QUINCE
A modular approach to music editing

by Maximilian Marcoll
http://quince.maximilianmarcoll.de
mail@maximilianmarcoll.de

June 26, 2011

Abstract

This paper is an introduction to the basic concepts of the software “quince” as well as to some fundamental ideas which lead to its development. Quince is a modular, non-media-centric, open source editor that presents time based data in a very flexible way which enables quince to be used for a multitude of applications.
Because of its open and extensible structure quince can be interesting for all those who want to perform a variety of nonstandard operations on different types of data in a single project.
This document does not cover all the functionality of quince, nore can it serve as a tutorial or user guide. For tutorials and a more comprehensive discussion of features, please refer to the documentation provided on the quince website.

1 Introduction

Almost all artists, working with digital media, at some point ask themselves:

> What tools fit my needs best?

The problem that one encounters very often is that computer programs do not only carry out tasks and give us options and possibilities, but also determine the ways in which we use them. In order to find out what tools we need, we first have to find out what we actually want to do. This is not as easy as it might seem. We tend to do what technology suggests us and sometimes we even confuse those suggestions with our own ideas. In order to find out what we really want, it is necessary to critically reflect on the use of technology and to overcome the tendency to do what our tools were made for, what is easy and convenient. Sir Ken Robinson put it this way:

> "Being creative means challenging what you take for granted."

but:

> "The problem in challenging what you take for granted is, that you don't know what it is, because you take it for granted."[1]

There are many excellent programs available, for all kinds of specialized tasks. Usually, they are solutions to *one* particular problem. Quince, however, is not.
The process of composing music, as I understand it, is a complex network of many interconnected tasks which have to be carried out in ever changing orders on different stages of various kinds of data. Quince's structure corresponds to this nonlinear approach to the working process and is not bound to any traditional working paradigm. Its flat hierarchy and flexible modular structure offer full responsibility and control about the working process and the treatment of data. Even more, quince was designed to carry out operations I do not yet know I want it to be able to. The fundamental concept of quince is a genuinely digital approach that cuts some old strings in order to help users find suitable workflows for their respective situations.
Just like the fruit, the usage of the program quince requires quite some knowledge about its features, its strengths and weeknesses as well as some getting used to. Once one knows how to deal with it thuogh, the results can be very delicious.
The structure and working paradigm of quince are very different from those of traditional editors and digital audio work stations or sequencers. It was made for those who like to challenge the standard working processes and paradigms and who seek to *gain responsibility* for how they work with their material.

2 General Structure

2.1 Working Paradigm

Quince breaks with the classic "tape recorder & mixing desk" paradigm. Every digital audio work station presents audio data in (monophonic) tracks that can be mixed using a digital simulation of a mixing board. The only aspect of these models that quince does implement is the timeline. Quince does not represent data arranged in tracks and since there are no tracks, there also is no mixer to mix them. Instead, data is hosted by "Strips": Vertically aligned areas containing Layers in which data can be displayed and edited. There can be virtually any number of Layers in a Strip. Lay-

ers in a Strip are displayed on top of each other (just like in Photoshop [2]). Thus, data can be arranged in three dimensions (see Figure 1).

2.2 Modularity

Almost everything that does carry out significant tasks in quince is a plug-in. The views that display data, the components that play back sequences as well as functions which perform operations on data all are external bundles loaded at runtime. It is not necessary to rebuild the entire program to add new functionality.

2.3 Media Types

Quince is non media centric. The only constraint about data is that it has to be time based if it should be edited in quince. Wether an event represents an audio or video file, a note in a score or the execution of a shell script - it does not matter as long as the thing being represented somehow happens in time. However, in order to be drawn and played back appropriately, there need to be plug-ins that can understand the respective media type.

2.4 Hierarchy

Almost every piece of software is designed in a hierarchical way, so that some functions are easier to reach and some tasks are easier to achieve than others - depending on the judgement of the programer(s).
As stated earlier, it is my belief that a program's structure determines the ways in which we use it. In this sense, the design of the program (its hierarchy of functionality) might have a great impact on the (supposedly creative) choices that artists make while using it.

To strengthen the decisions of the user and to give him or her more freedom, quince implements a flat hierarchy in which all functionality is directly in one's grasp. Nothing is hidden in complicated drop down menu trees and no function is considered to be more important than any other. Of course there are always tasks (and workflows) that are complicated and involve multiple steps. Quince does however offer a way to combine multiple operations into one single step, in order to create more powerfull tools and to simplify the utilization of complicated operations. (See the chapters *Functions* and *FunctionGraphs* for an explanation.)

3 Data Representations

3.1 Events

An event in quince simply is a set of parameters. A parameter is a combination of a keyword (e.g. "volume") and a value (e.g. 0dB). Every event is created with a basic set of parameters like "start", "duration" and "description" that can be extended to virtually any number. Apart from a few reserved words, any string can be used as the keyword of a parameter.

3.2 Sequences

There actually is no difference between an event and a sequence in quince: Every event can contain an arbitrary number of sub-events. *If* an event contains sub-events it may be regarded as a sequence. Hence, it is a question of perspective wether something is an event or a sequence, which is why I prefer the term *object*. A selection of objects can always be *folded* into a sequence by the push of a button. Respectively, a sequence can always be *unfolded* in a similar manner.

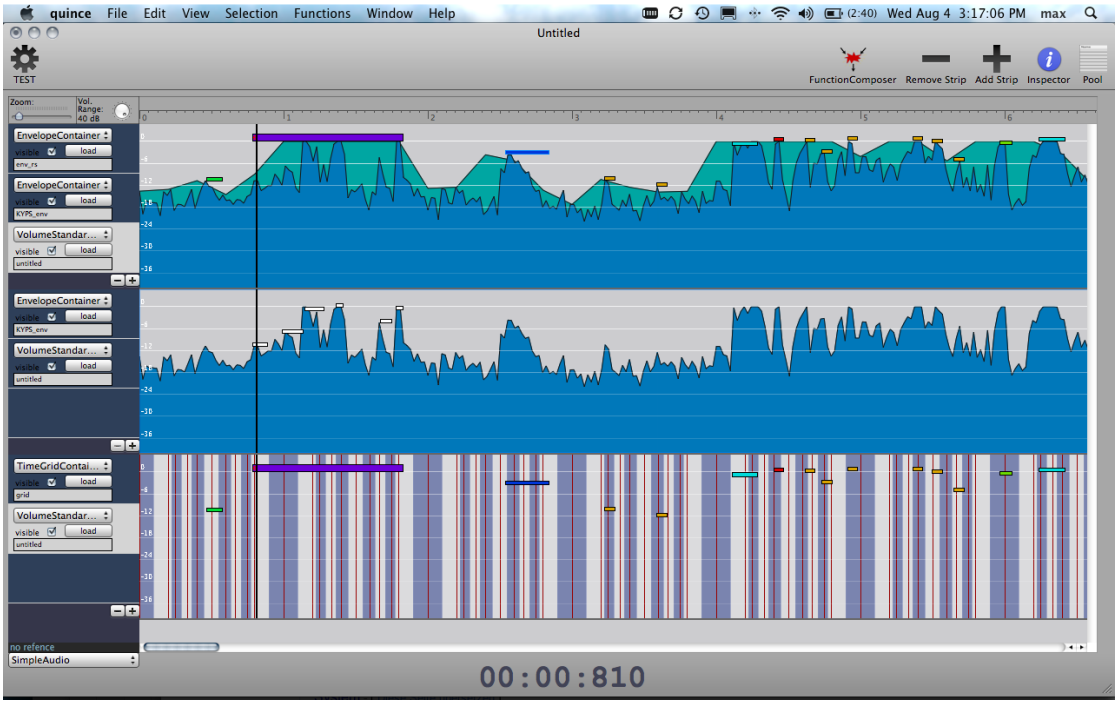

Figure 1: Project window with three strips containing multiple layers.

3.3 Data Types

For a better understanding of how data is being represented it might be helpful to know about a few internal data types used to distinguish between different kinds of objects:

- *QuinceObject*, the most basic type of object, used for Events and Sequences.[1]
- *Envelope*, a representation of a volume envelope, typically extracted from an audio file.
- *DataFile*, a reference to any kind of file.
- *AudioFile*, a reference to an audio file.

[1] *QuinceObject* also is the superclass of all other data types.

3.4 Visual Representations

3.4.1 ContainerViews

ContainerViews are plug-ins that display data and provide interfaces for the editing and arrangement of objects. There are different ContainerViews available for different *types* of data and for the display of different *parameters*. While the time related parameters ("start" and "duration") are *always* assigned to the x-axis of the ContainerView, the parameter that is displayed on the y-axis, is determined by the view. Every Layer contains one ContainerView to show its contents.

3.4.2 ChildViews

While ContainerViews display the contents of objects, ChildViews are plug-ins used to graphically represent the sub-events of the

object loaded by the surrounding ContainerView.

4 Manipulating Data

4.1 Functions

Functions are plug-ins which perform operations on objects. The result of the performance of a function can either be

- a new object,
- a change in the object the function operated on, or
- the export of data into an external file.

There already is a great variety of functions available. A few examples are:

- EnvToSeq (a transient detector),
- ImportFrequencies (imports analytical data created by Praat [3] and SPEAR [4]),
- LilyPondExport (the translation of sequences into LilyPond code[5]),
- Quantization,
- several functions for the alignment and equal distribution of objects in various parameters,
- functions performing the basic set operations, and many more.

4.2 FunctionGraphs

There is a way to create powerful customized tools right within quince: Functions can be combined, very similarly to the way in which events can be folded into sequences. Using the *FunctionComposer*, two Functions can be combined into a bigger unit, called *FunctionGraph*.
Not all combinations of functions are possible, though. Two functions are compatible if the output object of the first is of the same type as one of the inputs of the second. For example, if the output of the first function is an envelope (an object of type *Envelope*), and the second expects a sequence (an object of type *QuinceObject*), they can not be combined.
Although only two functions can be joined together at a time, much more complex FunctionGraphs can be built: FunctionGraphs behave just like functions, so you can combine them both with one another.

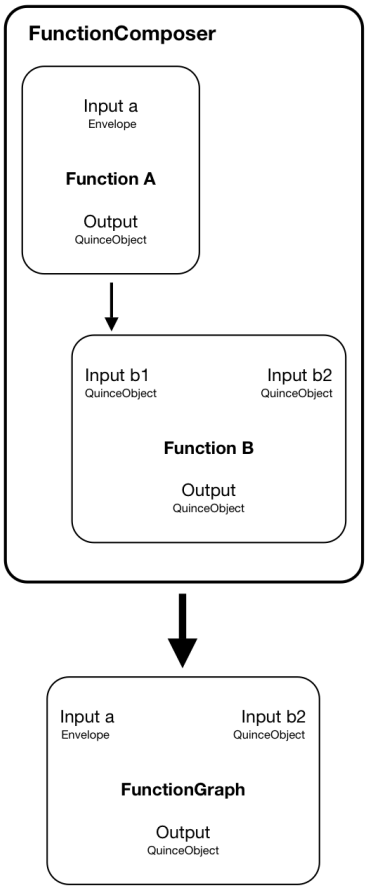

Figure 2: FunctionComposer Example

5 Playback

Objects are played back using *Players*, plug-ins which interpret an object's parameters in a certain way and produce output in real time. When playback is started, quince sends all the data to the currently selected player plug-in. How the player reacts to the data is dependent on its implementation.
There are currently two players available:

- The *AudioFilePlayer* that interprets events as references to audio files and
- *The CSoundPlayer*, an actual instance of Csound [6] that converts events into Csound score lines and performs the resulting Csound score in realtime using one of the template Csound instruments or a custom orchestra written by the user.[2]

6 Expandability

Quince is released under the GNU Public License[3]. It is written in Objective-C and can easily be extended by writing plug-ins (also in Objective-C). The QuinceApi is lightweight and simple to use. It was designed in such a way that in order to create a new plug-in, only a minimum of administrative code has to be written.[4] Players, Views and Functions can all be added by users to customize quince to their respective needs.

[2]The orchestra files used for the playback of sequences using the *CSoundPlayer* can be written directly in quince, too. In this way quince can also be used as a front end for Csound.

[3]http://www.gnu.org/licenses

[4]For a simple function plug-in that performs an operation on an object without creating a new output object: four lines.

7 Conclusion

It is a well known fact that new technologies can have a great influence on the works of artists. This is of course also true the other way around: Whenever art changes in respect of content and aesthetics, the techniques and technologies have to change as well.
Quince was developed to implement a new working paradigm that corresponds to new ways of composing music.[5] In addition to that, it opens up possibilities for decision where many traditional programs have automatic solutions. It is a working environment for all those who think beyond the boundaries of traditional software solutions.

8 Future Work

Although there already is a variety of plug-ins available, there are also still lots of things missing: FFT and pitch detection plug-ins, export and import to and from various formats (Midi, SDIF, MusicXML, to name a few), players that support video, OSC, amongst others.
So far quince only runs under Mac OS X. Although it should be possible to port it to Linux/Unix and Windows using the GNUStep framework [7], there are no plans to do so in the near future.

[5]Over the last years I developed a working method that I call nonlinear composition. The composition process can be split up into three stages: creation, disposition and processing of material. Where the paradigm of what I call *axiomatic composition* starts with a (conceived) nucleus - a singularity - and tries to derive almost everything from this central model, *nonlinear composition* is a bottom-up approach. It starts with a multiplicity (e.g. with a detailed transcriptions of recorded sounds), the material disposition is rhizomatic and pieces mostly are not written from start to end.

Of course, contributions of any kind are very welcome and much appreciated!

9 Acknowledgements

I would like to thank Paul Boersma and David Weenink for the development of the program Praat[3], one of the most innovative pieces of software I have come across. Many ideas concerning the structure of quince were inspired by Praat.
I would also like to thank Roman Pfeifer and Sönke Hahn for their help, support and shared thoughts.

References

[1] Sir Ken Robinson, Address during a honoraring degree ceremony, available at http://our.risd.edu/2009/06/02/sir-ken-robinson

[2] http://www.photoshopcafe.com/tutorials/layers/intro.htm

[3] http://www.fon.hum.uva.nl/praat

[4] http://www.klingbeil.com/spear

[5] http://www.lilypond.org

[6] http://www.csounds.com

[7] http://www.gnustep.org