str().c_str(), ios::trunc);
      if (!derivFiles[i]) {
        LOG("derivFiles[" << i << "] didn't open!\n");
      }
    }

    // Output subsequent energies as a 2d array
    if (outputEnergy) {
      fname.str("");
      fname << "output/energy" << i << ".dat";
      energyFiles[i].open(fname.str().c_str(), ios::trunc);
      if (!energyFiles[i]) {
        LOG("energyFiles[" << i << "] didn't open!\n");
      }
    }

    if (outputReson != -1) {
      fname.str("");
      fname << "output/reson" << i << ".dat";
      resonFiles[i].open(fname.str().c_str(), ios::trunc);
      if (!resonFiles[i]) {
        LOG("resonFiles[" << i << "] didn't open!\n");
```

```
    }
  }

  // Output subsequent phase consts as a 2d array
  if (outputPhase) {
    fname.str("");
    fname << "output/phase" << i << ".dat";
    phaseFiles[i].open(fname.str().c_str(), ios::trunc);
    if (!phaseFiles[i]) {
      LOG("phaseFiles[" << i << "] didn't open!\n");
    }
  }

}

if (outputEnergy) {
  energySumFile.open("output/energysum.dat", ios::trunc);
  if (!energySumFile) {
    LOG("energySumFile didn't open!\n");
  }
}
if (outputReson != -1) {
  fname.str("");
  fname << "output/resonsum.dat";
  resonSumFile.open(fname.str().c_str(), ios::trunc);
  if (!resonSumFile) {
    LOG("resonSumFile didn't open!\n");
  }
}
if (outputPhase) {
  phaseSumFile.open("output/phasesum.dat", ios::trunc);
  if (!phaseSumFile) {
    LOG("phaseSumFile didn't open!\n");
  }
}
if (phasing) {
  clickFile.openFile("output/click.wav", 1, WvOut::WVOUT_WAV, Stk::STK_SINT16);
}


// Set up bandpass filters and envelope followers
for(uint i = 0; i < nSubBands; ++i) {

  bpFilters.push_back(new Filter(bpOrder + 1, bpB[i],
                                 bpOrder + 1, bpA[i]));

  envFollowers.push_back(new HanningFollower(0.2 / (i + 1), audioRate));

}

// Set up resonators
int envSamplesPerMin = envRate * 60;
int lowDelay = static_cast<int>(envSamplesPerMin / highBpm + 0.5);
int highDelay = static_cast<int>(envSamplesPerMin / lowBpm + 0.5);

// Use maximum possible resonator resolution, i.e. one resonator per tick
nResonators = highDelay - lowDelay;

// Resize enough to hold the longest possible delay line. This vector is
// used in CheckResonators.
delaySums.resize(highDelay);

LOG("Number of resonators: " << nResonators << endl);

double halfTimeInEnvSamples = (halfTime / 1000.0) * envRate;

resonators.resize(nSubBands);

for (uint i = 0; i < nSubBands; ++i) {

  for (uint j = 0; j < nResonators; ++j) {

    int delay = lowDelay + j;

    // Make sure each resonator has the same half energy time
    resonators[i].push_back(new PhaseResonator(delay, halfTimeInEnvSamples));

    #ifdef _DEBUG
      if (i == 0)
        LOG("resonators[" << i << "][" << j << "]: " <<
          "delay: " << delay << " ticks, " <<
          "tempo: " << (static_cast<float>(envRate) / delay) * 60 << "bpm, " <<
          endl);
    #endif

  }

}

if (phasing) {
  // Load drum sound into memory
  auto_ptr<FileInStream> ds(new FileInStream()); // too large for stack
  ds->SetRate(audioRate);
  ds->Open(drumFile);
  drum.clear();
  drum.reserve(1000); // arbitrary number enough for a very short sound
  while (ds->HasMore()) {
    drum.push_back(ds->Read());
  }
  ds->Close();
}

// Output tempo range at top of energy and phase files
for (uint j = 0; j < nSubBands; ++j) {
  if (outputEnergy) { energyFiles[j] << 0 << " "; }
  if (outputPhase) { phaseFiles[j] << 0 << " "; }
}
if (outputEnergy) { energySumFile << 0 << " "; }
if (outputPhase) { phaseSumFile << 0 << " "; }
for (uint i = 0; i < nResonators; ++i) {
  float bpm = (static_cast<float>(envRate) / resonators[0][i]->GetDelay()) * 60;
  for (uint j = 0; j < nSubBands; ++j) {
    if (outputEnergy) { energyFiles[j] << bpm << " "; }
    if (outputPhase) { phaseFiles[j] << bpm << " "; }
  }
  if (outputEnergy) { energySumFile << bpm << " "; }
  if (outputPhase) { phaseSumFile << bpm << " "; }
}
for (uint j = 0; j < nSubBands; ++j) {
  if (outputEnergy) { energyFiles[j] << endl; }
  if (outputPhase) { phaseFiles[j] << endl; }
}
if (outputEnergy) { energySumFile << endl; }
if (outputPhase) { phaseSumFile << endl; }


}

/*****************************************************************************
  DeleteMemory
*****************************************************************************/
void ScheirerTracker::DeleteMemory() {

  for (size_t i = 0; i < bpFilters.size(); ++i) {
    delete bpFilters[i];
    delete envFollowers[i];
```

```
    for (uint j = 0; j < nResonators; ++j) {
      delete resonators[i][j];
    }
    resonators[i].clear();
  }
  bpFilters.clear();
  envFollowers.clear();

}

/**************************************************************************
  DoTrackTempo
**************************************************************************/
auto_ptr<IntShortMap> ScheirerTracker::DoTrackTempo() {

  LOG("Starting to beat track...\n");

  Reset();

  auto_ptr<IntShortMap> range(new IntShortMap());

  // Number of audio samples between each sample in downsampled envelope
  int envInterval = audioRate / envRate; // will be 210 for 44100 audio
  int phaseCtr = 0; // counts down to next  beat

  // Number of audio samples between each resonator check, 1s if not phasing
  int chkInterval = phasing ? (audioRate / phaseRate) : audioRate;

  while (in->HasMore()) {

    BandPassFilter();

    // Downsample envelope by taking only every envInterval sample
    if ((++sampleCtr % envInterval) == 0) {

      // Notify observers once every 1/5th second of audio.
      // Need to make sure here that the divisor used divides evenly for
      // all possible audio rates. 5 works for 11025, 16000, 22050, 44100.
      if ((sampleCtr % (audioRate / 5)) == 0) {
        Notify();
        if (stopped) { return range; }
      }

      // Check if we've reached a downbeat
      if (phasing && --phaseCtr == 0) {
        boom = true;
      }

      ExtractEnvelopes();

      // Check resonators every chkInterval samples
      if ((sampleCtr % chkInterval) == 0) {

        pair<int, int> bpmPhase = CheckResonators();
        int bpm = bpmPhase.first;
        phaseCtr = bpmPhase.second;

        // Store current bpm in tempo range every second
        if ((sampleCtr % audioRate) == 0) {

          uint time = static_cast<float>(sampleCtr) / audioRate;

          // Don't report tempo before startup seconds have passed
          if (time >= startup) {

            LOG(time << ": " << bpm << " BPM, phase: " <<
              static_cast<float>(phaseCtr) / envRate << endl);

            range->insert(make_pair(time, bpm));

          }

        } // end if audioRate

      } // end if chkInterval

    } // end if envInterval

  } // while in has more

  if (trackChanges) {
    PostProcess(*range);
    return range;
  }
  else {
    OverallTempo(*range);
    return range;
  }

}

/**************************************************************************
  Reset
**************************************************************************/
void ScheirerTracker::Reset() {

  // Need to size all the vectors to the correct lengths so that we can
  // use array-style indexing (v[i]).

  subBands.resize(nSubBands);
  envelopes.resize(nSubBands);
  prevEnvs.assign(nSubBands, 0.0);
  derivs.resize(nSubBands);

  energy.resize(nSubBands);
  vector<vector<double> >::iterator it;
  for (it = energy.begin(); it != energy.end(); ++it) {
    it->resize(nResonators);
  }

  delayLines.resize(nSubBands);

  stopped = false;

  boom = false;
  drumIdx = 0;

  sampleCtr = 0;

}

/**************************************************************************
  BandpassFilter
**************************************************************************/
void ScheirerTracker::BandPassFilter() {

  double input = in->Read();

  if (phasing) {
    if (boom) {
      // Mix signal with drum
      clickFile.tick((input + drum[drumIdx++]) / 2);
      if (drumIdx >= drum.size()) {
        drumIdx = 0;
        boom = false;
      }
```

```
    }
    else {
      clickFile.tick(input);
    }
  }

  for (uint i = 0; i < nSubBands; ++i) {

    // Send signal to bandpass filters
    subBands[i] = bpFilters[i]->tick(input);

    // and then to envelope followers
    envFollowers[i]->Tick(subBands[i]);

    if (outputSub) {
      subFiles[i].tick(subBands[i]);
    }

  }

}

/*****************************************************************************
  ExtractEnvelopes
*****************************************************************************/
void ScheirerTracker::ExtractEnvelopes() {

  for (uint i = 0; i < nSubBands; ++i) {

    envelopes[i] = envFollowers[i]->GetOutput();

    // Take half-wave rectified derivative
    derivs[i] = envelopes[i] - prevEnvs[i];
    if (derivs[i] < 0) { derivs[i] = 0; }
    prevEnvs[i] = envelopes[i];

    // Send derivs into resonators
    for (uint j = 0; j < nResonators; ++j) {
      resonators[i][j]->Tick(derivs[i]);
    }

    if (outputEnv) {
      envFiles[i] << envelopes[i] << endl;
    }
    if (outputDeriv) {
      derivFiles[i] << derivs[i] << endl;
    }
    if (outputReson != -1) {
      resonFiles[i] << resonators[i][outputReson]->GetLast() << endl;
    }

  }

  if (outputReson != -1) {
    double sum = 0.0;
    for (uint i = 0; i < nSubBands; ++i) {
      sum += resonators[i][outputReson]->GetLast();
    }
    resonSumFile << sum << endl;
  }

}

/*****************************************************************************
  CheckResonators
*****************************************************************************/
pair<int, int> ScheirerTracker::CheckResonators() {

  // Collect energy of all resonators
  vector<double> phaseSums;
  if (outputPhase) { phaseSums.assign(nResonators, 0.0); }
  for (uint i = 0; i < nSubBands; ++i) {

    if (outputEnergy) {
      energyFiles[i] << (static_cast<float>(sampleCtr) / audioRate) << " ";
    }
    if (outputPhase) {
      phaseFiles[i] << (static_cast<float>(sampleCtr) / audioRate) << " ";
    }

    for (uint j = 0; j < nResonators; ++j) {
      energy[i][j] = resonators[i][j]->GetEnergy();

      if (outputEnergy) {
        energyFiles[i] << energy[i][j] << " ";
      }
      if (outputPhase) {
        float c =
          resonators[i][j]->GetPhaseConstancy();
        phaseFiles[i] << c << " ";
        phaseSums[j] += c;
      }

    }

    if (outputEnergy) { energyFiles[i] << endl; }
    if (outputPhase) { phaseFiles[i] << endl; }

  }

  if (outputPhase) {
    phaseSumFile << (static_cast<float>(sampleCtr) / audioRate) << " ";
    for (uint j = 0; j < nResonators; ++j) {
      phaseSumFile << phaseSums[j] << " ";
    }
    phaseSumFile << endl;
  }


  // Sum across subbands
  double maxSum = 0.0;
  int winner = -1;

  if (outputEnergy) {
    energySumFile << (static_cast<float>(sampleCtr) / audioRate) << " ";
    energySumFile << 0 << " ";
  }

  for (uint i = 1; i < nResonators - 1; ++i) {

    double sum = 0.0;
    for (uint j = 0; j < nSubBands; ++j) {
      // Taking neighbours into account should help in the cases where the
      // correct tempo is not exactly matched by any of the resonators
      sum += (0.8 * energy[j][i-1]) + energy[j][i] + (0.8 * energy[j][i+1]);
    }

    // Pick resonator with highest energy as winner
    if (sum > maxSum) {
      maxSum = sum;
      winner = i;
    }

    if (outputEnergy) { energySumFile << sum << " "; }

  }
```

```
  if (outputEnergy) {
    energySumFile << 0 << " ";
    energySumFile << endl;
  }

  // Return 0 if no winner
  if (winner == -1) {
    // This happens if all resonators have equal energy
    return make_pair(0, 0);
  }

  // Otherwise work out BPM of winning resonator
  uint delay = resonators[0][winner]->GetDelay();
  ushort bpm = (static_cast<float>(envRate) / delay) * 60 + 0.5;

  if (!phasing) {
    return make_pair(bpm, 0);
  }

  // Work out current phase
  int pos = resonators[0][winner]->GetIndex();

  // Get pointers to delay lines for phase calculation
  for (uint i = 0; i < nSubBands; ++i) {
    delayLines[i] = resonators[i][winner]->GetDelayLine();
  }

  // Sum delay lines across subbands
  maxSum = 0.0;
  winner = -1;
  for (uint i = 0; i < delay; ++i) {

    delaySums[i] = 0.0;
    for (uint j = 0; j < nSubBands; ++j) {
      delaySums[i] += delayLines[j][i];
    }

    // Pick delay with highest total response as winner
    if (delaySums[i] > maxSum) {
      maxSum = delaySums[i];
      winner = i;
    }
  }

  // Wrap around if winner is before current pos
  int timeToBeat = winner - pos;
  if (timeToBeat < 0) {
    timeToBeat = (delay - pos) + winner;
  }

  // Return bpm and time to next beat in envelope samples
  return make_pair(bpm, timeToBeat);

}

/***************************************************************************
  Normalise
***************************************************************************/
void ScheirerTracker::Normalise(double* array, int length) {

  double highest = numeric_limits<double>::min();
  double lowest = numeric_limits<double>::max();

  for (int i = 0; i < length; ++i) {
    if (array[i] > highest) {
      highest = array[i];
    }
    if (array[i] < lowest) {
      lowest = array[i];
    }
  }

  highest -= lowest;

  if (highest != 0.0) {
    for (int i = 0; i < length; ++i) {
      array[i] -= lowest;
      array[i] /= highest;
    }
  }

}

/***************************************************************************
  PostProcess
***************************************************************************/
void ScheirerTracker::PostProcess(IntShortMap& range) {

  // Send range to finite state machine
  processor.Process(range);

  // Delete all tempi that stay the same.
  // Have to do deletion this way to not saw off branch, see Josuttis p. 205.
  short last = -1;
  for (IntShortMap::iterator pos = range.begin(); pos != range.end(); ) {
    if (pos->second == last) {
      range.erase(pos++);
    }
    else {
      last = pos->second;
      ++pos;
    }

  }


}

/***************************************************************************
  OverallTempo
***************************************************************************/
void ScheirerTracker::OverallTempo(IntShortMap& range) {

  // Accumulator with one bin for each tempo, the highest wins.
  // OK to use a vector with its indices representing tempi as the tempi
  // in the TempoRange are all rounded shorts at this stage.
  vector<int> acc;
  acc.assign(highBpm, 0);

  IntShortMap::const_iterator it;
  for (it = range.begin(); it != range.end(); ++it) {
    ++acc.at(it->second);
  }

  float maxtot = 0.0;
  int winner = -1;
  for (size_t i = lowBpm + 1; i < highBpm - 1; ++i) {

    // Look at three neighbouring tempi and work out weighted total
    float tot = 0.75f * acc[i-1] + acc[i] + 0.75f * acc[i+1];

    LOG(i << " BPM, total count: " << tot << endl);

    if (tot > maxtot) {
      maxtot = tot;
```

```
      winner = i;
    }

  }

  range.clear();
  range.insert(make_pair(0, winner));

}
```

```
/*****************************************************************************

  TapperDialog.h

  GUI dialog window and functionality for tapper feature that lets you tap
  out a tempo manually.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef TAPPERDIALOG_H
#define TAPPERDIALOG_H

class TapperDialog : public wxDialog {

  public:

    /*************************************************************************
      Constructor takes parent window and ID.
    *************************************************************************/
    TapperDialog(wxWindow*, wxWindowID);

    /*************************************************************************
      Destructor
    *************************************************************************/
    virtual ~TapperDialog() { }

    /*************************************************************************
      Handler for Tap button.
    *************************************************************************/
    void OnTap(wxCommandEvent&);

    /*************************************************************************
      Handler for Reset button.
    *************************************************************************/
    void OnReset(wxCommandEvent&);

    /*************************************************************************
      Handler for OK button.
    *************************************************************************/
    void OnOK(wxCommandEvent&);

    /*************************************************************************
      Handler for Cancel button.
    *************************************************************************/
    void OnCancel(wxCommandEvent&);

    /*************************************************************************
      Retrieve tapped tempo.
    *************************************************************************/
    int GetTempo() { return tempo; }


  private:

    /*************************************************************************
      Autogenerated method for building dialog
    *************************************************************************/
    void BuildMe();


    /*************************************************************************
      Variables
    *************************************************************************/

    // Controls
    wxButton* tapBut;
```

```
    wxButton* resetBut;
    wxTextCtrl* tempoTC;

    wxButton* okBut;
    wxButton* cancelBut;

    // Others
    bool running;       // tapping is underway
    int tempo;          // current tempo
    wxLongLong lastTap; // time of last tap
    int lastBpm;        // tempo after last tap
    int total;          // sum of BPMs so far
    int nTaps;          // loop counter for number of taps

    enum {
      ID_TAP_BUT,
      ID_RESET_BUT,
      ID_TEMPO_TC
    };

    DECLARE_EVENT_TABLE()

};

#endif
```

```
/*********************************************************************************

  TapperDialog.cpp

  GUI dialog window and functionlity for tapper feature that lets you tap
  out a tempo manually.

  Author: Erik Jälevik

*********************************************************************************/

#include "wx/wxprec.h"

#ifdef __BORLANDC__
  #pragma hdrstop
#endif

#ifndef WX_PRECOMP
  #include "wx/wx.h"
#endif

#include "TapperDialog.h"

#include <wx/statline.h>

#include <cmath>

/*********************************************************************************
  Constructor
*********************************************************************************/
TapperDialog::TapperDialog(wxWindow* parent, wxWindowID id) :
  wxDialog(parent, id, "Tapper", wxDefaultPosition, wxDefaultSize,
    wxCAPTION | wxSYSTEM_MENU),
  tempo(0),
  nTaps(0),
  total(0),
  running(false) {

  BuildMe();

}

/*********************************************************************************
  Event table
*********************************************************************************/
BEGIN_EVENT_TABLE(TapperDialog, wxDialog)
  EVT_BUTTON(ID_TAP_BUT, TapperDialog::OnTap)
  EVT_BUTTON(ID_RESET_BUT, TapperDialog::OnReset)
  EVT_BUTTON(wxID_OK, TapperDialog::OnOK)
  EVT_BUTTON(wxID_CANCEL, TapperDialog::OnCancel)
END_EVENT_TABLE()

/*********************************************************************************
  OnTap
*********************************************************************************/
void TapperDialog::OnTap(wxCommandEvent& event) {

  if (!running) {
    running = true;
    lastTap = ::wxGetLocalTimeMillis();
    lastBpm = 0;
    total = 0;
    nTaps = 0;
  }
  else {
    ++nTaps;
    wxLongLong time = ::wxGetLocalTimeMillis();
    wxLongLong lli = time - lastTap;
```

```
    long interval = lli.ToLong();

    // Tempo based on this interval only
    int newBpm = (int)(1 / (interval / 1000.0F) * 60);
    total = total + newBpm;

    // Running average tempo
    tempo = floor((float)total / nTaps + 0.5);

    wxString t;
    t << tempo;
    tempoTC->SetValue(t);

    lastTap = time;
    lastBpm = tempo;
  }

}

/*********************************************************************************
  OnReset
*********************************************************************************/
void TapperDialog::OnReset(wxCommandEvent& event) {

  running = false;
  tempoTC->SetValue("000");
  tempo = 0;

}

/*********************************************************************************
  OnOK
*********************************************************************************/
void TapperDialog::OnOK(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_OK);
  }
  else {
    SetReturnCode(wxID_OK);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************************
  OnCancel
*********************************************************************************/
void TapperDialog::OnCancel(wxCommandEvent& event) {

  if (IsModal()) {
    EndModal(wxID_CANCEL);
  }
  else {
    SetReturnCode(wxID_CANCEL);
    Show(FALSE);
    Destroy();
  }

}

/*********************************************************************************
  BuildMe
  Mostly autogenerated by wxDesigner, hence the unintuitive variable names.
*********************************************************************************/
void TapperDialog::BuildMe() {

  wxBoxSizer *item0 = new wxBoxSizer( wxVERTICAL );

  wxBoxSizer *item1 = new wxBoxSizer( wxHORIZONTAL );

  wxBoxSizer *item2 = new wxBoxSizer( wxVERTICAL );

  tapBut = new wxButton(this, ID_TAP_BUT, "Tap",
    wxDefaultPosition, wxDefaultSize, 0);
  tapBut->SetDefault();
  item2->Add(tapBut, 0, wxALIGN_CENTER|wxALL, 5);

  resetBut = new wxButton(this, ID_RESET_BUT, "Reset",
    wxDefaultPosition, wxDefaultSize, 0);
  item2->Add(resetBut, 0, wxALIGN_CENTER|wxALL, 5);

  item1->Add( item2, 0, wxALIGN_CENTER|wxALL, 5 );

  tempoTC = new wxTextCtrl(this, ID_TEMPO_TC, "000",
    wxDefaultPosition, wxSize(120,61), wxTE_READONLY | wxTE_CENTRE);
  tempoTC->SetFont(wxFont(36, wxSWISS, wxNORMAL, wxBOLD));
  tempoTC->SetBackgroundColour(wxColour(0, 0, 0));
  tempoTC->SetForegroundColour(wxColour(210, 240, 0));
  item1->Add(tempoTC, 0, wxALIGN_CENTER|wxALL, 5);

  item0->Add( item1, 0, wxALIGN_CENTER|wxALL, 5 );

  wxStaticLine *item6 = new wxStaticLine(this, -1, wxDefaultPosition,
    wxSize(20,-1), wxLI_HORIZONTAL);
  item0->Add(item6, 0, wxGROW|wxALIGN_CENTER_VERTICAL|wxLEFT|wxRIGHT, 10);

  wxBoxSizer *item7 = new wxBoxSizer( wxHORIZONTAL );

  okBut = new wxButton(this, wxID_OK, "OK", wxDefaultPosition,
    wxDefaultSize, 0);
  item7->Add(okBut, 0, wxALIGN_CENTER|wxALL, 5);

  cancelBut = new wxButton(this, wxID_CANCEL, "Cancel",
    wxDefaultPosition, wxDefaultSize, 0);
  item7->Add(cancelBut, 0, wxALIGN_CENTER|wxALL, 5);

  item0->Add(item7, 0, wxALIGN_RIGHT|wxALIGN_CENTER_VERTICAL|wxALL, 5);

  SetAutoLayout(TRUE);
  SetSizer(item0);
  item0->Fit(this);
  item0->SetSizeHints(this);

}
```

```
/*****************************************************************************

  TempoRange.h

  Class for holding tempo ranges, i.e. pairs of time and tempo.

  Author: Erik Jälevik

*****************************************************************************/

#ifndef TEMPORANGE_H
#define TEMPORANGE_H

#include "Globals.h"

#include <map>

class TempoRange {

  public:

    /*************************************************************************
      Constructors
    *************************************************************************/
    TempoRange() { }
    TempoRange(const TempoRange& that) : range(that.range) { }
    ~TempoRange() { }

    /*************************************************************************
      Assignment just copies map
    *************************************************************************/
    TempoRange& operator=(const TempoRange& that) {
      if(&that != this) { range = that.range; }
      return *this;
    }

    /*************************************************************************
      Adds a time (in seconds) and a tempo to the range.
    *************************************************************************/
    void Add(uint time, ushort tempo) { range[time] = tempo; }

    /*************************************************************************
      Returns a string consisting of the starting tempo followed by a hyphen
      followed by the end tempo.
    *************************************************************************/
    std::string GetRangeString();

    /*************************************************************************
      Returns the first tempo in range.
    *************************************************************************/
    ushort GetFirst() { return range.begin()->second; }

    /*************************************************************************
      Returns the last tempo in range.
    *************************************************************************/
    ushort GetLast() { return range.rbegin()->second; }

    /*************************************************************************
      Returns true if no tempi are in range.
    *************************************************************************/
    bool IsEmpty() { return range.empty(); }

    /*************************************************************************
      Returns a reference to the range map.
    *************************************************************************/
    IntShortMap& GetMap() { return range; }

    /*************************************************************************
      Sets the tempo range map.
    *************************************************************************/
    void SetMap(const IntShortMap& r) { range = r; }

    /*************************************************************************
      Clears tempo range.
    *************************************************************************/
    void Clear() { range.clear(); }


  private:

    /*************************************************************************
      Variables
    *************************************************************************/
    IntShortMap range;

};

#endif
```

```
/*****************************************************************************

  TempoRange.cpp

  Class for holding tempo ranges, i.e. pairs of time and tempo.

  Author: Erik Jälevik

*****************************************************************************/

#include "TempoRange.h"

#include <sstream>

using namespace std;

/*****************************************************************************
  GetRangeString
*****************************************************************************/
string TempoRange::GetRangeString() {

  if (!range.empty())  {
    // Take first and last tempo and return them with a hyphen in-between
    ostringstream os;
    os << GetFirst() << "-" << GetLast();
    return os.str();
  }
  else {
    return "";
  }

}
```

Appendix B. Project Log

As the log is taken from the project web site, it is presented in reverse chronological order.

April 2004

27/4/04
Made some changes in view of the feedback and put all sections of the final report together.

26/4/04
Received some feedback on the draft report from supervisor.

23/4/04
Switched to OpenOffice instead and the figures were now behaving. Proofread and finalised report.

22/4/04
Finished diagrams and tried putting them all in with the text in a Word document. Gave up in the end as the figures just kept jumping around.

21/4/04
Finished tidying up the code. Produced most of the diagrams needed.

20/4/04
Started drawing diagrams needed for report. Started tidying up the code.

18/4/04
Worked on the report. Implemented Export to CSV function.

17/4/04
Worked on the report.

16/4/04
Worked on the report.

15/4/04
Worked on the report.

14/4/04
Realised that some of my implemented code wasn't as modular as the design dictated so spent some time refactoring it to remove some of the coupling. Continued writing up the design chapter.

13/4/04
Started writing on the Design chapter in the final report.

12/4/04
Further testing and tweaking of beat tracker. It works almost satisfactory now. If using overall tempo, it correctly tracks all but one of the test tracks. If tracking changes, it doesn't track all of

them correctly all the way through, but they are tracked correctly for the most part of their duration.

11/4/04
Realised that the bandpass filters weren't operating properly when using other sample frequencies than 22,050 Hz. So added filter coefficients for all the different sample rates. Also implemented the post-tracking heuristics for improving a tempo range and for working out overall tempo.

8/4/04
Wasted a lot of time hunting down some elusive bugs. Equipped beat tracker with lots of output facilities and wrote some Matlab code to visualise this data in useful ways.

7/4/04
Implemented tracking of phase constancy in the resonators and it works! It improves beat-tracking performance on tracks that have a very triplet-like rhythm, like Carl Craig's "Science Fiction".

6/4/04
Tested beat tracker on test tracks at different sample frequencies and analysed results. It seems you can go as low as 8,000 Hz and still get reasonable results. Performance does get worse on tracks where a large part of the rhythmic content is in the treble however. Also came up with the concept of phase constancy to be able to weight the reliability of a tempo estimate. I'm not sure if it's workable implementation-wise but I will experiment further with it.

5/4/04
Since it was easy to implement, I added phase estimation to the beat tracker. It now predicts the times of downbeats too so that you can output a file containing the analysed music with a drum sound at the points where the tracker thinks there is a downbeat. This should help debugging and fine tuning as you can hear exactly where the algorithm is going wrong. It can also be used for the real-time tracking when I get round to adding that.

3/4/04
Picked out a selection of test tracks and prepared 30-second snippets of them for properly testing the beat tracking. Started looking at tidying up and improving the beat tracker.

2/4/04
Changed error reporting so that more technical info is only displayed in error dialogs in debug builds. Made MadWrapper able to detect invalid MP3s. Introduced transactions and indices in the database in an attempt to speed it up. This improved batch deletion considerably but sorting and filtering is still slow.

1/4/04
Continued tidying up MP3 stream and the other input streams. Added support for parsing of ID3 tags.

March 2004

30/3/04
Implemented sample rate conversion for both the MP3 stream and the CD stream. Added GetProgress method to all streams.

29/3/04
The only way to solve the buffering problems was to use the MAD low-level API instead of the high-level one. I implemented this today and it works fine. Left to do is taking care of different sample rates. Currently an MP3 encoded in 11 kHz comes out chipmunk style.

25/3/04
Got MP3 decoding to work but there are some problems with buffering and starting/stopping streams.

24/3/04
Started implementing MP3 support using the MAD library directly.

23/3/04
Tidied up a few things in connection with Options dialog. Fixed some bugs and made some improvements to FilterPanel and CheckableDirCtrl. Found and implemented a nice application icon and fixed About window.

22/3/04
Finished Options dialog.

12/3/04
Started on Options dialog.

11/3/04
Started structuring final report thinking about what sections it needs to have etc.

6/3/04
Implemented tapper and linked it from MusicTrackDialog. Fixed some bugs in the same dialog and added functionality for deleting, inserting tempi and for playing tracks. Also rewrote Icons class to only use static data.

4/3/04
Implemented configuration file support. All customisable aspects of the program such as window sizes, column widths, last entered filter criteria etc are now maintained between sessions. Also changed the main display so that unverified tracks appear in red.

1/3/04
Fixed some bugs to do with CDInStream and others. Added support for automatic addition of new genres. Moved file parsing support to FileInStream.

February 2004

29/2/04
Implemented parsing of returned FreeDB data and finished the AddCDDialog.

24/2/04
The AddCDDialog can now retrieve CD information from the CDInStream and also query FreeDB. Still need to implement parsing of returned FreeDB data. Spent some time tracking down a bug in the version of the AKRip library I was using. Problem was solved by upgrading to a newer version of the AKRip DLL.

23/2/04
Finally managed to fix the bug that made CD reads only work once. Also started adding functionality to CDInStream to allow the AddCDDialog to query it for CD information.

22/2/04
Implemented visual part of AddCDDialog. Need to think about how it should get access to the CD reading class. Also made some changes to how track information is passed around from the add dialogs to the progress dialog to the beat tracking engine. Had to add an extra field to the MusicTrack class ("access") for consistency in accessing the source for the data.

16/2/04
Spent some time fixing a bug that made the program crash when trying to delete multiple tracks. Implemented progress dialog for beat tracking.

15/2/04
Finished the checkable directory tree control and tied the Add Files dialog in with the back end. Files can now be added and beat-tracked through the interface. Although there is no progress window yet to tell the user what's going on.

14/2/04
Changed strategy and started looking at using two windows and enable drag and drop between them to select files. But this proved difficult to implement, mainly because of problems with the way the wxWindows directory control handled multiple selections. So I went back to the check box approach and got it to work thanks to some advice from some people in the wxWindows newsgroup.

11/2/04
Spent most of the day trying to customise the wxWindows directory tree control to display checkboxes to allow multiple selection. Suffice to say it's not going well. Mouse clicks aren't captured and it's not behaving at all.

10/2/04
Created menus, toolbar with icons and started on Add Files dialog.

5/2/04
Made some improvements to SQL statements in MusicCollection. Implemented filter bar for main collection window.

2/2/04
Used wxDesigner to create a proper version of the track info dialog. Implemented validation, saving etc. Still has some bugs.

January 2004

31/1/04
Continued work on track info dialog. Got it working and integrated into the main screen but it still looks crap. Realised how time-consuming it was to handcode dialogs so downloaded wxDesigner, a visual editor for GUI components, for evaluation.

30/1/04
Started working on the dialog window for displaying detailed track info.

29/1/04
Contents of collection are now displayed correctly by the main window. Sorting works but there are some bugs to iron out.

28/1/04
Started implementing GUI.

27/1/04
Spent all day finishing off MusicCollection and related database classes. All functionality should be there now including filtering and sorting although more testing is needed. Also decided to scrap the idea of having several genres per track, mainly because there is just no way of sorting on genre if that's allowed. The choice of which of a track's several genres should be sorted would be arbitrary.

26/1/04
Looked into MadSam, a C++ wrapper for the MAD MP3 library. It's not completely painless but still better than using the raw library. It does need boost installed though, will need to investigate if this has any effect on executable size etc. Also continued with database classes, only the filtering functionality is left to implement.

20/1/04
Continued work on database classes. Most of the functionality is now there except filtering and some other bits.

19/1/04
Did some more tweaking with the main algorithm. Changed the envelope followers from using LPFs to using convolution with a Hanning window just like in Scheirer's paper. This made the envelopes much cleaner and improved performance. The tracker now classified "Rae" correctly (at least part of the way) but is still having problems with noisy stuff like songs with lots of guitar feedback. I'm not sure whether it's a good idea to weight all subbands equally. If there's not much rhythmic activity in one band at all, this band's resonators' semi-random fluctuations should then not count as much as a band with clearly marked rhythm. Solved this for now by implementing a threshold for envelope derivatives.

18/1/04
Continued working on MusicCollection. Communication between the program and the database is now working. Thought about the interaction between the wxListCtrl displaying the data and the MusicCollection class. Due to various complications, I decided to try out a simple approach first where tracks are not cached, but instead brought into memory from the database as requested by the list control. If this turns out to be to slow, it should be possible to cache the records in a map or similar data structure within the MusicCollection class.

17/1/04
Finished tidying up exceptions in BeatTracker and InStream classes. Performed some tests on the SQLite database. Tests went well and library was easy to work with so I went ahead and implemented the database model in SQLite. Also started implementing C++ database classes. Wrote TempoRange, MusicTrack and started on MusicCollection.

16/1/04
Mainly fiddling with exceptions and hunting down a memory leak. The leak was due to a dynamically allocated array in the TrackTempo method that didn't get deleted if an exception was thrown.

14/1/04
Spent hours implementing CDInStream. Since AKRip forces you to deal with number manipulation at the byte level it took lots of code to get it to work properly.

13/1/04
Looked into how to decode MP3s with the MAD library and it's quite involved as the library only does decoding, it doesn't take care of input/output or anything else. Found a post on the MAD mailing list about a C++ wrapper to which I replied requesting more info. Wrote Observer and Observable classes and subclassed BeatTracker off the Observable. Started writing the CDInStream class with the help of the AKRip library. The library is far from ideal. It's very low-level and entirely in C so it's proving quite time-consuming to use.

12/1/04
Wrote a mail to Scheirer asking about a few steps in his algorithm.

2/1/04
Changed a setting in the resonators (beta) which made the beat tracker not always go for the lowest fraction. The performance is still not great, on a rhythmically complex track ("Rae" by

Autechre), it estimated a tempo which was 2/3 of the actual tempo. I would like to investigate this further and try to reduce the occurrences of this but today is the planned deadline for implementing the beat tracker.

December 2003

30/12/03
The beat tracker is working! It seems to detect the tempo quite well but for some reason it always goes for the lowest fraction of the tempo within the detected range. More investigation is needed.

29/12/03
Worked on the spacing of the resonators and realised that the best thing would probably be to determine the number of resonators dynamically at runtime depending on the envelope sampling rate and tempo range used. This way, one resonator for every possible envelope sample range can be used and the resolution available will be neither under or over utilised.

28/12/03
Implemented Resonator class and experimented with comb filter settings.

27/12/03
Finetuned attack and release times for envelope follower, did the downsampling and derivative of envelope plus spent a lot of time trying to work out exactly how to implement the comb filters. It seems the comb filters will not correspond to whole BPM values as I thought they would. This means there will not be a one-to-one correspondence between filters and BPM values and therefore it might not be possible for the accuracy to always be within 1 BPM as per the requirements.

26/12/03
Implemented bandpass filters for splitting signal into frequency subbands and wrote envelope follower. Realised that to get accurate envelopes for the different subbands, the attack and release times for their envelope followers need to vary. The higher the subband frequency range, the shorter the attack and release.

23/12/03
Continued work on FileInStream, started implementing TrackTempo method of ScheirerTracker.

22/12/03
Wrote skeleton classes for BeatApp, BeatFrame, BeatTracker, ScheirerTracker, InStream and FileInStream.

21/12/03
Drew up data model, set up bug database and started coding.

10/12/03
Finished interim report.

9/12/03
Decided that I was not getting much further in the design and that it would be more productive to start doing some coding. I'm also unsure whether it wouldn't be a waste of time to draw nice sequence diagrams for everything. It would probably be faster just to specify the behaviour that needs specifying in the form of pseudo-code.

8/12/03
Did some more work on the design, mainly investigating wxListCtrl and how it would integrate with the model and the database.

4/12/03
Finished first draft of interim report.

2/12/03
Met with supervisor. He suggested I got a version of the interim report ready in a few days time so he can have a read through and suggest any changes before the deadline. Apart from that, mainly discussed general software engineering issues and current music-related research. Wrote large part of interim report.

1/12/03
Started on interim report laying out the general structure for discussion in tomorrow's meeting with Drew. Also extended project plan to include the tasks completed prior to setting up this website as a complete plan needs to be included in the interim report.

November 2003

28/11/03
Did some testing of STK's capabilities for reading, manipulating and writing wav files. It was incredibly easy and convenient and worked pretty much straight away.

26/11/03
Downloaded the demo version of Native Instrument's Traktor, an MP3 DJ tool. It was quite a demotivating experience as it's doing exactly what I am trying to do plus a lot more in a very nice way indeed. Its beat-tracking seems fast and reasonably accurate. Oh well, at least my program will hopefully be the first free program to let you analyse tempo and build up a database.

25/11/03
Did further work on the high-level design. Also fleshed out most classes with method names and have got most of the low-level design done. Did a lot of thinking about patterns and how to fit them in. The Observer pattern will have to use multiple inheritance for the GUI observer classes but this is OK since the abstract Observer class is really just an interface. Also decided to use the Strategy pattern for the relationship between a BeatTracker and its different input streams.

24/11/03
Added a high-level design.

23/11/03
Been working on high-level design but am finding it hard to work out how to lay out the classes according to a clean model-view-controller approach.

21/11/03
Researched libraries for reading CD and MP3 audio.

20/11/03
I've been too busy with assignments the last couple of weeks to get much done on the project. Therefore, I decided to skip the testing of some of the libraries as there just won't be enough time. Today I've done some research into suitable databases. Also fleshed out the project plan as a full one needs to be included in the interim report. It now outlines everything that needs to be done until the end of the project with very rough guesstimates as to how long each step will take.

8/11/03
Finished looking at Scheirer's algorithm and now have a good idea of what's needed to implement it. Also got a response from Eric Scheirer himself to a question asked in the Music DSP mailing list! Apparently the envelope calculation can be performed with a simple low-pass IIR filter. Was also advised to look for code for "envelope followers".

5/11/03
Due to all the evil assignments given to us, my plan for this week has not exactly been followed. I decided to reshuffle the plan a bit and focus on Scheirer's algorithm and the audio library components needed before testing wxWindows as these things seem to be higher risk. I spent most of today reading my DSP book trying to understand the concept of convolution. Which I now do. Sort of. I'm still not entirely sure how to implement it though, the method described in the book seems very slow. I've got a feeling there is some recursive filter that can do it but not really sure where to find one. I've also realised that the DSP knowledge required to implement, and fully understand, Scheirer's algorithm might be greater than I first thought.

October 2003

31/10/03
Added risks section to requirements spec. Did some prototype GUI sketches and worked out what GUI controls will be needed.

30/10/03
Added data requirements, scenarios and some other further requirements to the requirements specification. Wrote first version of project plan. Added cumulative bibliography.

29/10/03
Decided to write all project documentation in (X)HTML, created this web site and transferred proposal and log to it. Wrote large part of requirement specification.

28/10/03
Meeting with supervisor. Discussed next stage of project and agreed that the next thing to do would be to write up a more detailed requirements document. Furthermore, I will draw up a plan detailing what needs to be done before design of the software can begin.

21/10/03
Wrote proposal.

10/10/03-27/10/03
As a way of evaluating and getting to know the wxWindows library, I have been writing a small GUI application for generating Winamp playlist files. The library seems good so far.

10/10/03
Initial meeting with supervisor presenting project idea. Agreed that project idea seemed suitable for a 3rd year project.

1/9/03 – 21/10/03
Researched beat detection, C++ programming and digital signal processing.

Bibliography

Publications

Allen, P.E., Dannenberg, R.B. (1990), “Tracking Musical Beats in Real Time”, 1990 International Computer Music Conference.

Baray, C., “The Model-View-Controller Design Pattern”, online article, http://www.cs.indiana.edu/~cbaray/projects/mvc.html (up on 24/11/03).

Burbeck, S. (1992), “Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)”, online article, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html (up on 13/4/04).

Cliff, D. (2000), “Hang the DJ: Automatic Sequencing and Seamless Mixing of Dance Music Tracks”, HP Labs Technical Report HPL-2000-104.

Dixon, S. (2001a), “Automatic Extraction of Tempo and Beat from Expressive Performances”, Journal of New Music Research, Vol. 30 No. 1, 2001.

Dixon, S. (2001b), “An Interactive Beat Tracking and Visualisation System”, Proceedings of the International Computer Music Conference, Havana, Cuba, 2001.

Eckel, B. (2000), *Thinking in C++ Volume 1*, online book, www.mindview.net (up on 30/10/03).

Eckel, B., Allison, C. (2004), *Thinking in C++ Volume 2*, Pearson Prentice Hall.

Foote, J., Cooper, M. (2001a), “Visualizing Musical Structure and Rhythm via Self-Similarity”, Proceedings of the International Computer Music Conference, Havana, Cuba, 2001.

Foote, J., Uchihashi, S. (2001b), “The Beat Spectrum: A New Approach to Rhythm Analysis”, Proceedings of the International Conference on Multimedia and Expo (ICME) 2001.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995), *Design Patterns*, Addison-Wesley.

Goto, M., Muraoka, Y. (1995), “Music Understanding at the Beat Level - Real-time Beat Tracking for Audio Signals”, IJCAI-95 Workshop on Computational Auditory Scene Analysis.

Goto, M. (2001), “An Audio-based Real-Time Beat Tracking System for Music with or without Drum Sounds”, Journal of New Music Research Vol. 30 No. 2, 2001.

Johnson, J. (2000), *GUI Bloopers*, Morgan Kaufmann.

Josuttis, N. (1999), *The C++ Standard Library: A Tutorial and Reference*, Addison Wesley.

Kempster, C. (1996), *The History of House*, Sanctuary.Lee, G. (1993), Object-Oriented GUI Application Development, Prentice Hall.

Liberty, J. (2002), *Teach Yourself C++ in 24 Hours*, Sams.

Lynn, P., Fuerst, W. (1997), *Introductory Digital Signal Processing with Computer Applications*, Wiley.

Lyons, R. (2004), *Understanding Digital Signal Processing*, Addison-Wesley.

Meyers, S. (1996), *More Effective C++*, Addison Wesley.

Nielsen, J. (1994), “Ten Usability Heuristics”, online article, www.useit.com/papers/heuristic/heuristic_list.html (up on 29/10/03).

Patin, F., “Beat Detection Algorithms”, online article, www.gamedev.net/reference/programming/features/beatdetection (up on 30/10/03).

Pressman, R. S., Ince, D, (2000), *Software Engineering - A Practitioner's Approach*, McGraw Hill.

Robertson, J., Robertson, S. (2003), “Volere Requirements Specification Template”, online article, www.systemsguild.com/GuildSite/Robs/Template.html (up on 29/10/03).

Scheirer, E. (1998), “Tempo and Beat Analysis of Acoustic Musical Signals”, Journal of the Acoustical Society of America January 1998.

Scheirer, E. (1998), “Frequently Asked Questions: MPEG, Patents, and Audio Coding”, online FAQ, web.media.mit.edu/~eds/mpeg-patents-faq (up on 10/12/03).

Scheirer, E. (2000), *Music-Listening Systems*, PhD thesis, MIT Media Lab.

Sethares, W., Staley, T. (2001), “Meter and Periodicity in Musical Performance”, Journal of New Music Research Vol. 22 No. 5 2001.

Tzanetakis, G., Essl, G., Cook, P. (2002), “Human Perception and Computer Extraction of Musical Beat Strength”, Proceedings of the 5th International Conference on Digital Audio Effects.

Software Libraries and Applications

AKRip, CD audio library, akrip.sourceforge.net/intro.html (up on 10/12/03).

Berkeley DB, database, www.sleepycat.com (up on 10/12/03).

ID3Lib, ID3 tag reading library, id3lib.sourceforge.net (up on 27/04/04).

libsndfile, audio library, www.zip.com.au/~erikd/libsndfile (up on 10/12/03).

MAD, MP3 audio library, www.underbit.com/products/mad (up on 10/12/03).

Metakit, database, www.equi4.com/metakit.html (up on 10/12/03).

PortAudio, audio library, www.portaudio.com (up on 10/12/03).

Qt, GUI framework, www.trolltech.com/products/qt (up on 10/12/03).

Sig++, audio library, sig.sapp.org (up on 10/12/03).

SQLite, database, www.sqlite.org (up on 10/12/03).

Synthesis Toolkit in C++ (STK), audio library, www-ccrma.stanford.edu/software/stk (up on 10/12/03).

Traktor, MP3 DJ application, www.native-instruments.com/index.php?traktor2_us (up on 10/12/03).

wxWidgets, GUI framework, www.wxwidgets.org (up on 27/04/04).