Summercode project report

Sakari Bergen

August 29, 2008

Contents

1 Overview

1.1 Introducing Ardour

Ardour[1] is a free software hard disk recorder and digital audio workstation application released under the GNU General Public License. It is written in C++ and currently runs on Linux, Solaris, and Mac OS X.

Ardour is currently the most promising open source audio workstation available that strives to meet the needs of professional users. Its development is currently sponsored by The School of Audio Engineering (SAE)[2] to the extent that, together with donations from users, it has been possible for the primary author, Paul Davis, to work on Ardour full-time. In addition to Davis, the Ardour Project has a bunch of active contributors with skills ranging from translations, usability and concept work to coding new features and fixing bugs.

Ardour is divided into two main development branches: 2.0 and 3.0. The main difference in the 3.0 branch is MIDI[3] sequencing support, but other design changes have been made also. MIDI sequencing support started out as a Google Summer of Code project in 2006 and continued in Summer of Code 2007. Primary development focus is gradually shifting from from the 2.0 branch to the 3.0 branch as the code in 3.0 is starting to stabilize.

1.2 My project

The basis of my project is concept work[4] for renewing Ardour's export dialog by industrial designer Thorsten Wilms. The goals of the redesign are based on a user survey and aim for more versatile and streamlined functionality. The main effort is in improving flexibility and usability, but some new technical features are also introduced. The development was done in a dedicated branch based on the 3.0 branch.

2 List of planned features

2.1 Meta data

Meta data support was not part of the export redesign plan, but was something that fitted in, since meta data can be used to tag exported files.

- A set of XML-elements for defining relevant meta data will be defined
- Backend functionality for storing and editing this data will be added
- A GUI component for editing meta data will be added

2.2 Export

- An option for trimming and/or adding silence to exported files will be added

[1] `http://ardour.org`- Ardour website
[2] `http://ardour.org/node/976`- SAE Institute Sponsors Ardour Open Source DAW Project
[3] `http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface`- Musical Instrument Digital Interface
[4] `http://thorwil.files.wordpress.com/2007/08/export_design_2007-08-11.pdf`- Ardour Export Redesign

- An option for normalizing exported files will be added
- A system for semi-automatic file naming will be added
- An option for real-time export will be added
- Basic support for exporting with different encodings and containers shall be added to the backend. This will include lossily and losslessly compressed formats such as Ogg Vorbis and FLAC.
- The amount of encodings implemented will depend on how long implementing them takes. The goal is to add a couple of useful options to allow proper implementation of the rest of the project, rather than cover a multitude of encoding options.
- Support for basic multichannel export will be added
- The basic structures for storing and editing audio format profiles will be added to the backend
- A GUI component for editing these profiles will be created
- An option for exporting multiple files will be added
- A new GUI for the export dialog will be implemented as described in Thorsten Wilms' document

2.3 Tagging exported files

- Tagging support for exported files will be added

3 Implementation

This section includes descriptions of how specific features were implemented in the project. Files referenced can be found from the project SVN repository branch at `http://subversion.ardour.org/svn/ardour2/branches/sbergen`.

Ardour makes heavy use of the Model-view-controller pattern, and thus code is divided into the Ardour engine library in `libs/ardour` and the GUI in `gtk2_ardour` .

3.1 Meta data

3.1.1 Storage and modification

Basic functionality for storing and editing meta data in session files was implemented in the backend. The XML format used is simply in the form of `<field>value</field>` , which will probably change later. However, the technical implementation allows changing the format with minimum effort, as long as a final format is designed before the release of Ardour 3.0.

Files

- `libs/ardour/ardour/session_metadata.h`
- `libs/ardour/session_metadata.cc`

3.1.2 Tagging

Tagging of exported files was implemented using the TagLib[5] library. A class that reads meta data stored in the session and writes it to exported files was created (AudioFileTagger).

Files

- `libs/ardour/ardour/audiofile_tagger.h`
- `libs/ardour/audiofile_tagger.cc`

3.2 Export

3.2.1 General work flow

One exported file in Ardour is defined by four components, each represented by a C++ class:

- Timespan (ExportTimespan)
- Channel configuration (ExportChannelConfiguration)
- Format specification (ExportFormatSpecification)
- File location (ExportFilename)

Additionally one class is needed to handle these components (ExportHandler). When files are exported, the following steps take place:

1. One or more instance of each component is created and configured
2. Any number of combinations of these components (each combination having one of each type) is registered to the handler
3. Export is requested from the handler
4. The handler reads each relevant channel for one timespan into a temporary file
5. The individual channels are combined into a multichannel file and processed as necessary
6. The multichannel file is encoded and written to the requested location
7. Steps 5.-6. are repeated for each configuration registered for this timespan (each configuration consisting of one channel configuration, format and file location)
8. Steps 4.-7. are repeated for each timespan registered to the handler

This work flow allows very complex export configurations to be used fairly simply and efficiently. However, with very simple exports a slight overhead is present.

[5] `http://developer.kde.org/~wheeler/taglib.html`- TagLib Audio Meta-Data Library

Files

- `libs/ardour/ardour/export_timespan.h`
- `libs/ardour/export_timespan.cc`
- `libs/ardour/ardour/export_channel_configuration.h`
- `libs/ardour/export_channel_configuration.cc`
- `libs/ardour/ardour/export_format_specification.h`
- `libs/ardour/export_format_specification.cc`
- `libs/ardour/ardour/export_filename.h`
- `libs/ardour/export_filename.cc`
- `libs/ardour/ardour/export_handler.h`
- `libs/ardour/export_handler.cc`

3.2.2 Data flow

Because the audio data flow in the export process inherently changes sample format and length due to conversions, a generic templated graph system was created. The system consists of three abstract classes:

- GraphSink - represents a final destination for data. It has an input type and is capable of writing data of this type to a destination.
- GraphSource - represents a source of data. It has an output type and is capable of reading data of this type from a source.
- GraphSinkVertex - represents a vertex for data that processes data and passes it through. It has both an input and output type. Data of the input type can be written to the vertex. This data is then processed and converted to the output type if necessary. The processed data can be piped to another GraphSinkVertex or a GraphSink.

Files

- `libs/ardour/ardour/graph.h`

3.2.3 Export processors

Several different processing operations need to be applied to audio data during export. This includes normalizing, sample format conversion, sample rate conversion and encoding. A class for each task was created implementing the graph interface described in section 3.2.2. Also, a class for organizing and operating these processors according to format specifications was implemented.

Files

- `libs/ardour/ardour/export_utilities.h`
- `libs/ardour/export_utilities.cc`
- `libs/ardour/ardour/export_processor.h`
- `libs/ardour/export_processor.cc`

3.2.4 File I/O

Sound file input and output in Ardour is done using libsndfile[6]. There was no reason to change this, so all new code uses libsndfile also. All export related file readers and writers implement the graph interface described in section 3.2.2.

Data in temporary files is stored in raw 32-bit floating point format. The temporary file class also includes functions for trimming and adding silence to/from the beginning and end of files. Only absolute silence (sample values equal to zero) is detected by the silence trimming algorithm.

Files

- `libs/ardour/ardour/export_file_io.h`
- `libs/ardour/export_file_io.cc`

3.2.5 Encoding options

A not yet released version (1.0.18 pre22) of libsndfile was used to support FLAC and Ogg Vorbis in Ardour's export. All relevant formats supported by libsndfile were made usable in Ardour's export, but no additional libraries were used in order to support more formats. However, the possibility of later adding support for encoding options via additional libraries was kept in mind during the design process of encoding related components.

Broadcast Wave Format (BWF) was already supported as a recording format in Ardour. A special class for BWF header writing was created to be used for exporting BWF files. This class can and should later be used in other parts of Ardour also.

Files

- `libs/ardour/ardour/broadcast_info.h`
- `libs/ardour/broadcast_info.cc`

[6] `http://www.mega-nerd.com/libsndfile/`- libsndfile, a C library for reading and writing files containing sampled sound

3.2.6 Real time export

Implementing real time export itself was rather trivial. However, it imposes one extra requirement: post processing has to be threaded. This is due to the fact that most processing in Ardour is done in JACK[7] process-cycles, which are time limited by the sound card buffer size. Sound card buffer sizes are in the millisecond magnitude, so clearly whole files can not be encoded during one process-cycle. Thus all export post processing is done in a separate thread (handled by the ExportChannelConfiguration class).

Files

- `libs/ardour/session_export.cc`
- `libs/ardour/ardour/export_channel_configuration.h`
- `libs/ardour/export_channel_configuration.cc`

3.3 Export Management

3.3.1 General process management

Since the export process is rather complex, and has many places where errors could be encountered, two classes were created to help manage the export process:

- ExportFailed - The exception that is thrown if an unrecoverable failure is encountered during export.
- ExportStatus - A structure used for indicating the current state of export. Also used for aborting export.

Files

- `libs/ardour/ardour/export_failed.h`
- `libs/ardour/ardour/export_status.h`
- `libs/ardour/export_status.cc`

3.3.2 The format compatibility system

The export format dialog was specified to have a compatibility mechanism to identify compatibility relations between different formats and compatibility targets. E.g. selecting the "CD" compatibility target would show that WAV, BWF, AIFF and RAW formats with a sample rate of 44.1kHz and sample format of 16-bit integers are the only compatible options.

A base class for all export format related classes was created and set-algebraic operations were defined for it. This way it was fairly simple to calculate which formats are compatible with specific compatibility targets. Three classes were derived from the base class (ExportFormatBase):

[7] `http://jackaudio.org/`- JACK Audio Connection Kit

- ExportFormat - A format like WAV or Ogg Vorbis, which includes general information for this format e.g. supported sample rates and sample formats. ExportFormat was made an abstract class, which was extended for more specific use.
- ExportFormatCompatibility - Defines a compatibility target
- ExportFormatSpecification - A format specification that includes all information necessary for file creation. Also includes normalizing and silence trimming settings.

To handle the rather complex interaction between the different components of the compatibility system, a manager class (ExportFormatManager) was also created.

Files

- `libs/ardour/ardour/export_format_base.h`
- `libs/ardour/export_format_base.cc`
- `libs/ardour/ardour/export_formats.h`
- `libs/ardour/export_formats.cc`
- `libs/ardour/ardour/export_format_compatibility.h`
- `libs/ardour/export_format_compatibility.cc`
- `libs/ardour/ardour/export_format_specification.h`
- `libs/ardour/export_format_specification.cc`
- `libs/ardour/ardour/export_format_manager.h`
- `libs/ardour/export_format_manager.cc`

3.3.3 Export profile management

Export format specifications and whole export setups can be saved for later use. This is done by serializing them into an XML format. Almost all export related serialization is done in the class that manages export profiles in general (ExportProfileManager).

In addition to ready format specifications that will ship with Ardour, the user can create his own specifications for the formats he uses often. Format specifications are always used from a saved profile.

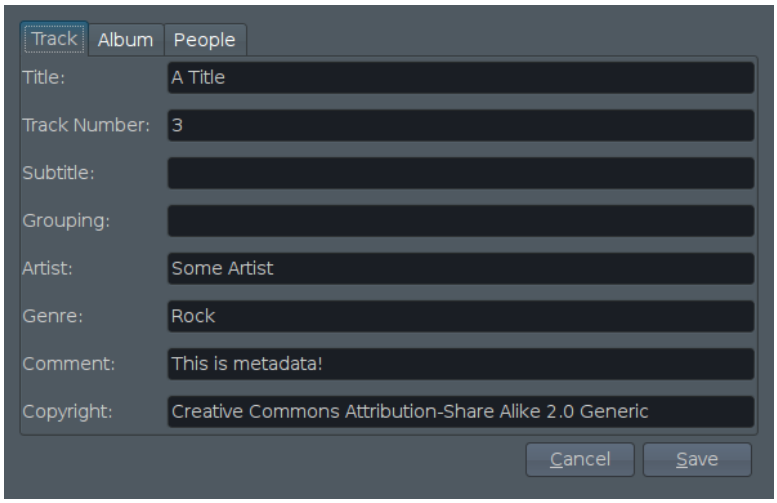
An export setup consists of one or more configurations, each including a timespan, channel configuration, format specification and file location. Whole setups can be saved and loaded explicitly, but the last used setup is also saved automatically with the session. The setup is split into a global and session specific part. The session specific part includes timespans and channel configurations, where as the global part includes format and file location. The path part of the file location is serialized as a relative path from the session directory when applicable.

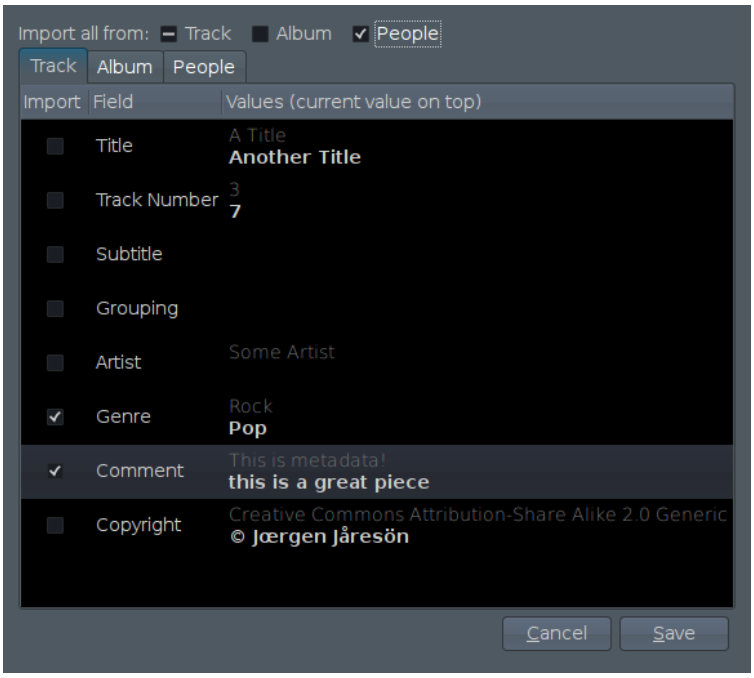
Files

- `libs/ardour/ardour/export_profile_manager.h`
- `libs/ardour/export_profile_manager.cc`

3.4 Meta data GUI

The GUI components for meta data were designed with extensibility in mind. In addition to an editing dialog, an import dialog was created as an extra feature. The definitions for the data fields in the dialog (including order ad grouping) is done only in one place in the code. Thus all dialogs inherently have the same layout and are thus easy to use. On the code level this was accomplished via usage of C++ templates, function pointers and polymorphism.





Files

- `gtk2_ardour/session_metadata_dialog.h`
- `gtk2_ardour/session_metadata_dialog.cc`

3.5 Export GUI

3.5.1 Overview

The export GUI consists of two parts, the main export dialog, and the format selection dialog. In the GUI section I will first present components in the main dialog, and then components in the format dialog. In the beginning of each section is a screen shot from the implementation and an original mock up from the plan. The mock ups are displayed above the actual screen shots.

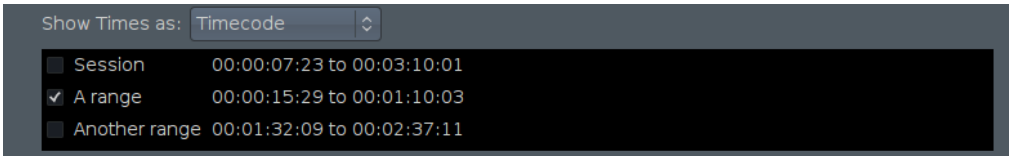
Files

- gtk2_ardour/export_main_dialog.h
- gtk2_ardour/export_main_dialog.cc
- gtk2_ardour/export_format_dialog.h
- gtk2_ardour/export_format_dialog.cc

3.5.2 Preset selector

The preset selector is a rather simple widget. It allows loading a preset by selecting it from the drop-down list. New presets can be created by typing in a name and clicking 'New'. The original design is lacking the 'New' button, but due to the ambiguity of the 'Save' button, the 'New' button was added to make a clear difference between creating a new preset and renaming an already existing one.

3.5.3 Timespan selector



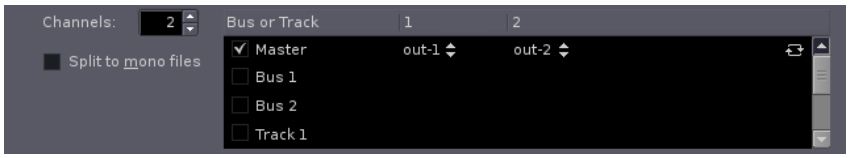
The original timespan selector was designed to allow selecting the session, current selection or multiple ranges. The implemented version allows selecting multiple timespans, including the session and current selection (when applicable), from a list.

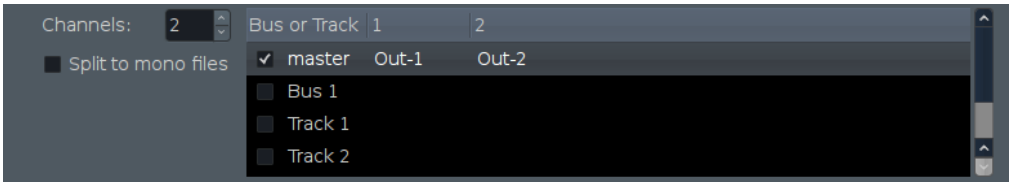
The CD marker export functionality was left out from this dialog, in favor of a separate CD export dialog with more functionality regarding CD markers. The backend code for CD marker export was made compatible with the new export components, but it is not available via the GUI at the moment.

Files

- gtk2_ardour/export_timespan_selector.h
- gtk2_ardour/export_timespan_selector.cc

3.5.4 Channel selector



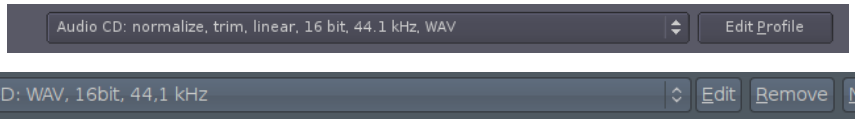


The channel selector implementation is very close to the plan. The only feature missing is the 'swap channels' function for stereo configurations. The widgets used in the implementation are standard gtkmm[8] widgets. The graphic outcome is a bit different from the plan, but works similarly. However, due to the nature of the standard widgets, some excess mouse clicks are required to operate the widget. Also the text in the port selector is falsely editable.

Files

- gtk2_ardour/export_channel_selector.h
- gtk2_ardour/export_channel_selector.cc

3.5.5 Format selector



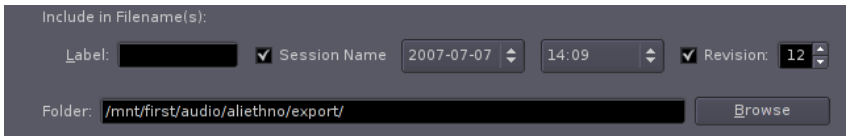
The format selector allows selecting, removing and adding format profiles. The 'Remove' and 'New' actions were moved from the format dialog (see section 3.5.9) to this widget to be more consistent with the rest of the dialog.

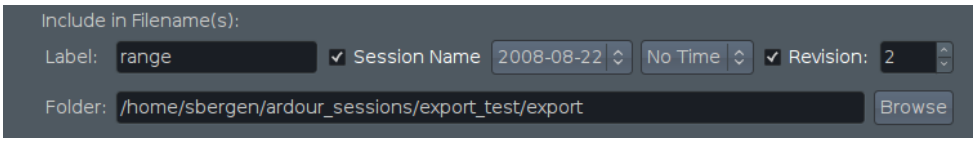
[8] `http://www.gtkmm.org/` - gtkmm,C++ Interfaces for GTK+ and GNOME

Files

- gtk2_ardour/export_format_selector.h
- gtk2_ardour/export_format_selector.cc

3.5.6 Filename selector



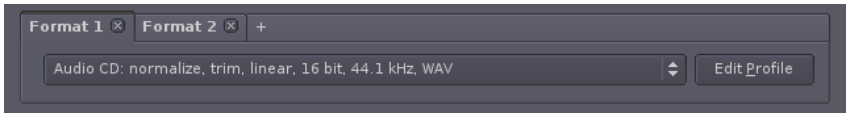


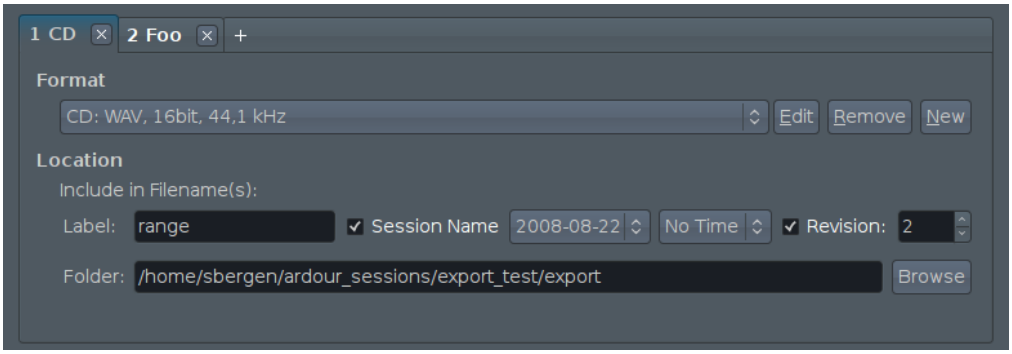
The filename selector allows the usage of smart filenames. The only difference between the plan and implementation, is the 'Folder' entry, which does not have auto-completion. This is due to the fact that no suitable component exists in gtkmm, and extra dependencies (libgnomeuimm) had to be avoided.

Files

- gtk2_ardour/export_filename_selector.h
- gtk2_ardour/export_filename_selector.cc

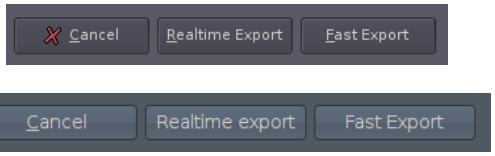
3.5.7 Multiplication





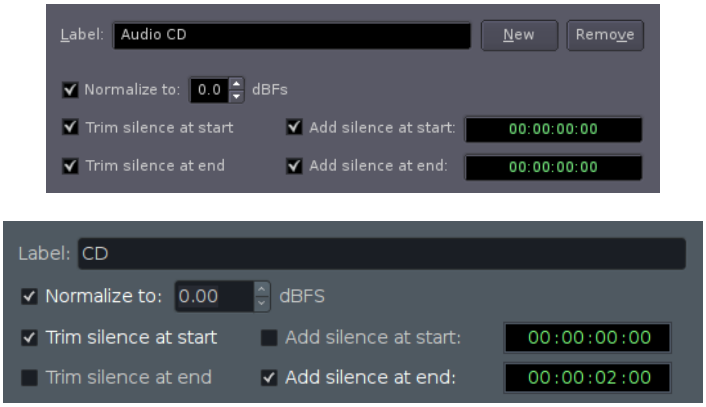
Multiple multiplication schemes were introduced in the plan. The final implementation is a version of the tabbed format multiplication depicted above. Instead of tabbing only the format, both format and location are tabbed. This way certain formats can be saved to special locations e.g. Ogg Vorbis to a shared folder and FLAC for personal use.

3.5.8 Actions



The action area of the dialog was implemented exactly as planned.

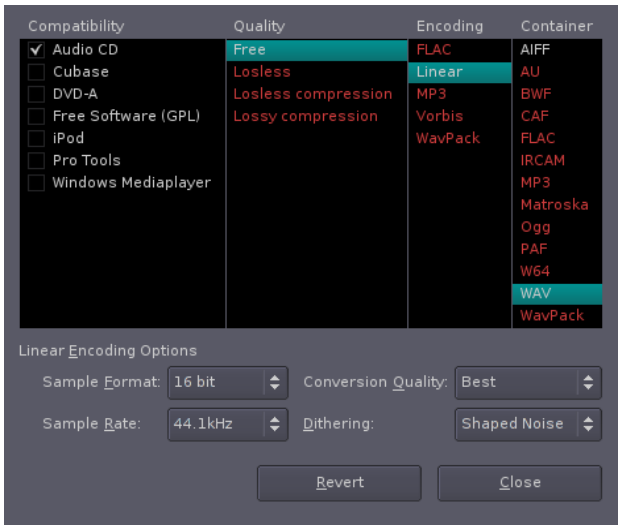
3.5.9 Format dialog part 1

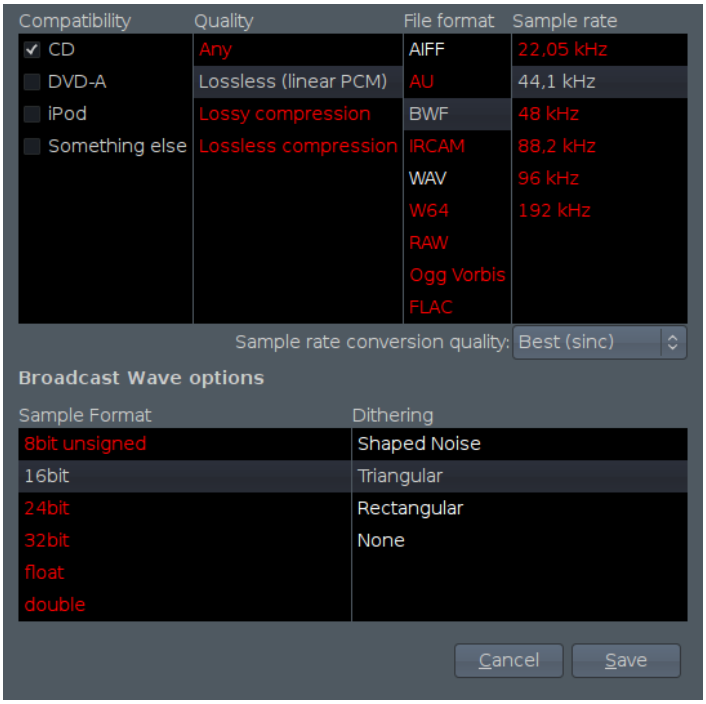


The top part of the format dialog was implemented as planned, except for the 'New' and 'Remove' buttons (as mentioned in section 3.5.5). Nevertheless, one minor design problem was encountered during implementation: times in the 'Add silence' selections can not be accurately converted between absolute time and bars/beats. This is due to the fact that the length of a bar/beat can vary depending on the location within the session (e.g. a bar at 00:00:00:00 can be longer than at 00:01:00:00 if the tempo has changed) and the selection in the dialog is not timespan (nor session) specific. However, this does not affect the actual functionality, as the durations are stored in the format used for entering them[9].

[9] Using times such as 6/8 and a duration of 5 beats will cause 5 beats to be used also in a 4/4 session. This leads to a different duration in bars (5/6 vs. 5/4), but is something that can not be avoided.

3.5.10 Format dialog part 2





The second part of the format dialog includes the actual format selection. The top part includes selections compatibility targets, format qualities, formats and sample rate. The bottom part is dependent on the selected format, and displays the related options. Some changes were made to this part:

- Encoding and container were combined into 'File Format' for more overall clarity.

- Sample rate was moved to the top row, because it is an option present in all formats.
- Sample format and dithering were in similar lists as other options, as they are also affected by the compatibility selection. This improves the visual feedback received from selecting different compatibility targets.

4 The project in retrospect

4.1 Schedule

The planned project schedule is presented in Figure 4.1. The schedule was well planned - the maximum deviation from it was no greater than one week:

- Meta data storage and editing was implemented in one week. Part of week two was used to implement meta data import.
- The format compatibility system was surprisingly complex to implement, and set off the schedule about one week from week 9 onward.

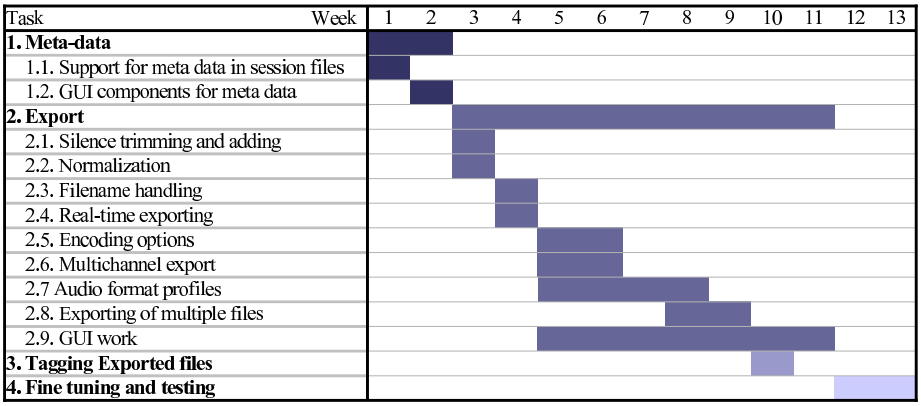


Figure 1: Project schedule

4.2 Plan vs. implementation

4.2.1 GUI

The project plan was not very strictly worded, so it left room for variation at several places. One of the biggest ones was the multiplication model. The most flexible model, the routing table, where any number of different export components could be combined, turned out to be too complex to be implement in this project. Some code was written for this model, but it ended up not being used after the decision was made to implement a tabbed multiplication model instead. The routing table model is a concept that was seen too complicated also for users - operations on it are not always very intuitive and in single file configurations the sole existence of the table can be confusing.

The tabbed model was not strictly one from the redesign document. It uses the concept of tabbing formats, but adds the flexibility of associating a file's format to its location. This can be useful in many situations.

Most other changes in the GUI are changes that emerged from problems encountered during implementation, and were discussed with the original designer. These changes should be considered as improvements. However some changes were due to technical limitations already discussed in sections 3.5.4 and 3.5.6.

4.2.2 Engine

The engine functionality was implemented as planned. The only questionable part is the "trim silence" feature. The problematic nature of this feature emerges from the fact that absolute silence is rare in recordings. The current implementation is capable of detecting only absolute silence, leading to an outcome that often differs from expectations. To get a more practical result, one would have to define a threshold value for silence, measure signal power levels and trim anything that falls below the threshold.

4.3 General thoughts

All in all the project taught me some important things about problem solving in general. Problems can roughly be divided in two categories:

1. Problems that can easily be split into smaller sub-problems
2. Problems that need a lot of planning before they can be split into smaller sub-problems

Often problems that initially seem complex fall into the first category, and problems that seem simple fall into the second. This has a big effect on the amount of work that goes into implementing code for solving these problems. A seemingly complex task can be completed relatively fast by just tackling category 1 problems one sub-problem at a time. In the case of category 2 problems, the time spent for planning and implementing interaction between sub-problems, often takes more time than implementing the solutions to the sub-problems them self. Most of the time when something didn't go as planned during the project, it was due to complex interactions between seemingly simple components.

4.4 Mentor's thoughts

A few words by my project mentor Sampo Savolainen:

> In my opinion Sakari's Summercode project was very successful. The project's outcome is an implementation for a very complex user interface and functionality entity. The implemented features improve Ardour's export/mixdown functionality significantly. The changes were made to the next generation 3.0 branch by request of Paul Davis. From a development perspective the decision was a good one, but unfortunate for users who will have to wait to get their hands on the new functionality. Communication between Sakari and the Ardour development team worked well via IRC. Sakari was always reachable by developers when necessary, and he informed others beforehand of periods of absence. His attitude towards the development process was also excellent - Sakari didn't dwell on problems, but rather brought them up in IRC where they were solved together.

5 What next

5.1 Meta data

The XML format used for storing meta data was not planned very much - it was rather created as a fast and simple solution. The format could probably be still refined by someone knowing more about designing XML formats. While the current meta data is consumer and tagging oriented (track, album etc.), it is clear that session management related meta data is something that would be useful to have in Ardour sessions.

5.2 Export

5.2.1 Exchanging presets and format profiles

Presets and format profiles are currently identified by a unique id number given to them. However, these ids are unique only for one user[10]. This problem should be tackled with a more complex id system where ids consist of two parts: one identifying the system, and another identifying the preset or profile. This way if files are exchanged between users, conflicting preset/profile ids can be resolved keeping already existing references to them valid.

5.2.2 Formats

The addition of FLAC and Ogg Vorbis was clearly a big step forward, but there is no reason to stop here. Support for more formats is clearly a good thing. WavPack[11], for example, is currently the only free lossless codec capable of encoding 32-bit floating point samples (the native sample format of JACK), and is thus a format that should be supported.

5.2.3 Compatibility targets and format presets

Only a few compatibility targets were created during this project, mainly for testing purposes. To be a truly useful feature, more targets should be created. Also a bunch of general purpose format profiles should be made to be shipped with future versions of Ardour.

5.2.4 Channel selector

The channel selector implemented is far from perfect. This is mainly due to technical limitations implied by the standard widgets used in gtkmm. The implementation could be improved by implementing custom cell renderers for the widget used. However, another solution has also been presented. In this solution the current widget would be replaced by a matrix routing design[12] by Thorsten Wilms. This design would primarily be used in other parts of Ardour, but would fit well in the export dialog also.

5.2.5 Other export dialogs

The export dialog implemented in this project is a general purpose dialog. A few special purpose export dialogs, using the new functionality implemented during this project, should still be implemented. These include a CD and region export dialog.

[10] User presets and format profiles are stored in ~/.ardour3/export.

[11] `http://www.wavpack.com/` - WavPack homepage

[12] `http://thorwil.wordpress.com/2007/12/06/matrix-14/` - Matrix routing design by Thorsten Wilms