Ecasound Control Interface Guide

Kai Vehmanen, Brad Bowman, Tony Leake, Jan Weil, Mario Lang

03062006

Contents

1 Introduction **3**

2 Document history **3**

3 General **4**
3.1 What's it good for? . . . 4
3.2 Services and behaviour . . . 4
3.2.1 Actions . . . 4
3.2.2 Return values . . . 4
3.2.3 Errors . . . 5
3.2.4 Other . . . 5
3.3 Porting to new environments . . . 5

4 Implementations **5**
4.1 General . . . 5
4.1.1 Overview . . . 5
4.1.2 Usage . . . 5
4.1.3 Example . . . 6
4.2 Notes Concerning Standalone ECI Implementations . . . 6
4.3 C++ . . . 6
4.3.1 Overview . . . 6
4.3.2 Usage . . . 6
4.3.3 Example . . . 7
4.4 C . . . 8
4.4.1 Overview . . . 8
4.4.2 Usage . . . 8
4.4.3 Example . . . 8
4.5 Emacs . . . 9
4.5.1 Overview . . . 9
4.5.2 Usage . . . 10
4.5.3 Example . . . 10
4.6 Python . . . 11
4.6.1 Overview . . . 11

4.6.2 Usage . . . 11
4.6.3 Example . . . 11
4.7 Perl . . . 12
4.7.1 Overview . . . 12
4.7.2 Usage . . . 12
4.7.3 Example . . . 12
4.8 PHP . . . 13
4.8.1 Overview . . . 13
4.8.2 Usage . . . 13
4.8.3 Example . . . 13
4.9 Ruby . . . 15
4.9.1 Overview . . . 15
4.9.2 Usage . . . 15
4.9.3 Example . . . 15

5 Application development **16**
5.1 Tips for debugging . . . 16

1 Introduction

The idea behind the Ecasound Control Interface (ECI) is to take a subset of functionality provided by libecasound, provide a simple API for it, and port it to various languages. At the moment, implementations of the ECI API are available for C, C++, elisp, Python and Ruby. These all come by default with the Ecasound package. Additional implementations, for example for Perl and PHP, are distributed independently.

ECI is heavily based on Ecasound's interactive mode (EIAM), and the services it provides. See `ecasound-iam(1) manual page` for a detailed EIAM documentation.

2 Document history

- 04.06.2006 - Added more information to the "Tips for debugging" section.
- 21.03.2005 - Updated the "Tips for debugging" section.
- 25.09.2004 - Updated the "Return values" section based on feedback from Adam Linson.
- 02.05.2004 - "Emacs" section added (written by Mario Lang).
- 28.11.2003 - "Ruby" section added (written by Jan Weil). Updated the introduction.
- 26.11.2003 - Fix filename for the alternative Python API (eci.py).
- 18.11.2003 - Typo fixes. Updated documentation to reflect the new naming convention (ecasound refers to the binary, Ecasound refers to the whole package).
- 26.10.2002 - Changed the C++ linking example.
- 24.10.2002 - Added "Notes Concerning Standalone ECI Implementations" section. Added compilation examples.
- 06.10.2002 - Added "Application development" section.
- 05.10.2002 - Changed the libecasoundc link path.
- 29.09.2002 - "PHP" section added (written by Tony Leake).
- 25.04.2002 - Changed headers path from "<ecasoundc/file.h>" to "<file.h>" and added library version number to link instructions.
- 21.10.2001 - Added this history section. Minor changes to ECI examples.

3 General

ECI doesn't provide any routines that directly manipulate audio or Ecasound objects. What it does provide is an easy and generic way to issue EIAM (Ecasound Inter-Active Mode) commands, access the command return-values and perform error handling.

This approach has two benefits. First, it is possible to keep the API small, and thus make it easier to port ECI to new languages. Secondly, it's possible to keep ECI relatively stable. Ecasound itself is a large, developing library. New features are added all the time, and from time to time, older parts of the library will get rewritten to better suit new uses. Now for application developers wanting to take advantage of libecasound, these constant changes are very annoying, especially if your specific app doesn't need the latest new features. In these cases, ECI is the best platform for application development.

3.1 What's it good for?

Specific tasks ECI is aimed at:

- 1. automating (scripting in its traditional sense)
- 2. frontends (generic / specialized)
- 3. sound services to other apps

3.2 Services and behaviour

Here is a list of services provided by all ECI implementations:

3.2.1 Actions

command(string) Issue an EIAM command.

command_float_arg(string, float) Issue an EIAM command. This function can be used instead of *command(string)*, if the command in question requires exactly one numerical parameter. This way it's possible to avoid the extra string -> float conversion, which would lead to lost precision.

3.2.2 Return values

Each EIAM command has exactly one return value type. After a command has been issued, only one last_type() functions returns a non-empty value. For example, last_float() only returns a valid value if *last_type() == 'f'* holds true. Not all EIAM commands return a value (return type is void).

last_string() Returns the last string return value.

last_string_list() Returns the last collection of strings (one or more strings).

last_float() Returns the last floating-point return value. Note! last_float() doesn't refer to the C/C++ type 'float'. In most implementations, floats are 64bit values (doubles in C/C++).

last_integer() Returns the last integer return value. This function is also used to return boolean values, where non-zero means 'true' and zero 'false'.

last_long_integer() Returns the last long integer return value. Long integers are used to pass values like 'length_in_samples' and 'length_in_bytes'. It's implementation specific whether there's any real difference between integers and long integers.

3.2.3 Errors

error() Returns true (!= 0) if error has occured during the execution of last EIAM command. Otherwise returns false (= 0).

last_error() Returns a string describing the last error. If the last EIAM command was executed succesfully, last_error() returns an empty string.

3.2.4 Other

initialize() Reserve resources.

cleanup() Free all reserved resources.

3.3 Porting to new environments

Porting ECI to new languages should be easy. All there is to do is to implement the services listed in the previous section to the target language. In most cases it's to easiest to use the C++ or C ECI as the underlying implementation to build upon.

4 Implementations

4.1 General

4.1.1 Overview

This section contains overview of how ECI is implemented in the discussed language (eg. as a single class, set of classes, set of routines, etc).

4.1.2 Usage

A quick tutorial to get you started.

4.1.3 Example

Implementation of the following:

1. Setup ECI to read audio from file, apply a 100Hz lowpass filter, and send it to the soundcard (/dev/dsp).
2. Every second, check the current position. If the stream has been running for over 15 seconds, exit immediately. Also, every second, increase the lowpass filter's cutoff frequency by 500Hz.
3. Stop the stream (if not already finished) and disconnect the chainsetup. Print chain operator status info.

4.2 Notes Concerning Standalone ECI Implementations

The C implementation of ECI is not directly linked against the main Ecasound libraries. Instead, the ecasound executable is launched on the background and command pipes are used to communicate with it.

The launched ecasound executable can be selected by using the *ECASOUND* environment variable. If it is not defined, the C ECI implementation will try to launch "ecasound" (ie. has to be somewhere in PATH).

In addition to the C implementation, this also affects all ECI implementations that are based on the C version. Currently this includes at least the Perl, PHP and Python ECI modules.

4.3 C++

4.3.1 Overview

C++ implementation is based around the ECA_CONTROL_INTERFACE class. STL vector is used for representing collections of objects (last_string_list()).

4.3.2 Usage

1. #include <eca-control-interface.h>
2. create an instance of the ECA_CONTROL_INTERFACE class and use its member functions
3. link you app agains libecasoundc (-lecasoundc)
4. compilation example: *c++ -o ecidoc_example ecidoc_example.cpp 'libecasoundc-config –cflags –libs'*

4.3.3 Example

```
#include <iostream>
#include <unistd.h>
#include <eca-control-interface.h>

int main(int argc, char *argv[])
{
  double cutoff_inc = 500.0;

  ECA_CONTROL_INTERFACE e;
  e.command("cs-add play_chainsetup");
  e.command("c-add 1st_chain");
  e.command("ai-add some_file.wav");
  e.command("ao-add /dev/dsp");
  e.command("cop-add -efl:100");
  e.command("cop-select 1");
  e.command("copp-select 1");
  e.command("cs-connect");
  e.command("start");
  while(1) {
    sleep(1);
    e.command("engine-status");
    if (e.last_string() != "running") break;
    e.command("get-position");
    double curpos = e.last_float();
    if (curpos > 15.0) break;
    e.command("copp-get");
    double next_cutoff = cutoff_inc + e.last_float();
    e.command_float_arg("copp-set", next_cutoff);
  }

  e.command("stop");
  e.command("cs-disconnect");
  e.command("cop-status");
  cerr << "Chain operator status: " << e.last_string() << endl;

  return(0);
}
```

4.4 C

4.4.1 Overview

All C ECI functions are prefixed with "eci_". When returning string values, a const pointer to a null-terminated char array (const char*) is returned. It's important to keep in mind that these are "borrowed" references. If you need to later use the data, you must copy it to application's own buffers.

Returning a list of strings is implemented using two functions: *eci_last_string_list_count()* returns the number of strings available, and *eci_last_string_list_item(int n)* returns a pointer (const char*) to the string at index *n*.

Note! As of Ecasound 2.0.1, the C ECI implementation also provides reentrant access to the ECI API. These alternative routines are marked with '_r' postfix.

4.4.2 Usage

1. #include <ecasoundc.h>
2. use the eci_* routines
3. link your app against libecasoundc (-lecasoundc)
4. compilation example: *gcc -o ecidoc_example ecidoc_example.c 'libecasoundc-config –cflags –libs'*

4.4.3 Example

```
#include <stdio.h>
#include <unistd.h>
#include <ecasoundc.h>

int main(int argc, char *argv[])
{
  double cutoff_inc = 500.0;

  eci_init();
  eci_command("cs-add play_chainsetup");
  eci_command("c-add 1st_chain");
  eci_command("ai-add some_file.wav");
  eci_command("ao-add /dev/dsp");
  eci_command("cop-add -efl:100");
  eci_command("cop-select 1");
  eci_command("copp-select 1");
  eci_command("cs-connect");
  eci_command("start");
```

```
  while(1) {
    double curpos, next_cutoff;

    sleep(1);
    eci_command("engine-status");
    if (strcmp(eci_last_string(), "running") != 0) break;
    eci_command("get-position");
    curpos = eci_last_float();
    if (curpos > 15.0) break;
    eci_command("copp-get");
    next_cutoff = cutoff_inc + eci_last_float();
    eci_command_float_arg("copp-set", next_cutoff);
  }

  eci_command("stop");
  eci_command("cs-disconnect");
  eci_command("cop-status");
  printf("Chain operator status: %s", eci_last_string());
  eci_cleanup();

  return(0);
}
```

4.5 Emacs

4.5.1 Overview

The Ecasound package comes with an 'Ecasound' library for Emacs included. ecasound.el is a implementation of the ECI API for Emacs, as well as an interactive interface to Ecasound sessions implemented on top of that. Simply use "M-x ecasound RET" to fire up an interactive Ecasound session.

All Emacs Lisp ECI functions are prefixed with "eci". 'ecasound.el' is implemented in a high level manner which means that you won't find most of the commands known from libecasoundc like last_string, last_float, etc. Instead of that every call to function "eci-command", which accepts all the well known IAM commands, returns ecasound's response in an appropriate type automatically. If an error occurs, e. g. there's a typo in a command or a file is not found, the function returns "nil". In all other cases, either an automatically converted Lisp value is returned, or "t" in the case where there was no particular value returned.

Additionally, most of the available IAM commands have their own Emacs Lisp function including documentation and possibly a parameter list. All these functions are interactive, so you can use them in ecasound-iam-mode simply by invoking them via M-x or by pressing an assigned key combination. Emacs will prompt you for the required parameters, providing completion wherever

possible.

As a convention, "eci-command" and its variants do take a buffer or process as an optional last argument. If this is "nil", the current buffer is assumed to be the ecasound session refered to by this call. This makes it possible to use several ECI sessions concurrently, dispatching on the buffer or process in use.

4.5.2 Usage

1. make ecasound.el available in your "load-path"
2. (require 'ecasound)
3. create a buffer with an associated Ecasound session ("eci-init")
4. use "eci" functions with the new buffer

4.5.3 Example

```
(require 'ecasound)

(defun example (file &optional cutoff-increment session)
  (unless cutoff-increment (setq cutoff-increment 500.0))
  (with-current-buffer (or session (eci-init))
    (eci-cs-add "play_chainsetup")
    (eci-c-add "1st_chain")
    (eci-ai-add file)
    (eci-ao-add "/dev/dsp")
    (eci-cop-add "-efl:100")
    (eci-cop-select 1)
    (eci-copp-select 1)
    (eci-cs-connect)
    (eci-start)
    (sit-for 1)
    (while (and (string= (eci-engine-status) "running")
                (<= (eci-get-position) 15))
      (eci-copp-set (+ cutoff-increment (eci-copp-get)))
      (sit-for 1))
    (eci-command "stop")
    (when (eci-cs-disconnect)
      (destructuring-bind
  ((cop n1 (copp n2 val)))
  (cdr (assoc "1st_chain" (eci-cop-status)))
(message "%s %s is now %f" cop copp val)))))
```

NOTE: function "eci-cop-status" is actually a very high level function which already converts the returned information to a nested list structure.

For more complex examples of the Emacs Lisp ECI implementation, see function “eci-example”, “ecasound-normalize” and “ecasound-signalview” in ecasound.el.

4.6 Python

4.6.1 Overview

Python implementation is based around the ECA_CONTROL_INTERFACE class. Lists are used for representing collections of objects.

Note! Eric S. Tiedemann has written an alternative Python interface to ECI. You'll find this interface included in the main Ecasound packege, in “pyecasound/eci.py”. To use this instead of the standard interface, just 'import eci' and you're set! :)

4.6.2 Usage

1. import pyeca
2. create an instance of the ECA_CONTROL_INTERFACE class and use its member functions
3. python 'yourapp.py' and that's it :)

4.6.3 Example

```
#!/usr/local/bin/python
import time
from pyeca import *
e = ECA_CONTROL_INTERFACE()
e.command("cs-add play_chainsetup")
e.command("c-add 1st_chain")
e.command("ai-add some_file.wav")
e.command("ao-add /dev/dsp")
e.command("cop-add -efl:100")
e.command("cop-select 1")
e.command("copp-select 1")
e.command("cs-connect")
e.command("start")
cutoff_inc = 500.0
while 1:
    time.sleep(1)
    e.command("engine-status")
    if e.last_string() != "running": break
    e.command("get-position")
    curpos = e.last_float()
    if curpos > 15: break
```

```
    e.command("copp-get")
    next_cutoff = cutoff_inc + e.last_float()
    e.command_float_arg("copp-set", next_cutoff)
e.command("stop")
e.command("cs-disconnect")
e.command("cop-status")
print "Chain operator status: ", e.last_string()
```

4.7 Perl

4.7.1 Overview

Audio::Ecasound provides perl bindings to the Ecasound control interface of the Ecasound program. You can use perl to automate or interact with Ecasound so you don't have to turn you back on the adoring masses packed into Wembly Stadium.

Audio::Ecasound was written by Brad Bowman. At the moment this module is not distributed with Ecasound. To get the latest version, check the following `CPAN link`.

4.7.2 Usage

See the below example. For more info, here's another `CPAN link`.

4.7.3 Example

use Audio::Ecasound qw(:simple);

```
eci("cs-add play_chainsetup");
eci("c-add 1st_chain");
eci("ai-add some_file.wav");
eci("ao-add /dev/dsp");
# multiple \n separated commands
eci("cop-add -efl:100
     # with comments
     cop-select 1
     copp-select 1
     cs-connect");
eci("start");
my $cutoff_inc = 500.0;
while (1) {
    sleep(1);
    last if eci("engine-status") ne "running";
    my $curpos = eci("get-position");
    last if $curpos > 15;
```

```
    my $next_cutoff = $cutoff_inc + eci("copp-get");
    # Optional float argument
    eci("copp-set", $next_cutoff);
}
eci("stop");
eci("cs-disconnect");
print "Chain operator status: ", eci("cop-status");
```

4.8 PHP

4.8.1 Overview

This PHP extension provides bindings to the Ecasound control interface. It is useful both for scripting Ecasound and for writing graphical audio applications with PHP Gtk.

The PHP Ecasound extension was written by Tony Leake. At the moment this module is not distributed with Ecasound. The latest version and example scripts, are available from `http://www.webwise-data.co.uk/php_audio/php_audio_extension.html`.

4.8.2 Usage

1. Obtain and build the Ecasound PHP extension
2. Initialise Ecasound, eci_int();
3. Issue EAM commands eg, eci_command("cs-add my_chain_setup");
4. Free resources, eci_cleanup();

4.8.3 Example

```
Implementation of the following:

1. Setup ECI to read audio from file, apply a 100Hz lowpass filter,
and send it to the soundcard (/dev/dsp).
2. Every second, check the current position. If the stream has been
running for over 15 seconds, exit immediately. Also, every second,
increase the lowpass filter's cutoff frequency by 500Hz.
3. Stop the stream (if not already finished) and disconnect the chainsetup.
Print chain operator status info

<?php

$cutoff_inc = 500.0;
$curpos=0;
$next_cutoff=0;
```

```
eci_init();
eci_command("cs-add play_chainsetup");
eci_command("c-add 1st_chain");
eci_command("ai-add /tmp/somefile.wav");
eci_command("ao-add /dev/dsp");
eci_command("cop-add -efl:10");
eci_command("cop-select 1");
eci_command("copp-select 1");
eci_command("cs-connect");
eci_command("start");

while(1) {

    sleep(1);

    eci_command("engine-status");
    if (eci_last_string() !="running"){
        break;
    }

    eci_command("get-position");
    $curpos = eci_last_float();
    if ($curpos > 15.0){
        break;
    }

    eci_command("copp-get");
    $next_cutoff = $cutoff_inc + eci_last_float();
    eci_command_float_arg("copp-set",$next_cutoff);
}

eci_command("stop");
eci_command("cs-disconnect");
eci_command("cop-status");

printf("Chain operator status: %s", eci_last_string());

eci_cleanup();
?>
```

4.9 Ruby

4.9.1 Overview

The Ecasound package comes with an 'Ecasound' module for Ruby included. If ruby is detected during the installation process it is installed automatically (assuming you are installing ecasound from source code). The module contains the class definition of a native ecasound control interface called "ControlInterface".

'Ecasound::ControlInterface' is implemented in a high level manner which means that you won't find most of the commands known from libecasoundc like last_string, last_float, etc. Instead of that every call to the instance method "command", which accepts all the well known IAM commands, returns ecasound's response in an appropriate type automatically. If an error occurs, e. g. there's a typo in a command or a file is not found, an exception of type EcasoundError is raised.

4.9.2 Usage

1. require 'ecasound'
2. create an instance of Ecasound::ControlInterface
3. use it's command method to send IAM commands to ecasound
4. catch an EcasoundError if necessary

4.9.3 Example

```
#!/usr/bin/env ruby
require "ecasound"

SOME_FILE = "path/to/file.wav"

e = Ecasound::ControlInterface.new()
e.command("cs-add play_chainsetup")
e.command("c-add 1st_chain")
e.command("ai-add #{SOME_FILE}")
e.command("ao-add /dev/dsp")
e.command("cop-add -efl:100")
e.command("cop-select 1")
e.command("copp-select 1")
e.command("cs-connect")
e.command("start")

cutoff_inc = 500.0

loop do
    sleep(1)
```

```
    break if e.command("engine-status") != "running"
    break if e.command("get-position") > 15
    e.command("copp-set #{cutoff_inc + e.command('copp-get')}")
end

e.command("stop")
e.command("cs-disconnect")

$stdout << "Chain operator status: " + e.command("cop-status") + "\n"
```

5 Application development

5.1 Tips for debugging

Here's a few tips what to do if the ECI app you have developed is not working correctly.

1. Check your Ecasound installation. Try to run the "ecasound" console user-interface and verify that the basic functionality is working (ie. something like "ecasound -i foo.wav -o /dev/dsp".

2. If developing in C or C++, check that your application is correcly linked: "ldd /path/to/myapp". All the libraries should be properly found.

3. Check error conditions. You should remember to check for errors in your ECI apps using the eci_error() and eci_last_error() functions. Especially when intializing ECI for the first time and after important commands like "cs-connect", you should always check for errors.

4. Use the ECASOUND_LOGFILE environment variable to write all engine output to a separate logfile. See ecasound(1) manpage for details on how to use this mechanism. Requires Ecasound version 2.4.5 or newer.

5. Utilize the "int-log-history" ECI command added to version 2.4.0 of Ecasound. Recent messages from the engine can help to track down the problem. Before use, you need to first set the history length to a non-zero value with "int-set-log-history-length".

6. Launch Ecasound in interactive mode ("ecasound -c"), and issue the commands your ECI application is using, manually one-by-one and see what happens. If something goes wrong, increase Ecasound's debug level (for instance "-ddd") and re-run the test.