# jAudio: Additions and Improvements

**Daniel McEnnis, Cory McKay, and Ichiro Fujinaga**
Music Technology Area, McGill University, 555 Sherbrooke West, Montreal, Quebec
{daniel.mcennis,cory.mckay}@mcgill.ca, ich@music.mcgill.ca

## Abstract

jAudio is an application designed to extract features for use in a variety of MIR tasks. It eliminates the need for re-implementing existing feature extraction algorithms and provides a framework that greatly facilitates the development and deployment of new features. Three classes of features are presented and explained—features, metafeatures, and aggregators. A detailed description of jAudio's dependency resolution algorithm is also discussed. Finally, ways in which jAudio can be embedded in and integrated with new systems are discussed, along with a description of jAudio's ability to add new features or aggregators, potentially at runtime.

**Keywords**: Java Audio Environment, Audio Feature Extraction, Music Information Retrieval.

## 1. Introduction

Since the introduction of jAudio at ISMIR 2005 [1], it has been incorporated as the feature extraction system for the OMEN project [2], which embeds jAudio in a larger framework that performs distributed feature extraction on a distributed dataset. This involved the addition of various new functionalities in jAudio. This paper describes the most significant of these improvements. In addition, jAudio's feature dependency resolution mechanism, one of its key aspects, is described in detail, as it was mentioned only briefly in the original paper.

## 2. Related Work

A comparison of existing MIR-oriented feature extraction systems is available in [1]. There have been two particularly significant developments in feature extraction since then that should be reviewed.

Marsyas has been updated so that it can be configured using a scripting language that makes it relatively easy to control which features are selected for extraction. It also now supports distributed feature extraction [3]. However, users are still required to manually resolve dependencies between features, and adding new features to the system requires recompilation.



Begrestra et. al. [4] have introduced a new class of features that they refer to as "aggregate features." These take as input a sequence of vectors created by another feature and generate either a single vector or as smaller sequence of vectors.

While CLAM [5] has made many improvements in the past year, its essential feature extraction components are essentially unchanged from the previous year.

## 3. Features and Metafeatures

jAudio is able to extract many basic features [1]. These features may be one-dimensional (e.g., RMS), or may consist of multi-dimensional vectors (e.g., MFCC's).

Metafeatures are feature templates that automatically produce new features from existing features. These new features function just like normal features—producing output on a per-window basis. Metafeatures can also be chained together. jAudio provides three basic metafeature classes (Mean, Standard Deviation, and Derivative), which are also combined to produce two more metafeatures (Derivative of the Mean and Derivative of the Standard Deviation). These five metafeatures can be used to provide up to five additional features for each standard feature. An example of this in practice: the feature Root Mean Square (RMS) can be expanded from one to six features—RMS, Derivative of RMS, Running Mean of RMS, Running Standard Deviation of RMS, Derivative of the Running Mean of RMS, and Derivative of the Standard Deviation of RMS.

## 4. Aggregators

Aggregators are functions that collapse a sequence of vectors into a single vector or into a smaller sequence of vectors. This is similar to aggregate features introduced by Begrestra et. al. [3]. Compared to metafeatures, aggregators produce output only on a per-song basis, not on a per-window basis. Previously, jAudio provided only two fixed aggregators—Mean and Standard Deviation. Both of these aggregators take as input a sequence of vectors—the extracted windowed feature values over the entire input file—and output a single vector in its place. In essence, information on how a feature varies with time is collapsed into a single feature vector for each input file in its entirety. In the case of Standard Deviation, for example, each dimension of the output vector is the standard deviation of the values for that dimension in the input vector. It is also possible to design aggregators that

encapsulate more sophisticated and detailed information on how a feature varies from window to window.

Aggregators come in two varieties. The first is a function that is applied to the output of every feature to be saved. Mean and Standard Deviation are examples of this. The other type is a targeted aggregator that is applied only to one or more specific features. An example of this kind of aggregator is the AreaMoments Aggregator that, for a given order of input features, treats their combined sequence of vectors as a two-dimensional image matrix and calculates two-dimensional moments from this matrix. This second type of aggregator is useful for acquiring information about how a collection of features change together over time, a potentially very meaningful type of information that can not generally be accessed with alternative feature extractors.

Users of jAudio can create their own aggregators of either type. Currently, jAudio has three aggregators that are applied to the output of every feature: Mean, Standard Deviation, and MFCC. There are also two features that are applied selectively: Multiple Feature Histogram and Area Moments.

## 5. Dependency Resolution

While dependency resolution in jAudio has been briefly mentioned previously [1], the algorithm used was not described in any detail. Dependency resolution has two distinct steps: determining which features to extract and determining the order in which they should be extracted.

jAudio begins by making a list of all features whose output is to be saved (A). Another list (B) will be built that will eventually contain a list of all features to be extracted. These two lists can differ, as a user may not wish to save a given feature, but this feature may nonetheless need to be extracted in order to calculate another feature that does in fact need to be saved. An advantage of jAudio is that it hides from users the details of such hidden features as well as the order in which dependent features need to be extracted. Users need only enter the features that they actually want saved without considering any of these implementation details.

Initially B is set to be identical to A. jAudio then loops through each feature in B and adds each feature's dependencies to B if they are not already in B. The loop terminates when jAudio completes an iteration without adding any new features.

Once the features to be extracted have been identified, jAudio orders these features to ensure that every feature is calculated only after its dependencies have been calculated. To accomplish this, jAudio creates an ordered list of features to extract (C). It then cycles through all the features in B. If all dependencies of a given feature are in C, then it is added to C as well. This loop terminates once all features in B have been added to C.

## 6. Adding Features

jAudio is designed so that it can be used either as a standalone application or as class libraries that can be embedded in other applications. Special emphasis has been placed on making it easy to add new features and aggregators to jAudio. In order to do so, one need only add a reference to the new class to an XML configuration file. There is no need to recompile jAudio after doing this.

In order to further accommodate adding features at runtime, jAudio now includes a plugin folder that is automatically searched for compiled Java classes specified in the configuration file. The location of the plugin folder is specified in the first line of the configuration file as a URL. By using an URL instead of a path, it is possible to use a web site or other remote location to acquire the class files needed to construct features and aggregators. By modifying the configuration file and adding corresponding code to the plugin folder, it is possible to add new features at runtime. In addition, features already located in the classpath are found automatically.

## 7. Conclusions and Future Work

jAudio has continued to evolve over the past year, and new capabilities have been added to make it a more complete and easier to use feature extraction system. However, there remains more work to be done. The most pressing concern is that jAudio needs to be more efficient in its memory usage in general. In addition, jAudio would benefit from more features, especially higher-level features such as key identification and beat detection.

## References


[1] D. McEnnis, C. McKay, I. Fujinaga, and P. Depalle, "jAudio: A feature extraction library," in *Proceedings of the International Conference on Music Information Retrieval,* 2005, pp. 600–603.

[2] D. McEnnis, C. McKay, and I. Fujinaga, "Overview of OMEN," in *Proceedings of the International Conference on Music Information Retrieval*, 2006 (in press).

[3] S. Bray and G. Tzanetakis. "Distributed audio feature extraction for music," in *Proceedings of the International Conference on Music Information Retrieval,* 2005, pp. 434–437.

[4] J. Bergstra, N. Casagrande, D. Erhan, D. Eck, and B. Kegl, "Aggregate features and AdaBoost for music classification," in *Machine Learning*, 2006 (in press).

[5] X. Amatriain and A. Pau, "Developing cross-platform audio and music applications with the CLAM framework," in *Proceedings of the International Computer Music Conference*, 2005, pp. 403–410.