This document was created by YK 2011-07-25, a user of Festival with Flinger (Festival-Flinger).

Table of Contents

Overview

This document assumes that you are familiar with what Festival v2.1 and Flinger v1.1.2 are/do, and want to know more about this particular distribution. My goal was to quickly get singing to work with Scooty, my ASIO VST sequencer program; not to provide a clean, fully-tested, Windows-version port of Festival and Flinger. Some features such as --libdir will probably be broken.

This Festival-Flinger user (me!) Windows distribution is comprised of Festival code, OGI-Flinger code, speech_tools code, and some default TTS voices. The codes compile to create festival.exe. Festival is the main code, OGI-Flinger is add-on code, and speech_tools is a general code library.

This Version of Festival:

FTLIBDIRC=c:\\festival\\lib;
FTOSTYPEC=win32_vc;
DFTNAME=Festival Speech Synthesis System;
FTVERSION=2.1;
FTSTATE=release;
FTDATE=November 2010

About This Distribution

This distribution is targetted for MS Windows developers and users. There are some of my hacks in the code that will break on other platforms. If you are interested in portability, then I suggest you download the latest original versions of Festival, speech_tools, OGI-Flinger, and TTS voices from the world wide web and start from there.

In this distribution you will find all of the code needed to compile festival-Flinger to a flinger.exe executable using MS Visual Studio 2008:

Festival code package (with HTML documentation from the texi file)
OGI-Flinger code package
Speech-tools code package
Some TTS voices from the download locations

The Festival source code came from a compilation by Cameron Wong, posted on his eGuideDog website. I'm guessing his package remains true to the original v2.1, but with all of the files' EOLs changed to CRLFs. Because URLs tend to change, I won't give external web URLs here. I suggest you surf the web to locate his website, if you're interested in the steps he took to port Festival to Windows. He posted his version of Festival v2.1 and v1.9.6, but not Flinger, or the speech_tools code needed to compile to completion. (The source code should contain URLs in the comments to identify where the source came from.)

The speech_tools and OGI-Flinger code are from their original sources on the web. If you are interested, my website has the other OGI TTS voices (except German) packaged as an installable, for use with this distribution. And Scooty is there, too.

I created this distribution on the following platform:

1.8GHz computer with MS Windows XP Pro SP3 installed
MS Visual Studio 2008 Pro, version 9.0.etc...
Cygwin v2.738 (you don't need it to compile this distribution)

You'll need MS Visual Studio 2008 to compile this distribution.

Installing This Distribution

The licences from the sources state that when I post Festival-Flinger for download, I must include the original code with my code changes identified (see next section). When installed, the Festival-Flinger portion of the distribution occupies two root folders on drive C:

- C:\festival for the Festival-Flinger code and executable.
- C:\speech_tools for the speech_tools library code.

Everything MUST reside in these target folders. My code hacks rely on this. Also, your temporary folder path (TMPDIR, TEMP, TEMPDIR etc.) must be on drive C. For just about everyone this is the default.

I created this distribution to provide external singing functionality for Scooty, my ASIO VST sequencer. Therefore, this distribution package is really a module useable by Scooty.

The distribution package uses the freeware Inno Setup installer for easy install and uninstall.

Changes I Made To The Source

I've created and added two visual studio *solutions* to the distribution, one for Festival and one for the speech_tools. I'll explain these later.

My visual studio is set to level 3 warnings. There were hundreds of typecast warnings when I compiled for the first time. I cleaned these up as best I could. The code is spotted throughout with my tag, //YK 2011-07-25 to identify my code changes. There are two or three warning codes that I didn't know how to fix (the more changes one makes, the greater the possibility of adding new bugs), so I changed the project settings to suppress them. They really should be cleaned up by an expert. Two warnings will appear in the compiler output window, and they're about returning a pointer to a temporary variable. I left them alone:

```
3>..\..\..\..\..\..\speech_tools\base_class\EST_Pathname_win32.cc(198) : warning C4172: returning address of local variable or temporary
3>..\..\..\..\..\..\speech_tools\base_class\EST_Pathname_win32.cc(200) : warning C4172: returning address of local variable or temporary
```

Three other warnings may appear. They alert you to the fact that their files do not contain callable code, only structs/defines/etc. I don't know why these are considered “warnings.”

```
3>EST_TVector.obj : warning LNK4221: no public symbols found; archive member will be inaccessible
4>OGI_Track_file_templates.obj : warning LNK4221: no public symbols found; archive member will be inaccessible
4>resLPC.obj : warning LNK4221: no public symbols found; archive member will be inaccessible
```

The German voices require a lexicon which is not included in their source tarballs (a unix term for archive file) since it's copyrighted. I did not install the German voices. I did install ogi_as_diphone and ogi_aec_diphone voices. The other voices can be downloaded from their original sources on the web, or from my own website where I've repackaged them using the freeware Inno Setup installer for the convenience of us Windows users. If you download them from the original sources, they are tarballs which will require Cygwin (or similar unix emulator, or unix) to unpack. Search for any website that explains “tar” for instructions to do this.

I've added a line to initialize OGI-Flinger upon startup to script file init.scm.
I've added some more lines to ogi_configure_voice.scm.
I've renamed libestools.lib to libesttools.lib in the visual studio solutions.
I've changed the scaling for midi cc event 30 so that crescendo now works.

Finally, some real code changes needed to be made to get the final executable to run properly. From memory, these are:

- Adding a 'debug mode because the original coder added a file dump feature to Flinger and didn't restore stdout/stderr, which caused the program to hang (actually, not showing the prompt so the user is staring at a blank window wondering if it hung). I discovered restoring them isn't simple, so I wrapped his file dump code with a debug mode switch. Normally you would type the festival command (Flinger.sing mymidfile.mid nil) to sing mymidfile.mid. To sing, create the dump file and then hang, type (Flinger.sing mymidfile.mid 'debug).
- My computer's temp environment variables use the DOS format, ie: c:\temp. This caused problems when trying to run an XML script in festival. I've made some hacks to get this to work, but it means the code isn't portable to unix etc. Search for ESTLIBDIR in the code for an example.

- Real stupid errors like empty control statements, missing braces, missing break statements, etc. that generated compiler errors or were bugs. Fortunately these were very few.

How To Get It To Compile

I hate makefiles. The coder who created the latest version of OGI-Flinger didn't update the makefiles, either. So I created two visual studio *solutions*, one to compile Festival-Flinger to a final festival.exe executable, and the other to build the speech_tools library test suite for debugging/testing. The solutions are in:

C:\festival\src\lib\SolutionVC\SolutionVC.sln
C:\speech_tools\lib\SolutionVC\SolutionVC.sln

The speech_tools solution is only for testing that library, which I didn't bother to do. Use the festival solution to create the final executable. Start up your visual studio, load the festival solution, and do a "rebuild solution" to recreate C:\festival\src\main\festival.exe.

(Rest of page left unintentionally blank because no one carries a pen anymore.)

(Insert doodle here)

Running Festival-Flinger

Start a Command Prompt window (DOS window, DOS box, DOS prompt, whatever) and type:

```
C:
cd \festival\src\main
festival
```

This will start festival in command mode. For an easy demo, type these commands shown in blue:

```
C:\festival\src\main>festival

FTNAME 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
clustergen_engine: Copyright (C) CMU 2005-2010
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)'
```

To speak "I am a talking computer voice" give the following command:

```
festival> (SayText "I am a talking computer voice.")
#<Utterance 01256A30>
```

To save your spoken text to a wave file in c:\festival\src\main:

```
festival> (utt.save.wave (SayText "Hello world.") "testtalk.wav" 'riff)
#<Utterance 0113EC20>
```

To see a list of installed voices:

```
festival> (voice.list)
(kal_diphone
 ogi_aec_diphone
 ogi_as_diphone
 ogi_jph_diphone
 ogi_mwm5_diphone
 ogi_rab_diphone
 ogi_tll_diphone
 ogi_abc_diphone
 ogi_hvs_diphone)
```

To show information about a voice:

```
festival> (voice.description 'ogi_aec_diphone)
(ogi_aec_diphone
 ((language english)
  (gender male)
  (dialect american)
  (description
   "This voice provides an American English male voice using a
      residual excited or sinusoidal LPC diphone synthesis module created at
      OGI.  It uses a lexicon compiled from MOBY and CMU lexicons, and
      other trained modules used by CSTR voices.")
  (samplerate 16000)))
festival>
```

To change to voice ogi_as_diphone:

```
festival> (voice_ogi_as_diphone)
nil
```

To sing a midi file containing a melody track with embedded lyrics:

```
festival> (Sing "\\festival\\src\\modules\\flinger\\examples\\mist-me.mid")
...
...
...
#<Utterance 011FE2E0>
```

Another command to sing a midi file:

```
festival> (Flinger.sing "\\festival\\src\\modules\\flinger\\examples\\ogi.mid" nil)

0 Bar:1 Beat:1  Adding common MetaTime Event 4/4, Clocks per Beat 24, 8 demisem

0 Bar:1 Beat:1  Adding common MetaTempo Event, Value 109

0 Bar:1 Beat:1  Adding track MetaPortNumber Event 0
#<Utterance 011FD850>
```

To save the singing to a wave file in c:\festival\src\main:

```
festival> (utt.save.wave (Flinger.sing "\\festival\\src\\modules\\flinger\\examples\\ogi.mid" nil)
"testsing.wav" 'riff)
#<Utterance 01F22300>
```

To sing without using OGI-Flinger:

```
festival> (tts "\\festival\\examples\\songs\\doremi.xml" 'singing)
00002"replaced by slasheshes in URL path "\DOCUME~1\Rei\LOCALS~1\Temp\est_03060_
00002"replaced by slasheshes in URL path "\DOCUME~1\Rei\LOCALS~1\Temp\est_03060_
nil
```

To exit Festival-Flinger:

```
festival> (quit)


C:\festival\src\main>
```

To run a script in script mode: this example runs scfg_parse_text.sh which parses an input text file using a given SCFG. I don't know what an SCFG is. See the script file for more information.

```
C:\festival\src\main>festival --script "\festival\examples\scfg_parse_text.sh" "
\festival\examples\intro.text"
(("NT00"
  ("NT14"
   ("NT06" ("dt" "This"))
   ("NT17"
    ("NT04" ("vbz" "is"))
    ("NT08"
     ("NT02"
      ("NT01" ("NT01" ("dt" "a")) ("NT16" ("jj" "short")))
      ("NT07" ("nn" "introduction")))
     ("NT09"
      ("NT15" ("to" "to"))
      ("NT02"
       ("NT01" ("dt" "the"))
       ("NT07"
        ("NT07" ("nn" "Festival"))
        ("NT07"
         ("NT07" ("nn" "Speech"))
         ("NT07" ("NT07" ("nn" "Synthesis")) ("NT07" ("nn" "System"))))))))))
  ("NT13" ("punc" "."))))

(("NT00"
  ("NT14"
   ("NT07" ("nnp" "Festival"))
   ("NT17"
    ("NT04" ("vbd" "was"))
    ("NT08"
     ("NT04" ("vbn" "developed"))
     ("NT08"
```

```
      ("NT03"
       ("NT09"
        ("NT15" ("in" "by"))
        ("NT10"
         ("NT02"
          ("NT01"
           ("NT07" ("NT07" ("nnp" "Alan")) ("NT07" ("nnp" "Black")))
           ("NT05" ("cc" "and")))
          ("NT07" ("nnp" "Paul")))
         ("NT12" ("nnp" "Taylor"))))
       ("NT13" ("punc" ",")))
      ("NT09"
       ("NT15" ("in" "at"))
       ("NT10"
        ("NT02" ("NT01" ("dt" "the")) ("NT07" ("nnp" "Centre")))
        ("NT12"
         ("NT15" ("in" "for"))
         ("NT10"
          ("NT02"
           ("NT01"
            ("NT07"
             ("NT07" ("nn" "Speech"))
             ("NT07"
              ("NT07" ("nn" "Technology"))
              ("NT07" ("nn" "Research"))))
            ("NT05" ("punc" ";")))
           ("NT07" ("nnp" "University")))
          ("NT12" ("NT15" ("of" "of")) ("NT07" ("nnp" "Edinburgh")))))))))
  ("NT13" ("punc" ".")))

C:\festival\src\main>
```

Documentation

I suggest you look at C:\festival\doc\festival\index.html and the scripts in the Examples folder. I ran the makefile utility in Cygwin to create the html pages from the supplied source texi file found at C:\festival\doc\festival.texi.

Personal Notes

This section of the document is a scratchpad for my personal notes.

- Tried lyric “la” and one note. Failed
- Tried two lyrics and two notes. Worked
- Need to try 1 lyric 2 notes, 2 lyrics 1 note, etc.

Parameters (more is in OGIeffects and OGIresLPC):

volume_scalebase
Adjustment to velocity. A volume_scalebase value of 10, for instance, provides a useful scale of 1-10. However, MIDI files can carry a value up to 0xFF (255) so this has to be allowed for.

vibrato_frequency
Vibrato frequency (Hz).

pitch_effects_max
Base values for vibrato & pitch bend (semitones).

```
sfVibratoAmplitude = sfVibratoAmplitude_in * sfPitchEffectsMax;
sfVibratoPeakFraction = (float) pow(2.0, sfVibratoAmplitude/12.0) - (float)1.0;
// 12 semitones in one octave. One octave requires a ration of 2.0.
// This gives peak-peak pitch variation of 2 * sfVibratoAmplitude.
```

portamento_time
Portamento is the glide from one note to another in msec. I guess this is the amount of time taken to glide from one note to another.

```
convert input value (milliseconds) to seconds.
//DJLB 10/06/2008 Correction to scaling.
sfPortamentoTime = (float)value/(float)1000.0; // 100.0;
```

drift_frequency_1
Drift freq 1 (Hz).

```
// Convert drift amplitude from semitones to fraction of frequency.
sfDriftPeakFraction = (float) pow(2.0, gl->F("drift_amplitude")/12.0) - (float)1.0;
boDrift = ((sfDriftF1 > 0.0) || (sfDriftF2 > 0.0) || (sfDriftF3 > 0.0)) && (sfDriftPeakFraction > 0.0);
```

/* This is arranged so that the peak drift is 1, regardless of how many of the three drift frequencies are active. */
Strange code. boDrift is true only when all 3 drift_frequency values are > 0.0. But calculating the drift assumes that some of the frequencies can be 0.0 when boDrift is true.

drift_frequency_2

drift_frequency_3

drift_amplitude
See drift_frequency_1.

transpose
In +/- semitones.

```
uiTranspose = (unsigned int)gl->F("transpose");// +/- semitones.
note_num += uiTranspose;
```

There's no bounds checking! note_num could go below 0 or above the max. Suggest always using transpose = 0.

consonant_stretch

```
ph_durs = siod_get_lval("fl_phoneme_durations","no phoneme durations");
```

Affects the duration of consonants. Need to set up segments or something, else you may get:

```
cerr << "WARNING: Phoneme '" << s->name()
    << "' has no default duration listed, so set to 0.1 sec." << endl;
```

Strange. Mist-me.mid doesn't display the above warning, but the other midi files do. Weird. I don't think its the trailing space in the lyrics.

min_rest_dur

```
Rt = max((float)0.0, next1stON-this1stON - (Vt+Ct+Ot));
if (Rt < gl->F("min_rest_dur")){
    Rt = 0.0;
}
```

Part of some hairy-scary code in get_syllable_target_durations.

phone_delimiter
Woohoo, this is read in, but used nowhere! Maybe in a script file?

syl_continuation
Woohoo, this is read in, but used nowhere! Maybe in a script file?

voice_track_name

voice_track_number
channel_number
common_events_track_name
common_events_track_number
use_just_tuning
These are used by the Just Tuning feature. Common_events variables are for common events. Didn't try the common events feature. Couldn't hear three-part harmony and the sound was tinny.

Try tempo event in track 0, lyrics+notes in track 1, set use_just_tuning to yes.
Using multiple tracks, with and without different midi channel numbers, didn't work. Maybe I should have added a tempo event to track 0, pushing the rest of the tracks down by one?

DJLB 10-08-08. Provision of alternative track identification, by name or number.

```
                //If the track name is nominated, use it. If not, use the track number.
                // If neither are given, the 1st track on which notes are found will be used.
                char chNum[18];
                esLocation = "Track ";
                sprintf(chNum, "%d", inReqdTrackNumber);
                esLocation += chNum;


                inReqdTrackNumber = DEF_TRACKNUMBER;
                esLocation = esReqdTrackName + " Track";

        Flinger will handle MIDI files in Format 0 or 1 only.
        In Format 0, there is only one track (0) and all events are in it in their
        chronological order.
        In Format 1, a number of tracks may be present. Track 0 is commonly unnamed,
                and contains file descriptors and some MIDI events - notably tempo events,
                with info. about their intended time location.
        Tracks 1 and up may contain notes and/or lyrics, volume changes etc. Each track
                may be named, the track name being provided by the METASEQUENCENAME event.
        The following code allows Flinger to:-
        (a) reject a file of Format > 1;
        (b) process the track in a Format 0 file;
        (c) process the tracks in a Format 1 file with the following provisos:-
                (1) events on track 0 are stored and added to utt when appropriate, during
                        the later processing of a track that contains notes;
                (2) if a track name has not been provided by the user (this can be done
                        through Flinger parameters) all tracks up to and including the next
                        track to contain note events are processed. Tracks following one with
                        note events are ignored;
                (3) if a track name has been provided, only the track so named is
                        processed. All others are ignored, and
                (4) if a track name for a track containing events intended to be common
                        to all note-bearing tracks has been provided, events on that track
                        are stored in a list, and merged with the note-bearing track when
                        that track's events are handled.

   /*NOTE: MIDI channel numbers are known as integers from 1 to 16, but
         are stored in the MIDI file as integers 0-15.*/
   int inReqdChannelNumber(gl->I("channel_number") - 1);
   if (!((inReqdChannelNumber < 16) && (inReqdChannelNumber > 0)))
Oops, shouldn't the above be >= 0? (I've fixed it.)
   {
      inReqdChannelNumber = DEF_CHANNELNUMBER;
   }
```

```
  else
  {
     char chNum[18];
     esLocation += " Channel ";
     //esLocation += itoa(inReqdChannelNumber + 1,chNum,10);
     sprintf(chNum, "%d", inReqdChannelNumber + 1);
     esLocation += chNum;
  }

  //DJLB 19-06-08. Channel selection modification & recognition of reqd. track number.
  if(esCommonEventsTrackName.length() <= 0)
  {
     if(inCommonEventsTrackNumber != DEF_COMMONEVENTSTRACKNUMBER)
     {
        if (!((inCommonEventsTrackNumber < 16) && (inCommonEventsTrackNumber > 0)))
Oops, shouldn't the above be >= 0? (I've fixed it.)
        {
           inCommonEventsTrackNumber = DEF_COMMONEVENTSTRACKNUMBER;
        }
        else
        {
           char chNum[18];
           esLocation = "Track ";
           //esLocation += itoa(inCommonEventsTrackNumber,chNum,10);
           sprintf(chNum, "%d", inCommonEventsTrackNumber);
           esLocation += chNum;
        }
     }
  }
  else
  {
     inCommonEventsTrackNumber = DEF_COMMONEVENTSTRACKNUMBER;
     esLocation = esCommonEventsTrackName + " Track";
  }
          Deal with Track 0 (in a Format 1 MIDI, this contains a tempo and time
          event map). Then deal only with the track and/or channel identified by
          Flinger input parameters, if that identification exists. Otherwise,
          deal with all tracks other than those following one with notes.
```

MIDI cc events:

```
          void FL_add_parameter_event(ParameterEvent *e, EST_String esType)
          case 1:   // Modulation
                    FL_add_vibrato( e->GetTime(), e->GetChannel(), e->GetValue());
          case 5:
                    FL_add_portamento_time(e->GetTime(), e->GetChannel(), e->GetValue());
          case 7:   // Volume for track.
                    FL_add_Volume(e->GetTime(), e->GetChannel(), e->GetValue(), FALSE);
          case 20: // DJLB addition - Gliss
                    FL_add_gliss(e->GetTime(), e->GetChannel());
          case 21:    // DJLB addition - 'notebend' - a pitchbend for current note only.
                    FL_add_notebend(e->GetTime(), e->GetChannel(), (long)e->GetValue());
          case 30:   // DJLB addition - Volume crescendo/diminuendo effect.
                    FL_add_Volume(e->GetTime(), e->GetChannel(), e->GetValue(), TRUE);
```

also active: tempo events, lyric events, note bars, timesig events (does not affect Flinger), end of track markers (obviously)

ignored: key pressure, program change, channel pressure (aftertouch), sysex, sequence#, text, copyright, sequence name, instrument name, marker, cue, channel prefix, port#, SMPTE, meta key, sequencer specific

MIDI pitch wheel event:

```
range [-8192,8192]
/* Max. value = 0x3FFF   must represent ~ +1
   Min. value = 0x00     must represent -1
   Mid-point value = 0x2000 must represent +/- 0.0 */

void Tclm_ParsePitchWheel(PitchWheelEvent *e)
```

My pitchwheel transmits values from $0 (0) to $7fff (32767) with $4000 (16384) as midpoint. The above is for midi v1.0 spec.

OGIresLPC:

"LPC" What does this stand for?
"res" What does this stand for?
"UV" = "unvoiced". What does voiced and unvoiced mean?
gmm = What does this stand for? General matrix mapping? A file containing mean, covariance, weight, bias info. Used by voice conversion

Features that are beyond my ability to decipher:

F0
T0
Anything that needs external files: vc_file, spectra, power
Anything that needs an input list: voice sections, pitchmarks

Adjustable parameters. Boldface=tested and sound changed for better/worse:

F0_default	possibly 0.0 up:
	// Functions for setting up duration modification.
	// Assumes we have
	// - "Target" relation with "pos" and "f0" features
	// - OGI_TimeWarp previously constructed
	//

T0_UV_thresh	possibly something to do with pitchmarks In.set("T0_UV_thresh", In.F("F0_range_factor") / In.F("guess_F0"));
T0_UV_pm	pmark (pitchmark) for unvoiced sections. Always seems to be unvoiced, voiced labelling flag isn't set to 1 anywhere?
post_gain	a multiplier, overall gain to apply, 1.0 means no gain
deemphasis	a multiplier to a value, 1.0 means no deemphasis
uv_gain	a multiplier, user-specified gain for unvoiced, 1.0 means no gain
uv_dither	1=true, else false
window_type	"trapezoid" or "triangle", default is triangle. Although I saw default can be "hamming" somewhere? hamming is not an acceptable input keyword, only trapezoid and triangle are. This param didn't affect the resulting sound
outbuf_incr	output wave buffer size. This param doesn't affect the resulting sound
mod_method	modify prosody to realize target output. "direct" method: interpolate within target F0 contour to

get target pitchmarks, use T0 of original pmarks in UV regions

dur_method Generate time warping function from durations

pitch_method "soft" doesn't appear to be implemented

vc_file provide a voice conversion mapping filename to activate voice conversion

vqual_mod has these subparams. This param didn't affect the resulting sound

```
VQUAL_global_warp_wave = get_param_float("vt_global_warp_wave",params, 1.0);
VQUAL_v_warp_wave = get_param_float("vt_voiced_warp_wave",params, 1.0);

VQUAL_global_warp_lsf  = get_param_float("vt_global_warp_lsf", params, 1.0);
      doesn't appear to be implemented
VQUAL_v_warp_lsf  = get_param_float("vt_voiced_warp_lsf", params, 1.0);
      doesn't appear to be implemented

float v_warp  = VQUAL_global_warp_wave * VQUAL_v_warp_wave;      for voiced
float uv_warp = VQUAL_global_warp_wave;                          for unvoiced
```

spectra_smooth Apply correction to smooth LSF trajectories at joins. A list of phonemes needed? What's a phoneme?

spectra_match_or_replace active if spectra_smooth is not "". Values are "" or "replace" or "match"
"match" = add an offset to each channel, increasing linearly from frB-->frM and frM<--frE, such that left and right sides meet at frame frM-0.5
"replace" = same as above, but replace instead of adding an offset
(there appears to be a smooth_ma proc, but it's not used.
smooth_ma = moving average with antisymmetric extension of endpoints. Keeps endpoints of vector same after filtering. The code is there but nothing references it!

power_smooth Apply correction to smooth power at joins. A list of phonemes?

power_match_or_replace active if power_smooth is not "". "replace" or "match". Same value meaning as spectra_match_or_replace

smooth_cross_ph_join used by spectra_smooth and power_smooth functions

```
// Get params from Scheme layer
In->set("F0_default",  get_param_float("F0_default",  params, 50.0));
In->set("T0_UV_thresh",get_param_float("T0_UV_thresh",params, (float)0.020));
In->set("T0_UV_pm",    get_param_float("T0_UV_pm",    params, (float)0.010));
In->set("post_gain",   get_param_float("post_gain",   params, 1.0));
In->set("deemphasis",  get_param_float("deemphasis",  params, (float)0.94));
In->set("uv_gain",     get_param_float("uv_gain",     params, 1.0));
In->set("uv_dither",   get_param_int("uv_dither",     params, 1));

// triangle, trapezoid
In->set("window_type", wstrdup(get_param_str("window_type",params, "triangle")));

In->set("outbuf_incr",  get_param_int("outbuf_incr", params, 1024));

// direct, none
In->set("mod_method",  wstrdup(get_param_str("mod_method",params, "direct")));

// direct, natural
In->set("dur_method",  wstrdup(get_param_str("dur_method",params, "direct")));
if (gl->S("dur_method") == "natural")

// direct, natural, soft
In->set("pitch_method",  wstrdup(get_param_str("pitch_method",params, "direct")));
if (gl->S("pitch_method") == "natural")

  // Voice conversion/quality mod
In->set("vc_file", wstrdup(get_param_str("vc_file",params, "")));
```

```
plisp = get_param_lisp("vqual_mod", params, NIL);
if (plisp != NIL){
  In->set("vqual_mod",  siod_sprint(plisp));
}

// Smoothing
plisp = get_param_lisp("spectra_smooth", params, NIL);
if (plisp != NIL){
  In->set("spectra_smooth",  siod_sprint(plisp));
}

plisp = get_param_lisp("power_smooth", params, NIL);
if (plisp != NIL){
  In->set("power_smooth", siod_sprint(plisp));
}

// "Y" or "N" - applies to power and spectra
In->set("smooth_cross_ph_join",
      (int) atobool(wstrdup(get_param_str("smooth_cross_ph_join",params, "N"))));

In->set("spectra_match_or_replace",
      wstrdup(get_param_str("spectra_match_or_replace",params, "match")));
In->set("power_match_or_replace",
      wstrdup(get_param_str("power_match_or_replace",params, "match")));

// list of dump info for debugging outputs
plisp = get_param_lisp("dump", params, NIL);
if (plisp != NIL){
  In->set("dump", siod_sprint(plisp));
}
```

OGI_Effects:

Effects are applied to a wave utterance. See festival_OGIeffect_init. Fortunately no extra work is involved to get these to do something.

effect_order	values are “reverb” “echo” “slapback”. You need to always use this command.
reverb	a list of taps, each tap has echo times (int), weights (float), and channel LPF's (reflectance of walls) (float)
echo	a list of echoes, each echo has delay time (int), and feedback (float)
slapback	a list of slapbacks, each slapback has delay time (int), and feedback (float)
	This param didn't change the resulting sound. Maybe my numbers were wrong.
	// take slapback_dtime + echo_dtime * Niter_until_w^N=0.01)
post_gain	gain to apply after effects
pre_gain	gain to apply before effects

```
LISP OGIeffect_Init(LISP params){

  lispprm   = get_param_lisp("effect_order",params,NIL);
  lispprm   = get_param_lisp("reverb",params,NIL);
    lispprm   = get_param_lisp("echo",params,NIL);
  lispprm   = get_param_lisp("slapback",params,NIL);
  In->post_gain = get_param_float("post_gain",params,1.0);
 In->pre_gain = get_param_float("pre_gain",params,1.0);
```

Resynthesize OGIresLPC.resynth:

This stuff is beyond me. Tried to get it to do something, but don't know how to. All but OGIresLPC.init.

```
  // parse input file list (this is the FILELIST when calling OGIresLPC.resynth in the script)
  pmfile   = wstrdup(get_param_str("pm_file", files, ""));
  wavfile  = wstrdup(get_param_str("wav_file",files, ""));
  lsffile  = wstrdup(get_param_str("lsf_file",files, ""));
  modlsffile = wstrdup(get_param_str("modlsf_file",files, ""));

void festival_OGIresLPC_init(void){
  proclaim_module("OGIresLPC");

     init_subr_1("OGIresLPC.init",OGIresLPC_Init,
    "(OGIresLPC.init PARAMS)\n\
  Initialize and set parameters of resLPC synthesizer. \n\
   PARAMS is an assoc list of parameter name and value.");

     init_subr_1("OGIresLPC.init++",OGIresLPC_AddInit,
    "(OGIresLPC.init++ PARAMS)\n\
  Add to initialization parameters of resLPC synthesizer \n\
  without destroying existing settings. PARAMS is an assoc list \n\
  of parameter name and value.");

    init_subr_1("OGIresLPC.synth",OGIresLPC_Synthesize,
    "(OGIresLPC.synth UTT)\n\
  Synthesize a waveform using the currently selected resLPC database.");
                        // (Time-domain modification of residual and LPC filtering both at once)
                        uses:
                        post_gain
                        uv_dither
                        deemphasis
                        window_type
                        vc_file
                        vqual_mod
                        outbuf_incr
                        pulses    0=no dump, 1=dump. Part of dumplist, when dump is turned on

    init_subr_1("OGIresLPC.lpc_analysis",OGIresLPC_LPC_Analysis,
    "(OGIresLPC.lpc_analysis PARAMS)\n\
   Do LPC analysis on speech files to build an OGIresLPC database. \n\
   PARAMS is an assoc list of parameter name and value.");

    init_subr_1("OGIresLPC.pmark_analysis",OGIresLPC_Pmark_Analysis,
    "(OGIresLPC.pmark_analysis PARAMS)\n\
   Do pitchmark analysis on speech/laryngograph files to build an OGIresLPC database. \n\
   PARAMS is an assoc list of parameter name and value.");

    init_subr_2("OGIresLPC.resynth",OGIresLPC_Resynth,
    "(OGIresLPC.resynth UTT FILELIST)\n\
  Resynthesize wavefile using prosodic modification\n\
  info in UTT.  Must have set Segment, Target, and SrcSeg relations. \n\
  FILELIST is an assoc list of keys and associated filenames.");
}

(utt.save.wave (Flinger.sing "scooty.mid" nil) "1_mod_method.wav" 'riff)
```