Sonic Annotator

Software by: *Chris Cannam, Mark Levy and Chris Sutton.*

This document by: *Ian Knopke and Chris Cannam.*

What is Sonic Annotator?

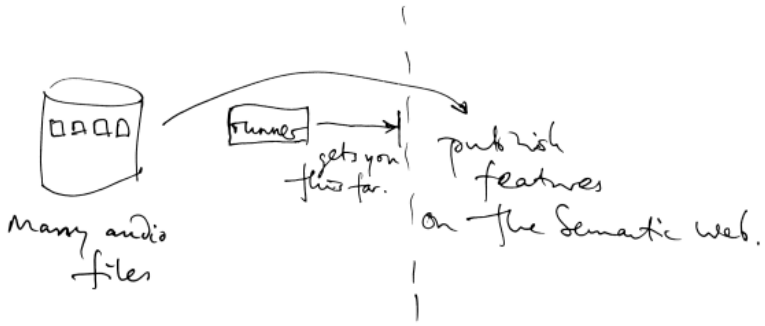
Sonic Annotator (formerly known as "Runner") is a command line program for batch extraction of audio features from multiple audio files. The basic idea is to abstract out the process of doing the feature extraction from the extraction methods, by using Vamp plugins for feature extraction. The output formats are RDF, CSV, and others; new output formats can be added with a modest amount of programming work.

Sonic Annotator is conceived of as having two main uses within the OMRAS2 project. First, it is useful for creating publishable feature data within the semantic web using the Music Ontology and Audio Features Ontology. Secondly, it can be used to created feature lists for use with AudioDb.

Though the official name of this program is Sonic Annotator, the working name during development was Runner, and the command-line examples in this document reflect the working name in some places. These may change in future.

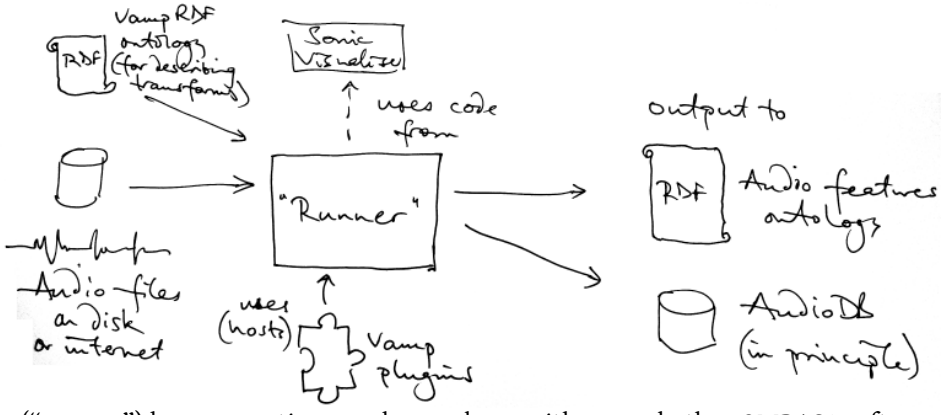
What could I use it for?

The most obvious intended use of Sonic Annotator (or "runner", as it is labelled in the diagram to the right) is for publishing features data about an audio collection – given a set of audio recordings, probably stored in a single place at the moment, and a set of feature specifications to automatically extract and publish for use by third parties.



Typical features might include tempo and key estimations, beat locations, segmentations, and so on – the set of available features does not depend on Sonic Annotator itself, but on the field of available Vamp plugins.

How does it fit in with other software?



Sonic Annotator ("runner") has connections as shown above with several other OMRAS2 software projects: the Music Ontology and Audio Features Ontology, Vamp audio feature extraction plugins, Sonic Visualiser, and in principle also AudioDb.

How complete is it?

Sonic Annotator is within striking distance of version 1.0. It's usable, with most of the desired features present. There are some details missing from the output format, and it also need extensive testing.

Where can I get it?

Sonic Annotator is GPL software. It is available on the web in the Sonic Visualiser SVN repository at.

> https://sv1.svn.sourceforge.net/svnroot/sv1/runner/trunk

You could also ask the author(s).

What language is it written in?

Sonic Annotator is written entirely in C++, and relies on some of Sonic Visualiser's code, also written in C++.

What input and output files are necessary to use Sonic Annotator?

Sonic Annotator requires several different things to work.

1) First of all, it needs a collection of audio files. Typically these will be on the local disk, often within the same file system directory, although Sonic Annotator can also retrieve and process files from the internet given a list of URLs.
2) While not strictly necessary, each Vamp plugin should have an RDF file supplied with it, which contains metadata that Sonic Annotator can use to enhance the descriptive data it outputs. The user should not have to concern themselves with this, but it may be a concern if you write your own Vamp plugins.
3) An input file is also required that explains the necessary features to be extracted, and the settings that each extractor requires. This can be in RDF format, or in XML. See *A Usage Example* below for more information.
4) Output may be in CSV, RDF or certain other formats. The user can specify whether output is written to a single big file, to one output file for each input audio file, or to one output for each transform and input.

What additional software does Sonic Annotator require?

Vamp plugins are needed to do any kind of extraction from audio files. Sonic Annotator also relies on a certain amount of Sonic Visualiser's code, so you'll need both if you plan on building it from source.

What is RDF?

RDF, or Resource Description Format, is a simple way of expressing the relationship between two pieces of information. It is becoming an important element in the construction of web-based ontologies, especially with regards to the semantic web.

In specific terms, RDF information is expressed as a set, or triple, of three pieces of information. Two objects are presented, along with the relationship between them. Usually the first item has something of a subject or ownership role. For example, in the following sentence:

```
John owns a bicycle
```

Conceptually, John and bicycle are objects, and the relationship between them is one of ownership. Usually, however, they look more like this:

```
<file:///share/music/testfile.mp3> a mo:AudioFile
```

In this case, the first part expresses the URI of an MP3 file. The second part, “a” is a common RDF term that

basically means “is of type”. “mo:AudioFile” identifies the URI as being of type “AudioFile” as defined in the music ontology (“mo”). For more information on the music ontology, see http://musicontology.com/.

Sonic Annotator uses RDF in three ways:

1. As a description of the vamp plug-in being used.
2. To describe the requested audio features to be extracted
3. As a description of the audio output format, as determined by the audio features ontology.

RDF is not strictly necessary for Sonic Annotator. For instance, it's possible to have input as XML and output as CSV, and not need RDF at all. However, RDF is the preferred format for working with ontologies and the semantic web.

A Usage Example

Here's an example of the RDF that might be produced by Sonic Annotator. In this case, the RDF uses the Audio Features ontology to describe a series of onset positions extracted using a particular Vamp percussion onset detector plugin.

The command line that produced this particular example was

```
$ runner -t percussiononsets.n3 -w rdf /share/music/testfile.mp3
```

Here, the file percussiononsets.n3 is an RDF/TTL file that specifies the use of a Vamp plugin for detecting percussive onsets, with certain parameter settings, as follows.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix vamp: <http://purl.org/ontology/vamp/>.
@prefix examples: <http://vamp-plugins.org/rdf/plugins/vamp-example-plugins#>.
@prefix : <#>.

:transform0 a vamp:Transform;
       vamp:plugin examples:percussiononsets;
       vamp:output examples:percussiononsets_output_onsets;
       vamp:step_size "256";
       vamp:block_size "512";
       vamp:sample_rate "0";      # This just means "use the audio file rate"
       vamp:window_type "Hanning";
       vamp:parameter_binding :param0 ;
       vamp:parameter_binding :param1 .

:param0 a vamp:ParameterBinding;
       vamp:parameter [ vamp:identifier "sensitivity" ];
       vamp:value "40".

:param1 a vamp:ParameterBinding;
       vamp:parameter examples:percussiononsets_param_threshold;
       vamp:value "2".
```

And this is the start of the output produced by Sonic Annotator:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix mo: <http://purl.org/ontology/mo/> .
@prefix af: <http://purl.org/ontology/af/> .
@prefix event: <http://purl.org/NET/c4dm/event.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

```
@prefix tl: <http://purl.org/NET/c4dm/timeline.owl#> .
@prefix : <#> .

<file:///share/music/testfile.mp3> a mo:AudioFile .

:signal_0 a mo:Signal ;
    mo:available_as <file:///share/music/testfile.mp3> ;
    mo:time [
        a tl:Interval ;
        tl:onTimeLine :tl_0
    ] .

:event_2 a af:Onset;
    event:time [
        a tl:Instant ;
        tl:onTimeLine :tl_0 ;
        tl:at "PT0.21S"^^xsd:duration ;
    ] .
:event_3 a af:Onset;
    event:time [
        a tl:Instant ;
        tl:onTimeLine :tl_0 ;
        tl:at "PT0.46S"^^xsd:duration ;
    ] .
:event_4 a af:Onset;
    event:time [
        a tl:Instant ;
        tl:onTimeLine :tl_0 ;
        tl:at "PT0.67S"^^xsd:duration ;
    ] .

# (etc)
```

This example is extremely rich in information – there is a lot of information about what exactly Sonic Annotator is describing: each output is an onset, which the audio feature ontology defines as a type of event, and we are specifying a time for each on a particular timeline and associating that with a particular audio signal.

However, Sonic Annotator can also write the output in CSV (comma-separated values) format. In this format, the above example is extremely simple, because each event only has a single value (its time):

```
0.21
0.46
0.67
(etc)
```

Command Line Options

Sonic Annotator is called from the command line as follows:

```
Usage: runner [-mr] -t trans.xml [...] -w <writer> [...] <audio> [...]
       runner [-mr] -T trans.txt [...] -w <writer> [...] <audio> [...]
       runner -s <transform>
       runner [-lh]
```

Where <audio> is an audio file or URL to use as input: either a local file path, local "file://" URL, or remote "http://" or "ftp://" URL.

The following audio file extensions are recognised:

aiff, au, avr, caf, flac, htk, iff, mat, mp3, ogg, paf, pvf, raw, sd2, sds, sf, voc, w64, wav, xi.

Playlist files in M3U format are also supported.

Various different transformation options are also available:

-t, --transform <T> Apply transform described in transform XML file <T> to all input audio files. You may supply this option multiple times. You must supply this option or -T at least once for any work to be done.

-T, --transforms <T> Apply all transforms described in transform XML files whose names are listed in text file <T>. You may supply this option multiple times.

-w, --writer <W> Write output using writer type <W>. Supported writer types are: audiodb, csv, default, rdf. You may supply this option multiple times. You must supply this option at least once for any work to be done.

-m, --multiplex If multiple input audio files are given, use mono mixdowns of all files as the input channels for a single invocation of each transform, instead of running the transform against all files separately.

-r, --recursive If any of the <audio> arguments is found to be a local directory, search the tree starting at that directory for all supported audio files and take all of those as input instead.

Housekeeping options:

-l, --list List all known transform ids to standard output.

-s, --skeleton <T> Generate a skeleton transform file for transform id <T> and write it to standard output.

-h, --help Show help.

If no **-w** (or **--writer**) options are supplied, either the **-l** or **-s** option (or long equivalent) must be given instead.

Additional options for writer type "audiodb":

--audiodb-catid <X> Catalogue ID

--audiodb-basedir <X> Base output directory path

Additional options for writer type "csv":

--csv-basedir <X> Base output directory path. (The default is the same directory as the input file.)

--csv-force If an output file already exists, overwrite it.

--csv-append If an output file already exists, append data to it.

--csv-separator <X> Column separator for output. Default is "," (comma).

Additional options for writer type "rdf":

--rdf-basedir <X> Base output directory path. (The default is the same directory as the input file.)

--rdf-many-files Create a separate output file for every combination of input file and transform. The output file names will be based on the input file names. (The default is to create one output file per input audio file, and write all transform results for that input into it.)

--rdf-one-file <X> Write all transform results for all input files into the single named output file.

--rdf-force If an output file already exists, overwrite it.

--rdf-append If an output file already exists, append data to it.

--rdf-plain Use "plain" RDF even if transform metadata is available.

--rdf-signal-uri <X> Signal URI to link to

Internal Architecture

Sonic Annotator is written in C++. It consists of a fairly small set of classes of its own, with a large set of

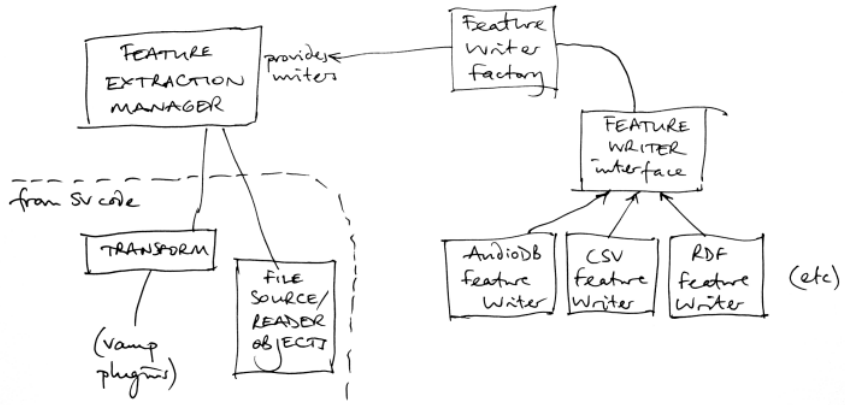
dependencies on libraries from Sonic Visualiser. These in turn depend on the Qt Core toolkit, and on many other audio file and RDF libraries.

Sonic Annotator class overview

Sonic Annotator uses library code from Sonic Visualiser (summarised in *Sonic Visualiser modules* below) for many of its subsystems, including retrieving and reading audio files, loading and configuring Vamp plugins through the "transform" abstraction, and parsing RDF documents of various kinds.

> **Transforms**
>
> A "transform" in Sonic Visualiser and Sonic Annotator conceptually consists of a plugin identifier together with the context in which it is executed.
>
> For example, consider the "percussion onset detector" plugin found in the Vamp example plugins set. This is a Vamp plugin, and it can be run against an audio file, but further information about how to run it is required if it is to produce the same results when used in more than one different program.
>
> The transform concept provides this information. It associates the plugin's identifier with its parameter values, the hop and frame sizes for plugin execution, sample rate at which the audio should be provided to it, and so on. All of this data is bundled together and treated as a single unit.

The most important classes that are actually contained in Sonic Annotator's own code base are the FeatureExtractionManager, and various FeatureWriter implementations.



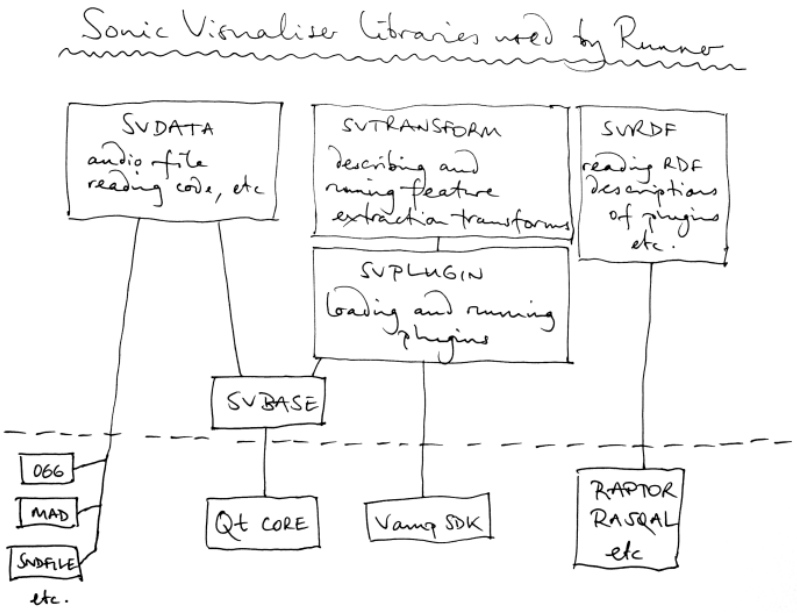
The `FeatureExtractionManager` class does most of the work in Sonic Annotator. It is first configured with a series of calls to `addFeatureExtractor` or `addFeatureExtractorFromFile`, which instruct it to load a transform description and associate that with a set of FeatureWriters (drawn from the available pool of `RDFFeatureWriter`, `CSVFeatureWriter` etc). Once the transforms and writers have been constructed, the main function makes a single call to extractFeatures for each audio file or URL to be used. This function reads the audio file, passing each block that is read to all of the transform plugins in turn, and sending the resulting output to the set of feature writers associated with that transform.

The `FeatureExtractionManager` is in principle capable of running any number of feature extraction transforms against any number of input audio files, and writing the results from each of them to a distinct set of different types of output writer. However, Sonic Annotator's main program only provides user capability for sending all transform outputs to the same set of writers.

Adding a new output format to Sonic Annotator is fairly straightforward: create a new subclass of `FeatureWriter`. There is also a subclass called `FileFeatureWriter` which may be used for implementations that write data to files; this provides options for writing all results to a single file, writing to multiple files (one per transform) and so on, and it handles opening output files and manages suitable stream objects using the Qt `QTextStream` API for use by the subclass implementation.

Sonic Visualiser modules

Sonic Annotator is built using a set of C++ libraries from the Sonic Visualiser code base. These are the *data*, *transform*, *plugin*, and *RDF* modules of Sonic Visualiser. Reusing this code not only simplifies the development of Sonic Annotator but also extends the effective test coverage for this code, improving code quality.



The *data* library contains an audio file reader abstraction which can decode mp3, Ogg, and various other formats using commonly available libraries for the low-level work (liboggz/libfishsound, libmad, libsndfile, and also QuickTime if built on OS/X). It also supports retrieving files from HTTP or FTP sources transparently – a file may be referred to using either local file path or a URL, and the library will work appropriately in either case.

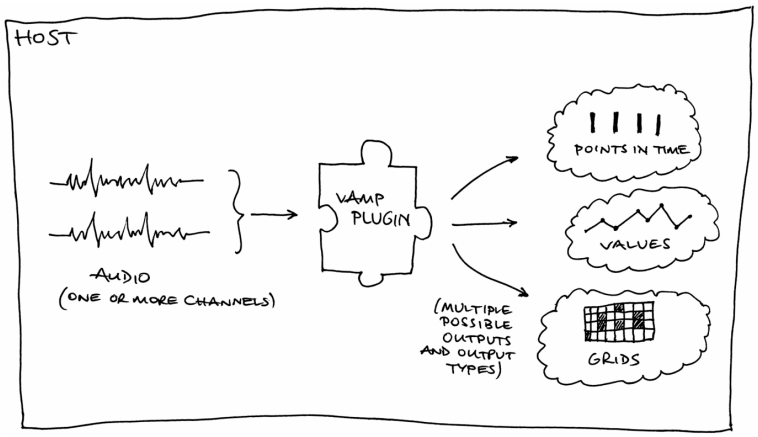
The *transform* library handles describing, loading and running feature extraction "transforms". A transform conceptually consists of a feature extraction plugin together with the context in which it is executed – the sample rate that the audio should be resampled to, hop and frame sizes for plugin execution, parameters for the plugin, and so on. The *transform* library relies on the *plugin* library, which handles low-level loading and running of feature extraction plugins.

Finally the *RDF* library provides classes for specific uses of RDF data, such as loading transform descriptions from RDF, loading metadata associated with a Vamp plugin from a Vamp ontology-based RDF file, and so on. This library uses the Rasqal and Raptor libraries from the Redland RDF suite.

NOTE: include hand-drawn diagram

A quick introduction to Vamp RDF

Below is an example of (part of) an RDF file that describes the percussion onset detector Vamp plugin using the Vamp plugin ontology.



This RDF includes some of the same information that can be obtained by loading and querying the plugin (so providing an "offline" means to get the same data), but it also contains some extra information about the meanings of the results (e.g. to explain that an output returns onsets, not just "points in time of some sort").

Sonic Annotator can load and use RDF such as this in order to enhance the RDF that it produces for feature outputs. However, it isn't essential for use of Sonic Annotator.

```
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix vamp:     <http://purl.org/ontology/vamp/> .
@prefix plugbase: <http://vamp-plugins.org/rdf/plugins/vamp-example-plugins#> .
@prefix owl:      <http://www.w3.org/2002/07/owl#> .
@prefix dc:       <http://purl.org/dc/elements/1.1/> .
@prefix af:       <http://purl.org/ontology/af/> .
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix cc:       <http://web.resource.org/cc/> .
@prefix :         <> .

<>  a   vamp:PluginDescription ;
    foaf:maker          <x> ;
    foaf:maker          <http://www.vamp-plugins.org/doap.rdf#template-generator> ;
    foaf:primaryTopic   <http://vamp-plugins.org/rdf/plugins/vamp-example-plugins> .

:vamp-example-plugins a  vamp:PluginLibrary ;
    vamp:identifier "vamp-example-plugins" ;
    vamp:available_plugin plugbase:percussiononsets .

plugbase:percussiononsets a   vamp:Plugin ;
    dc:title              "Simple Percussion Onset Detector" ;
    vamp:name             "Simple Percussion Onset Detector" ;
    dc:description        "Detect percussive note onsets from broadband energy rise" ;
    foaf:maker            [ foaf:name "Vamp SDK Example Plugins" ] ;
    vamp:identifier       "percussiononsets" ;
    vamp:vamp_API_version vamp:api_version_1 ;
    owl:versionInfo       "2" ;
    vamp:input_domain     vamp:FrequencyDomain ;
    vamp:parameter        plugbase:percussiononsets_param_threshold ;
    vamp:parameter        plugbase:percussiononsets_param_sensitivity ;
    vamp:output           plugbase:percussiononsets_output_onsets ;
    .
plugbase:percussiononsets_param_threshold a  vamp:Parameter ;
    vamp:identifier     "threshold" ;
    dc:title            "Energy rise threshold" ;
    dc:format           "dB" ;
    vamp:min_value       0 ;
    vamp:max_value       20 ;
    vamp:unit           "dB"  ;
    vamp:default_value   3 ;
    vamp:value_names     ();
    .
plugbase:percussiononsets_param_sensitivity a  vamp:Parameter ;
    vamp:identifier     "sensitivity" ;
    dc:title            "Sensitivity" ;
    dc:format           "%" ;
    vamp:min_value       0 ;
    vamp:max_value       100 ;
    vamp:unit           "%"  ;
    vamp:default_value   40 ;
    vamp:value_names     ();
    .
plugbase:percussiononsets_output_onsets a  vamp:SparseOutput ;
    vamp:identifier       "onsets" ;
    dc:title              "Onsets" ;
    dc:description        "Percussive note onset locations"  ;
    vamp:fixed_bin_count  "true" ;
    vamp:unit             "" ;
    vamp:bin_count        0 ;
    vamp:bin_names        ();
    vamp:sample_type      vamp:VariableSampleRate ;
    vamp:sample_rate      44100 ;
     vamp:computes_feature_type  af:Onset
```

Glossary

AudioDb – An audio database project, for doing fast similarity searches of audio files.

CSV - Comma-separated value file, where items on a line are separated by commas.

OMRAS2 – UK project for music research taking place at Queen Marys University and Goldsmiths, University of London.

Ontology – A Hierarchical arrangement of information.

Sonic Visualiser – A program for visualising information extracted from audio files.

Links

Sonic Visualiser http://www.sonicvisualiser.org/

Vamp Plugins http://www.vamp-plugins.org/

OMRAS2 http://omras2.org/

Intelligent Sound & Music Systems, Goldsmiths, University of London
http://doc.gold.ac.uk/isms/

Centre for Digital Music, Queen Mary, University of London
http://www.elec.qmul.ac.uk/digitalmusic/

AudioDb http://omras2.doc.gold.ac.uk/software/audiodb/

RDF http://en.wikipedia.org/wiki/Resource_Description_Framework

Semantic Web http://www.w3.org/2001/sw/