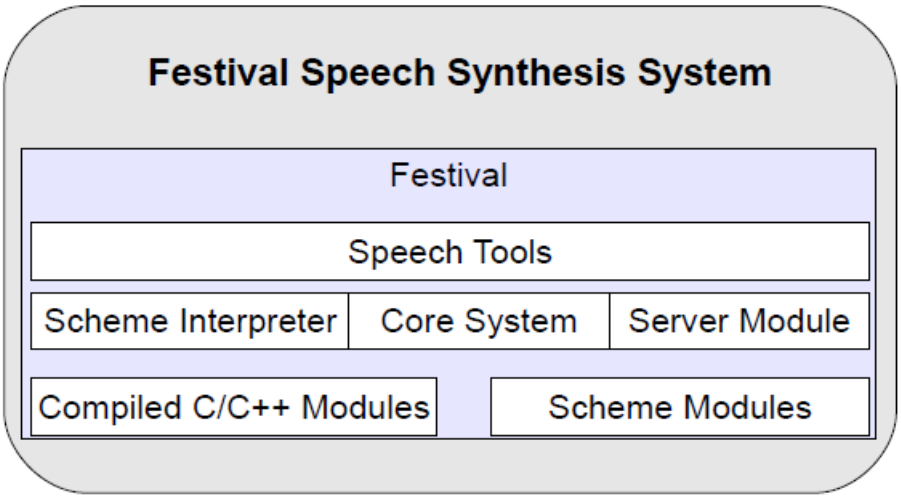
Festival Internals



Festivals core system essentially consists of a Scheme (a variant of LISP) interpreter implemented in compiled C++ code. On top of this system, various blocks are incorporated as ether compiled C/C++ code or as Scheme scripts. The Festival system makes extensive use of Edinburgh Speech Tools library, and many of its low level structures are built with it. The tools are another freely distributed GNU software package. Edinburgh Speech Tools contains C++ implementations of various signal processing methods and data classes for manipulating and accessing waveforms and linguistic structures.

Festival Basics Part I

Where is it ?

Internet

Site: http://www.cstr.ed.ac.uk/projects/festival/

Manual: http://www.cstr.ed.ac.uk/projects/festival/manual/

On the computer

```
$whereis festival
festival: /usr/bin/festival /usr/lib/festival /usr/include/festival
/usr/share/festival /usr/share/man/man1/festival.1.gz
```

Getting it started

You will see:

```
$festival
Festival Speech Synthesis System 1.96:beta July 2004
Copyright (C) University of Edinburgh, 1996-2004. All rights reserved.
For details type `(festival_warranty)'
festival>
```

Or:

```
Festival Speech Synthesis System 2.1:release November 2010
Copyright (C) University of Edinburgh, 1996-2010. All rights reserved.

clunits: Copyright (C) University of Edinburgh and CMU 1997-2010
hts_engine:
The HMM-based speech synthesis system (HTS)
hts_engine API version 1.04 (http://hts-engine.sourceforge.net/)
Copyright (C) 2001-2010  Nagoya Institute of Technology
              2001-2008  Tokyo Institute of Technology
All rights reserved.
For details type `(festival_warranty)'
festival>
```

Or something similar

Basic TTS

```
festival> (set! utt1 (Utterance Text "Hello world, or more specifically
SOFTENG 206"))
#<Utterance 1d08a0>
festival>
```

Although this creates an utterance it doesn't do anything else. To get a waveform you must synthesize it.

```
festival> (utt.synth utt1)
#<Utterance 1d08a0>
festival>
```

This calls various modules, including tokenizing, duration,. intonation etc. Which modules are called are defined with respect to the type of the utterance, in this case `Text`.

```
festival> (utt.play utt1)
#<Utterance 1d08a0>
festival>
```

will send the synthesized waveform to your audio device. You should hear "Hello world" from your machine.

To make this all easier a small function doing these three steps exists. `SayText` simply takes a string of text, synthesizes it and sends it to the audio device.

```
festival> (SayText "Good morning, welcome to Festival")
#<Utterance 1d8fd0>
festival>
```

A Scheme Aside

To work with Festival at most a small amount of Scheme is required.

Syntax: an expression is an *atom* or a *list*. A list consists of a left parenthesis, a number of expressions and right parenthesis. Atoms can be symbols, numbers, strings or other special types like functions, hash tables, arrays, etc.

Semantics: All expressions can be evaluated. Lists are evaluated as function calls. When evaluating a list all the members of the list are evaluated first then the first item (a function) is called with the remaining items in the list as arguments. Atoms are evaluated depending on their type: symbols are evaluated as variables returning their values. Numbers, strings, functions, etc. evaluate to themselves.

Comments are started by a semicolon and run until end of line.

Running Festival from the command line

Rather than starting the command interpreter, Festival may synthesize files specified on the command line

```
$ festival --tts myfile
$
```

Or

```
$echo hello world | festival --tts
$cat myfile | festival --tts
```

Sometimes a simple waveform is required from text that is to be kept and played at some later time. The simplest way to do this with festival is by using the `text2wave' program. This is a festival script that will take a file (or text from standard input) and produce a single waveform.

An example use is

```
$text2wave myfile.txt -o myfile.wav
```

Selecting the Voice of the synthetic speech

Find out the voices available in version of festival

```
festival>(voice.list)
(kal_diphone akl_nz_jdt_diphone rab_diphone)
```

Change to desired voice

```
festival> (voice_akl_nz_jdt_diphone)
```

Now synthesise text

```
Festival> (SayText….
```

And in text mode...

Create a *.scm file with both the voice selection command and the text embedded in a SayText command.

trial.scm

```
(voice_rab_diphone) ;; select Gordon
(SayText "Hello there")
(voice__akl_nz_jdt_diphone) ;; select Jono
(SayText "and hello from me")
```

Now synthesise the text.

```
$festival –b trial.scm
```

(but it can be a bit slow)

More Useful Festival Commands

Saving speech to wav files within Festival

```
(utt.save.wave (SayText "Put something here") "name.wav" 'riff)
```

"name.wav" will appear in the same directory that you ran festival from unless you put the path into.

Changing the rate of speech

To slow down

```
(Parameter.set 'Duration_Stretch 2.2)
```

To speed up

```
(Parameter.set 'Duration_Stretch 0.8)
```

Eg.

```
(Parameter.set 'Duration_Stretch 2.2)

(SayText "Tell me why there is no sun up in the sky, stormy weather.")
```

Changing the pitch

NB Pitch for Men approx. 105 Hz, For Women approx. 180 Hz

Eg Increase Pitch range over utterance

```
(set! duffint_params '((start 150) (end 100)))
(Parameter.set 'Int_Method 'DuffInt)
voice
```

```
Useage
(set! duffint_params '((start 150) (end 150)))
(Parameter.set 'Int_Method 'DuffInt)
(Parameter.set 'Int_Target_Method Int_Targets_Default)
(SayText "Tell me why there is no sun up in the sky, stormy weather.")
```

Different Emotions

Neutral: pitch range about 15 Hz across utterance for men , i.e start 120 Hz, end 105 Hz, rate of speech default –e.g. 1

Happy: increase both pitch range and pitch average eg start 135 end 110, and increase rate of speech eg to 0.8

Sad: decrease pitch range, and pitch average, and increase rate of speech.

Another *scm file example

Create a *.scm file with both the voice selection command and the text embedded in a SayText command.

Demo.scm

```
(utt.save.wave(SayText “Here is an example about speech rate”)
"intro.wav" 'riff)

(Parameter.set 'Duration_Stretch 2.2)
```

```
(utt.save.wave (SayText “Do you find this speech too slow?”) "slow.wav"
'riff)

(Parameter.set 'Duration_Stretch 0.8)

(utt.save.wave (SayText “Or perhaps it is far too fast?”) "fast.wav"
'riff)
(set! duffint_params '((start 150) (end 150)))
(Parameter.set 'Int_Method 'DuffInt)
(Parameter.set 'Int_Target_Method Int_Targets_Default)
(utt.save.wave (SayText “But is fast monotone that I find the worst.”)
"mono.wav" 'riff)
```

To process the scheme file go

```
>festival –b demo.scm
```

This will run the scm file, it will play the speech, and save the phrases in to the appropriate wav file.

Alternatively use the options with the text2wave function (this is probably better)

```
text2wave -o some.wav some.txt -eval slow.scm
```

where slow.scm is

```
(Parameter.set 'Duration_Stretch 2.2)
```

text2wave also has a volume option (eg `–s 50`) , to see all the options try

```
text2wave -h
```

`text2wave` will be faster than `festival –b foo.scm`

Singing

```
(tts "song.xml" 'singing)
```