FTAP: A Linux-based program for tapping and music experiments

Reference Manual

Version 2.1.05

Steven A. Finney

November 1, 2001

Contents

1 Introduction

FTAP is a Linux program for running a variety of psychology experiments with verified millisecond-resolution data collection.[1] It allows manipulation of auditory feedback in response to a user's key presses using MIDI devices for input and output. You are currently reading the Reference Manual, which assumes that you have read either the User's Guide or the article on FTAP in *Behavior Research Methods, Instruments, and Computers* (Finney, 2001a). That is, I will assume that you already understand the purpose and overall design of FTAP. In addition, it will be helpful if you have some knowledge of Linux/UNIX and that you know what MIDI is, as I cannot provide extensive background information in these areas. If you think that FTAP might be useful to you but you know nothing about Linux, you may need to seek outside assistance. (FTAP has been written for Linux because it's what I know how to program; if you want to port it to a different OS, you are free to do so, under the confines of the GNU license. Please contact me if you plan to do this.)

FTAP is provided in source code form at no charge (but without warranty) under the GNU Public License agreement. My understanding is that the GNU license allows you to use and modify the software in any way you like for "personal" use, but that any further distribution of the modifed software must be done in source code form under the GNU license agreement. My intent is that FTAP be available for unlimited use and modification; you can download the FTAP distribution from http://csml.som.ohio-state.edu/ftap. If you have comments, bug reports, bug fixes, enhancement requests, or coded enhancements, I would like to know so that I can deal with them and/or choose to incorporate them into the standard version of FTAP. Send comments and questions to sf@csml.som.ohio-state.edu.

NOTE: I am confident enough in this program that it has been my main system for data collection for the last 4 years or so. I have tested it extensively and it is the subject of various talks and writeups for publication. However, as of this writing it has received serious use from only a few other people, so it should perhaps best be viewed as a "beta" version. If you think it might be useful, try it out and

[1] FTAP began as a program I wrote for an SGI Indigo which I used for my research as a graduate student at Brown University. I thank Jim Anderson, David Ascher, Peter Eimas, Mike Tarr, and Bill Warren for their assistance during this time. The port to Linux and many important enhancements were completed during a post-doctoral fellowship at OSU; I thank Caroline Palmer, Pete Pfordresher, and Shane Ruland for their assistance. I'd also like to thank David Huron and Doug Reeder for providing a web home for FTAP at http://csml.som.oho-state.edu/ftap .

In addition, some other people have actually started using FTAP! Thanks to Grant Baldwin, Zeb Highben, and Anna-Lisa Ventola of Caroline Palmer's lab, and special thanks to Saul Sternberg for helpful comments and for being the first brave soul to install and use FTAP on his own.

Steve should be reachable at either sf@sfinney.com or sf@csml.som.ohio-state.edu.

let me know what you think. Although the documentation (this Reference Manual and the User's Guide) should be both accurate and complete, the organization and writing still need a fair amount of work. Preparation for public distribution (e.g., adequate documentation) has been a large task, and often has to take second place to my actual research.

Please cite Finney (2001a) in the writeup of any research you do that uses FTAP.

2 Installation: Initial Use and Evaluation

This section will attempt to provide the minimal information you need to be able to work (or play) with FTAP on a Pentium-based Linux system. With this setup, FTAP should will work fine, but data collection times may be a significant number of milliseconds off (unless you're on a very heavily loaded machine, this is not likely to be perceptible). If you decide to use FTAP for millisecond-resolution data collection, there are a number of additional concerns; see Section 9.

2.1 Distribution and Installation

The distribution (as downloaded from the web site) is in 'gzip'ed and 'tar'ed format. Run 'gunzip' on the downloaded file, and then extract the directory hierarchy using 'tar'. You will now have a subdirectory named 'ftap2.1.05' (or something close to that). Look at the RELEASE_NOTES file, and then put the 'bin/ftap' binary somewhere where you can execute it (e.g., in your personal 'bin' directory). There are no auxiliary files. If FTAP will be used by multiple users on your system, you might want to put it somewhere like '/usr/local/bin', but at that point you'll probably also want to recompile it from the provided source code and make it 'setuid root' (see Section 9).

2.2 Computer and Operating System

I have run FTAP on a couple of different Pentium boxes (200 MHz and up), running RedHat distributions 5.2, 6.1, 6.2, and 7.0. (2.0 and 2.2 Linux kernels). FTAP has also been run (but not successfully compiled) on a Mandrake 8.0 distribution with a 2.4 kernel. The provided binary should work on most Intel-based Linux systems. It might even work if recompiled on a totally different architecture; let me know what happens if you try this.

See Appendix C for additional information.

2.3 MIDI and Devices

FTAP uses MIDI for both data collection and (auditory) stimulus presentation, so you need a MIDI hardware interface of some kind with both an input and output port, and an installed Linux driver for that interface (in fact, currently the commercial MIDI driver from 4Front technologies (http://www.4front-tech.com) appears to be required; see Section C.2). FTAP accesses the raw MIDI stream using the device "/dev/midi". Next, you will need a MIDI keyboard for input and a MIDI tone generator for output. To run the demonstration files, you should set both of the devices to transmit/respond on MIDI channel 1. If you have a keyboard which allows you to turn off MIDI LOCAL mode, you can use that keyboard for both input and output if you disable LOCAL mode. If you have no external tone generator, but have a MIDI sound card, I have no idea what will happen. Let me know.

There are a number of details regarding the MIDI output generated by different keyboards, and the behavior of different card/driver combinations which are covered in later sections (e.g., Section 9 and Appendix D). If you're lucky, you won't need those details yet.

2.4 Demo Files

The 'params' subdirectory contains a number of sample parameter files; you should just try executing some of the ones in 'demo' and 'sample_experiments". For instance, in the "params/demo" directory the following command should sound a patterned sequence:

```
ftap metronpat3
```

2.5 It Doesn't Work!

If you have followed the above directions, and are the sort of person who always buys winning lottery tickets, then FTAP should be working beautifully. If that is not the case, look at later sections of this manual (e.g., Sections 8 and Appendix D) and try to figure out what's going on. If you can't figure it out after making a good faith effort, send me email.

3 Usage

FTAP is invoked as follows:

```
ftap <paramfile>  [param_override*]
```

The optional parameter override(s) allows for parameter specifications which take precedence over parameter specifications in the input file. The most common use of overrides is for specifying things like subject ID or trial sequence number, which are really not part of the experiment description per se and thus don't belong in the parameter file.[2] A parameter override must be in double quotes so that the Linux shell passes it to FTAP as a single argument, e.g. "SUB 1" or "NRDELAYS 4 10 20 30 2000". In actual data collection use, FTAP would normally be invoked with 3 parameter overrides, for subject (SUB), trial (TRIAL), and block (BLOCK). The parameter file name, subject, trial, and block are all (obligatorily) encoded in the output file name. FTAP will happily overwrite an existing output file without giving any warning.

An invocation of FTAP in an actual experiment might thus be the following; the resulting output file would be named "250p_250.1.2.7.abs":

```
ftap 250p_250 "SUB 1" "BLOCK 2" "TRIAL 7"
```

A run of FTAP may be terminated by typing Ctrl-C (or the equivalent keyboard interrupt character on your system).

When FTAP starts, it will print "Running as normal user" if you are running it without having done any special configuration. If you have set it up for real data collection (see Section 9), it will print "Running with realtime privileges". When FTAP terminates, it will print out a timing diagnostic like the following:

```
Mean time between sched()'s (ms): 0.01, schedcnt: 194077
> 1 ms: 1, > 5 ms: 0, > 10 ms: 0, max: 2 ms
```

This output is significant if you are doing real-time data collection, and it will be explained further in Section 9.5. It provides information about FTAP's internal timing on this run.

In research use, it may be best to run FTAP from a script that keeps track of trial numbers and parameter file locations, and that takes care of trial randomization. Appendix F contains an example of such a script in the Python language.

If FTAP is invoked with *no* arguments, it will enter an interactive mode, which allows running parameter files, changing parameters, etc. This mode was once useful for investigating FTAP, but is no longer supported. If you insist, see Appendix G.

[2] Overrides could also be used for using a single parameter file for a range of experiments in which only one value is changed (e.g., delay time), but it's really just as easy (and probably safer) to make separate parameter files for each condition.

4 Parameters

An FTAP trial is specified by a text file which lists parameters describing the experiment. A parameter is a user-settable value that controls some aspect of FTAP's behavior. Parameters are of three types, depending on the data value: integer, string, or array (of integers). In addition, triggers (which changes the value of an integer parameter value during the course of a trial) are also specified in the input file. The important parameter values for an experiment are listed in the output file; see Section 7.

4.1 Input File Format

The input parameter file is a simple text file containing parameter specifications. Any lines beginning with the '# ' comment character, or with a white space character, will be ignored. The format for a parameter specification is

```
<parameter name> <value>
```

This is also the format of override parameters, which are the easiest way to specify subject, trial, and block (see Section 3). If the value for a parameter is not specified by a user, a default value (documented in the following sections) will be used; this will usually lead to a rational level of inaction. If a parameter is important to your experiment, you should explicitly specify it rather than relying on the default. Additional documentation on FTAP parameters may be found in the source files "params.c" and "params.h".

A sample parameter file is provided in the User's Guide, and additional examples are provided in the 'params' directory of the distribution and in Section 5 of this Reference Manual. Looking at (and running and modifying) these files is one of the best ways to see how FTAP works.

4.2 Complete Parameter Listing

This section lists all of the parameters specifiable in an input file, including the default value if the user does not specify a value, and whether the parameter value will be printed by default to the output file. Parameters set by a user in a parameter file will always be listed in the output file, and setting the FULL_PARAM_PRINT parameter to 1 will cause *all* parameters can be printed to the output file. Any parameters not listed in the output file can be assumed to be set to the default values listed below.

The function of each parameter is explained in subsequent sections.

4.2.1 Integer Parameters

The following parameters take a single, non-negative integer as a value.

Name	Default Value	Default Print
FEED_ON	1	Yes
FEED_CHAN	1	Yes
FEED_LEN	0	Yes
FEED_PMODE	0	Yes
FEED_NOTE	96	Yes
FEED_DMODE	0	Yes
FEED_DVAL	250	Yes
FEED_VMODE	0	Yes
FEED_VEL	0	Yes
FEED2_ON	0	No
FEED2_CHAN	1	No
FEED2_LEN	20	No
FEED2_PMODE	0	No
FEED2_NOTE	80	No
FEED2_DMODE	1	No
FEED2_DVAL	250	No
FEED2_VMODE	0	No
FEED2_VEL	100	No
SPLIT_POINT	64	No
PITCHLAG	0	No
MASK_ON	0	Yes
MASK_CHAN	2	No
MASK_NOTE	64	No
MASK_VEL	35	No
METRON_ON	0	Yes
MET_CHAN	1	Yes
MET_NOTE	64	Yes
MET_VEL	100	Yes
MET_LEN	20	Yes
MSPB	600	Yes
STDOUT	0	No
CLICK1_OFFSET	0	No
CLICK2_OFFSET	0	No
FULL_PARAM_PRINT	0	Yes

4.2.2 String Parameters

The following parameters take a character string as a value. Although double quotes are used here for clarity, these quotes should *not* be used in the input file.

Name	Default Value	Default Print
SUB	"sub"	Yes
BLOCK	"block"	Yes
TRIAL	"trial"	Yes
COMMENT	""	No
CLICK1_FILE	""	No
CLICK2_FILE	""	No
PITCHSEQ_FILE	""	No

4.2.3 Array Parameters

The following parameters take an array of integers as a value. The first field is the array length, which is followed by the elements themselves. The default for all of these is an empty list (0 elements).

Name	Default Value	Default Print
RANDDELAY_ARRAY	0	No
MET_PATTERN_ARRAY	0	No
MET_VEL_ARRAY	0	No
MET_NOTE_ARRAY	0	No
MET_CHAN_ARRAY	0	No
MET_LEN_ARRAY	0	No

4.2.4 Output-only Values

In addition to the parameters which control how FTAP behaves, there are also certain diagnostic values that FTAP writes to the output file; these are listed below. These cannot be specified by the user. The diagnostic timing values are described in Section 7.1.

Name	Default Value	Default Print
TIME	---	Yes
VERSION_NUMBER	---	Yes
PARAMETER_FILE	---	Yes
AV_DELAY	---	No
SCHED_AV	---	Yes
SCHED_MAX	---	Yes
SCHED_MAXTIME	---	Yes
SCHED_GT1	---	Yes
SCHED_GT5	---	Yes
SCHED_GT10	---	Yes
IN_DISC_MAX	---	No
IN_DISC_MAX_TIME	---	No
OUT_DISC_AV	---	No

```
OUT_DISC_MAX          ---                    No
OUT_DISC_MAX_TIME     ---                    No
MIDI_ERROR            ---                    Yes
```

4.3 Integer Parameters

Integer parameters take a single non-negative integer as value. In some cases, this will be the values 0 and 1, for 'off' and 'on' respectively. For the feedback parameters FEED_CHAN, FEED_VEL, FEED_PMODE, FEED_DMODE, and FEED_VMODE, a value of 0 means that the input keystroke values will be echoed in the output, while a non-zero value means that there will be some alteration. Integer parameters (only!) are the ones which can be changed during the course of an experiment by trigger events.

4.3.1 Feedback Parameters

These parameters specify characteristics for feedback to a subject's keystrokes. A second feedback channel is handled by FEED2 equivalents to the FEED parameters documented here; see the discussion in Section 4.7.2. Most experiments will not require the second feedback channel.

Certain parameters are dependent on other parameters. For example, if FEED_ON is set to zero, none of the other feedback parameters will have an effect.

Most of these mapping characteristics are independent and can be freely manipulated and combined. Extension of mapping modes beyond what is provided here can only be done by modifications to the C source code, but this is fairly straightforward (see Appendix H).

1. FEED_ON: If set to 0, there will be no auditory feedback, that is, no MIDI output in response to keystrokes, and hence no sound. If set to 1, feedback is turned on. Values of 2 and 3 relate to the use of multiple feedback channels and a keyboard split; see Section 4.7.2. [Default: 1]

2. FEED_CHAN: The MIDI channel to be used for feedback: 0 means the input value from keystrokes will be unchanged, while values from 1-16 will send the feedback out on the specified MIDI channel. MIDI channel control allows specification of different sounds with a polytimbral tone generator. [Default: 1]

3. FEED_LEN: The length (in milliseconds) of the feedback tone. A value of 0 means that the user's key down and key up times are followed (sound will occur for a duration equal to the time the key is held down). A non-zero value means that the feedback will sound for that

number of milliseconds. When a non-zero length is used, both the NoteOn and NoteOff output events will be scheduled at time of keypress. [Default: 0]

4. FEED_PMODE: Pitch-mapping types. These must be specified as integers in the parameter file, but are provided here along with their internal designation for convenience (see "params.h"). Note that the SEQ and LAG values may cause awkward interactions if the second feedback channel is being used (see Section 8.3): [Default: 0]

   (a) 0 (RIGHTPITCH): Output pitch value is same as the keystroke input pitch.

   (b) 1 (SAMEPITCH): Output pitch is fixed as FEED_NOTE, regardless of input keystroke value.

   (c) 2 (REVPITCH) : Reversed pitch, low notes at right end of keyboard.

   (d) 3 (LARGEPITCH) : A semi-random (but consistent, i.e., each note is always mapped to the same pitch) pitch mapping. Each note will receive a pitch alteration (in semitones) according to the following list: C:+6, Db:-2, D:+3, Eb:0, E:+15, F:-6, Gb:0, G:+1, Ab:-5, A: +1, Bb:+2, B:0. (See Finney (1997) for an example use of this.)

   (e) 4 (RANDPITCH) : Another semi-random pitch mapping, but one that is not consistent (hitting the same key twice in succession will give different notes). Each played note will receive a semitone alteration randomly selected from the range of a 5th below to a 5th above the input note. NOTE: if FEED_NOTE is set to a non-zero value (as is currently the default), the output notes will range from a 5th below to a 5th above that note, regardless of input keystroke. Setting FEED_NOTE to 0 will cause the output note to randomly vary centered on the keystroke note. If this description makes no sense, try it out!

   (f) 5 (SEQPITCH) : A fixed pitch mapping. Each succeeding keystroke (regardless of key) will play the next note from a fixed sequence, specified by the user in a pitch sequence file (see the SEQPITCH_FILE string parameter). Great for non-musicians, as they can play a melody correctly no matter what keys they hit (see the "bachpitch" file in the "params/demo" directory).

   (g) 7 (LAGPITCH) : Play the pitch from PITCHLAG notes preceding the current note, but with velocity and timing values taken from the current keystroke. At the beginning of a trial, when there are no preceding notes to use, the FEED_NOTE value is used. If silence (or the actual keystroke values) would be preferable in this case, just use keystroke triggers

and the FEED_ON or FEED_PMODE parameter appropriately. (See Pfordresher (2001) and other work by P. Pfordresher and C. Palmer for use of this mode.)

5. FEED_NOTE: If FEED_PMODE is set to 1 (fixed pitch value in response to any keystroke), this value specifies what note to use for feedback (from 1-128). FEED_NOTE also affects behavior if FEED_PMODE is set to 4 (randomized pitch); see above. [Default: 96]

6. FEED_DMODE: Delay mapping types. These must be specified as integers in the parameter file, but are provided here along with their internal designation (see "params.h") for convenience. When a note is delayed, the duration of the note is unaffected, i.e., both the NoteOn and the NoteOff events for a note will be delayed by the same amount. [Default: 0]

   (a) 0 (SYNC_DELAY): No delay; feedback is synchronous with keystrokes.

   (b) 1 (FIXED_DELAY): Fixed delay, all keystrokes delayed by FEED_DVAL milliseconds.

   (c) 2 (RAND_DELAY): Randomly (for each keystroke) select a delay from RANDDELAYS_ARRAY (see Section 4.5).

   (d) 3 (UNIFORM_DELAY): The delay (for each keystroke) is chosen from a uniform distribution between 100 and 300 ms.

7. FEED_DVAL: The amount of delay (in milliseconds) if FEED_DMODE = 1. [Default: 250]

8. PITCHLAG: If FEED_PMODE is 7, PITCHLAG specifies the number of notes back from which the pitch value will be taken. [Default: 0]

9. FEED_VMODE: When set to 0, the velocity value for feedback messages will be the same as the input keystroke. When set to 1, the velocity on output will be fixed, and specified by FEED_VAL. There are also some preliminary velocity mappings: a value of 2 (REV_VEL) will cause harder keystrokes to give a softer sound, while a value of 3 (RAND_VEL) will give randomized loudness to each keystroke. [Default: 0]

10. FEED_VEL: If FEED_VMODE is set to 1, this specifies the fixed MIDI velocity value to use for feedback. [Default: 0]

4.3.2 Second Feedback Channel

A second feedback channel[3] is provided which can either provide a second feedback tone for a given keystroke (e.g., one synchronous tone and one delayed tone), or can provide different feedback

[3]The choice of the term "channel" is perhaps unfortunate, as this is totally independent of MIDI channel.

responses in different parts of the keyboard (such a keyboard split might be interesting for two finger polyrhythmic tapping experiments). Examples are in the "params/demo" directory (the "twochan" and "splitchan" files). The parameter names for the second feedback channel begin with "FEED2"; they behave exactly like the "FEED" equivalents, but have different default values (see Section 4.2). The defaults specify a fixed length, pitch, and velocity tone which is delayed by 250 ms, but the FEED2_ON parameter defaults to 0 (no sound).

Use of the FEED2 channel to provide a second feedback tone to a keystroke is simple; just set the FEED2 parameters to the feedback response you want, and set both FEED_ON and FEED2_ON to 1.

To provide different responses in different parts of the keyboard, it is necessary to use the SPLIT_POINT parameter, and specific settings for the FEED*ON parameters. SPLIT_POINT defines the splitting point on the keyboard (e.g., 60 for C4). If FEED_ON is set to 2, the response for the main feedback channel will occur for keystrokes with note values greater than or equal to the split point, while if FEED_ON is set to 3, the feedback response will occur for keystrokes below the split point. FEED2_ON is set in the same way. So, e.g., set SPLIT_POINT to 60, FEED_ON to 2, and FEED2_ON to 3 so that the FEED_* parameters define the feedback response for keystrokes on the right side of the keyboard, and the FEED2_* parameters define the response on the left side of the keyboard.

There are some limitations on the use of multiple feedback channels; see Section 4.7.2.

4.3.3 Masking Parameters

FTAP provides the ability to output masking noise for the duration of a trial that is, a MIDI note which stays on for the duration of a trial. MASK_ON only takes effect at trial beginning, and would typically be used with a tone generator that generates some approximation to white noise. MIDI channel, note, and velocity can be specified. Trigger events do *not* affect masking noise.

1. MASK_ON [Default: 0]
2. MASK_CHAN [Default: 2]
3. MASK_NOTE [Default: 64]
4. MASK_VEL [Default: 35]

4.3.4 Metronome Parameters

FTAP's metronome provides a flexible form of pacing tone. The integer parameters provide for an isochronous beat, while array parameters (see Section 4.5) can impose different types of structure on the tone sequence. If an array parameter is specified, it will override the corresponding integer parameter. The actual characteristics of the sound will depend on the tone generator used; all FTAP does is make sure that MIDI NoteOn and NoteOff messages go out at the right time.

The metronome parameters can be changed by triggers, altering the metronome behavior on the fly. A metronome trigger will affect the tone produced on the specified beat (but not the MSPB preceding that beat). One use for such triggers would be perturbing a stimulus sequence in a sychronization experiment.

1. METRON_ON: 1 if the metronome beat should be sounded, 0 if not. [Default: 0]
2. MET_CHAN: MIDI channel for the metronome. [Default: 1]
3. MET_NOTE: MIDI note value for the metronome. [Default: 64]
4. MET_VEL: MIDI velocity for the metronome. [Default: 100]
5. MET_LEN: Length (in milliseconds) of the metronome beat. [Default: 20]
6. MSPB: Length (in milliseconds) between beats. [Default: 600]

4.3.5 Miscellaneous Integer Parameters

1. CLICK1_OFFSET, CLICK2_OFFSET: if click files (see below) are being used, this is the offset (in milliseconds) from trial start at which the click file will be played. Up to 2 click files are available. These offset parameters are useful in allowing a single set of pre-programmed events to be used at different positions or "phases". There can be two click files, and each one can be offset by a certain amount. [Default: 0]
2. STDOUT: If set to 1, FTAP output will go to the video screen rather than to a text file. Only keystrokes will be displayed. This may be useful for long practice or test sessions where you don't need the data recorded, as it keeps input data structures where the information is stored from overflowing (FTAP normally does not write any data to disk while a trial is running; it is kept in memory). However, since the internal data structues now hold about 7500 notes (set to 15,000 in "ftaplimits.h", where each note involves a NoteOn and a NoteOff), it really isn't likely to be necessary. No longer supported, but probably works. [Default: 0]

3. FULL_PARAM_PRINT: Print out all parameters in the output file, rather than just the default subset. With this parameter set to 0, the output file header will only include the "obligatory" parameter printout (see Section 4.2) plus any user-specified parameters, and will be about 30 lines long. If FULL_PARAM_PRINT is set to 1, the header will be about 60 lines. If you are debugging a parameter file that doesn't work as you expect, I suggest setting this to 1 so you know exactly what you're getting (I will insist on this if you ask me to troubleshoot a problem). [Default: 0]

4.4 String Parameters

Some parameters take string values; these are typically file names or documentation strings. Double quotes, although used below, should *not* be used in the parameter file itself.

The first 3 of the following parameters are values which identify some aspect of the current experimental run; they are also used to form the output file name

1. SUB: subject id. [Default: "sub"]
2. BLOCK: trial block. [Default: "block"]
3. TRIAL: trial number. [Default: "trial"]
4. COMMENT: A single comment line which will go in the output file. [Default: " "]
5. CLICK1_FILE, CLICK2_FILE: names of up to two pre-existing files of data that will be played during the trial. These files must be in the same format as the FTAP output file (i.e., 8 columns, though not all columns are actually used), and the event type must be specified as 'F'(eedback) or 'M'(etronome) (other event types, including 'K', will be ignored and will cause an error message). Click events will be written to the output file; the 'F' or 'M' specification will be preserved. The easiest way to generate such files may be to either record a performance using FTAP, or generate an output file using the FTAP metronome. Offsets may be specified by the CLICKN_OFFSET parameters. The file names are interpreted relative to the directory containing the parameter file. See params/sample_experiments/finn_99_fixed for an example.

   Note that this is a fairly simplistic playing mechanism that is not really designed for presentation of complex musical stimuli. [Default: " "]
6. PITCHSEQ_FILE: if FEED_PMODE is set to 5, then this file contains a list of integers (1 per line) specifying the sequence of notes to be played in response to successive keystrokes. The

sequence will repeat when the end is reached. See 'params/demo/bachpitch' for an example. The file name is interpreted relative to the directory containing the parameter file. [Default: " "]

4.5 Array Parameters

Array parameters are used when there is a list of (integer) values. The format for an array parameter involves first specifying the number of elements ('count'), and then the elements themselves. The number of elements must be at least as large as 'count', and only the first 'count' elements will be used. All array parameters default to a count of 0. The maximum length of the random delay list is 10, and the maximum length of the metronome cycle is 20.

1. RANDDELAY_ARRAY: Used in conjunction with FEED_DMODE = 2, this allows for a form of random delay. Up to 10 delay values can be specified, and the delay for each keystroke will be randomly selected from the set. The following specification will randomly choose among 100, 200, or 300 ms delays.

   ```
   RANDDELAY_ARRAY        3       100     200    300
   ```

2. MET_PATTERN_ARRAY: This allows specification of a pattern on top of the underlying metronome beat (that is, a pattern of sounded and silent beats). The maximum length is 20 beats. The following line will sound a cycle of 3 beats, followed by a pause (where the rate and tone characteristics will be determined by the integer metronome parameters).

   ```
   MET_PATTERN_ARRAY      4       1       1      1       0
   ```

3. MET_VEL_ARRAY: This allows imposing a loudness (MIDI velocity) pattern on the metronome sequence. The following will make the first of every 4 beats louder than the other 3.

   ```
   MET_VEL_ARRAY          4       110     80     80      80
   ```

4. MET_CHAN_ARRAY: This allows creating a pattern of changing timbres on the metronome sequence by specifying the output MIDI channel for each beat (this assumes a polytimbral tone generator). The following lines will give different sounds for the 1st and 2nd tones of a two beat pattern, depending on how the tone generator is programmed to respond on MIDI channels 1 and 2.

```
MET_CHAN_ARRAY          2       1       2
```

5. MET_LEN_ARRAY: This allows imposing a pattern of tone length on the metronome sequence. The following would place a longer tone (agogic accent) every 3 beats in an isochronous sequence.

```
MET_LEN_ARRAY           3       150     80     80
```

6. MET_NOTE_ARRAY: This allows imposing a pitch pattern on the metronome sequence. The following line will repeat an ascending C-major scale.

```
MET_NOTE_ARRAY          8         60    62    64    65    67    69    71    72
```

If a metronome array parameter is specified, it will override the equivalent integer parameter. If multiple metronome array parameters are used (e.g., both velocity and length), they would typically all be of the same length (as in the following polyrhythm example), but this is not required by FTAP (see, e.g., the "metronpat3" file in the "params/demo" directory).

4.5.1 Polyrhythmic Pacing Example

The following parameters would provide a 3 vs 4 polyrhthmic pacing signal, with the 4 part of the signal distinguished by a higher pitch. Unfortunately, FTAP does not permit playing more than one pitch at the joint accent, so this is played by a third pitch, and stressed by length.[4] Although this example is a little complicated, once it is set up it is possible to change the tempo simply by altering the MSPB parameter.

```
METRON_ON               1
MSPB                    200
MET_VEL                 90
MET_PATTERN_ARRAY       12      1   0   0   1   1   0   1   0   1   1   0   0
MET_NOTE_ARRAY          12      79  0   0   84  72  0   84  0   72  84  0   0
MET_LEN_ARRAY           12      120 40  40  40  40  40  40  40  40  40  40  40
```

[4]It might be possible to do some messy manipulations using triggers and the MSPB parameter (making the time between the first two tones in the cycle effectively simultanous by setting MSPB to 1 millisecond) to simulate multiple (effectively) simultaneous tones. Also, using the MET_CHAN_ARRAY parameter in conjunction with clever programming of a polytimbral and stereo tone generator might allow better handling of the joint beat, as well as possible left ear/right ear separation of the components of the pacing signal.

4.6 Triggers

A trigger event is a metronome beat, keystroke number, or elapsed time which causes an immediate change in an integer-valued parameter[5]. Triggers are specified by the keyword "TRIGGER", followed by 5 obligatory fields. The first field following the word "TRIGGER" is a unique trigger ID (for identification in the output file), followed by the trigger type: K(eystroke), T(ime), or M(etronome). The fourth field is the count for the trigger (elapsed milliseconds for time triggers, keystroke number for keystroke triggers, and metronome count for metronome triggers), and the next two fields are the name of the integer parameter to change and the new value. A special pseudo-parameter "END_EXP" can be used in a trigger specification to terminate an experiment; it is followed by an arbitrary (but obligatory) integer. An unparseable or incorrect trigger specification will print a message to the screen; if your experiment isn't working correctly, check this. The first trigger specification below will turn the metronome off on the 10th beat, the second will cause the 6th (and all following) keystrokes to sound with a delay of 300 ms (assuming FEED_DMODE is set to 1), and the third specification will end the experiment after 15 seconds. Both keystroke and metronome triggers affect the triggering event itself; however, for the metronome the preceding MSPB has already been processed, so a metronome trigger that changes MSPB will affect the *following* metronome pulse.

```
TRIGGER 1 M 10 METRON_ON 0
TRIGGER 7 K 6 FEED_DVAL 300
TRIGGER 3 T 15000 END_EXP 1
```

Metronome triggers are counted based on the underlying beat, whether it is sounded or not, with the first such logical beat starting MSPB milliseconds after trial start, and a logical beat every MSPB milliseconds later. Metronome triggers are counted on this logical beat basis, regardless of the value of METRON_ON or MET_PATTERN_LEN.

4.7 Parameter Subtleties and Complexities

This section discusses some fine points of FTAP parameter usage.

4.7.1 Triggers

A somewhat minor feature of triggers may cause problems for a careless user (such as me). The trigger ID (the second field in the specification) currently serves no purpose other than marking

[5]There are a few special case instances where this will not work, e.g., integer parameters which only have an effect at trial start such as MASK_ON and STDOUT

trigger events in the output data file (i.e., it plays no role in sequencing). However, a second trigger with the same ID will erase a previous one. A warning message will be written if this occurs.

In FTAP version 2.1.03, there was a problem if a trigger which affected a feedback parameter occurred between a keystroke down and the corresponding key release (e.g., if a change of FEED_ON from 1 to 0 occurred, the NoteOff was not sent and the note would not terminate). One suggested workaround was to used fixed length output (FEED_LEN != 0), since in this case NoteOn and NoteOff messages are always scheduled together. This has been fixed in 2.1.05; all relevant feedback parameter values are saved when a key is pressed, and those values are used when that key is released (even if the value has been changed in the meantime). This has not been tested as thoroughly as I would like (particularly with respect to multiple feedback channels), but no one has reported any problems.

4.7.2 Feedback Channels

The second feedback "channel" was originally introduced into FTAP to handle experiments with both synchronous and delayed feedback in response to a single keystroke, including the ability to flexibly manipulate volume levels for each of the two feedback output events. It was then simple enough (from an implementation perspective) to extend this to allow two totally independent responses for a keystroke (e.g., synchronous and pitch mapped feedback combined with delayed and isotonal feedback), and then a further simple step to allow applying one or the other of those two feedback responses to different parts of a logically split keyboard , allowing different responses to different fingers or hands (e.g, one delay value above middle C, a different one below middle C). The resulting parameters are fairly simple, but the additions are not always fully integrated, and contain some pitfalls if not used carefully. Note that feedback "channel" here should not be confused with MIDI channel (which is completely separate); in addition, a feedback "channel" may be polyphonic. In addition, keyboard splitting relates to, but is also somewhat distinct from, the use of multiple feedback channels.

The particular issue is that a number of variables in the system are not factored out into the two channel scheme because it hasn't been worth the extra complexity. For example, there is only one RANDDELAY_ARRAY variable, so it is not (currently) possible to have independent random delays on the two channels. Also, MIDI controller input (e.g., pedals, pitch bend) will always be delayed based on the FEED_DMODE parameter.

The following facilities will work fine if only one feedback channel is configured for the given capability (you can still use the second channel, it just must be programmed with a different mapping),

but will probably give weird results if applied to both feedback channels at the same time. A good rule of thumb is: Unless you know exactly why you need to use the second feedback channel, don't. And if you do use it, try to avoid fancy pitch and delay alterations, and don't rely on use of MIDI controllers.

1. Fixed pitch sequences : FEED_PMODE = 5. The sequence file is shared by the two channels.
2. Pitchlag: FEED_PMODE = 7. Currently, you cannot do independent pitch lag alterations on the two channels.
3. Random pitch mods: FEED_PMODE = 4. ???
4. Random delay: there's only a single random delay array specification (RANDDELAY_ARRAY), though it can be selectively applied to one or both channels.

5 Sample Parameter Files

This section contains examples of some parameter files. More are included in the "params" directory of the distribution.

5.1 demo/keythru

```
# Just output (on MIDI channel 1) whatever the person plays for 10 seconds,
# as in normal music performance. This just demonstrates that you can
# collect performance data without doing any funny stuff. Setting FEED_ON
# to zero would allow investigating performance in the absence of auditory
# feedback.

FEED_ON                 1
FEED_CHAN               1
FEED_PMODE              0
FEED_VMODE              0
FEED_LEN                0

METRON_ON               0

TRIGGER 1 T  10000 END_EXP 1
```

5.2 Modified "keythru"

```
# File for a music experiment in which keystrokes are delayed by 250
# ms and also sound a random pitch, as in Finney (1997). Terminate
# the trial with <ctrl-C>.

FEED_ON                 1
FEED_CHAN               1
FEED_VMODE              0
FEED_LEN                0
FEED_PMODE              4
FEED_DMODE              1
FEED_DVAL               250

METRON_ON               0
```

5.3 sample_experiments/aschersleben

```
# Synchronization task with an 800 ms isochronous pacing signal and 70 ms
# delayed auditory feedback to the subjects keystrokes. The feedback tones
# are of fixed pitch, volume, and length, regardless of the subject's
# keystrokes. Ends after 36 pacing tones.  Similar to Aschersleben and
# Prinz (1997).

# Change FEED_DVAL to 0, 30, or 50 for the other delay conditions in
# their experiment. To cover up synchronous physical keystroke noise,
# set MASK_ON to 1 (this assumes you have an appropriate masking noise
# programmed on MIDI channel 2). To investigate effects of tempo on
# the amount of negative asynchrony (tapping before the metronome
# pulse), simply alter MSPB.

FEED_ON         1
FEED_CHAN       1
FEED_NOTE       90
FEED_VMODE      1
FEED_VEL        127
FEED_LEN        20
FEED_PMODE      1
FEED_DMODE      1
FEED_DVAL       70

MASK_ON         0
MASK_CHAN       2
MASK_NOTE       64
MASK_VEL        30

METRON_ON       1
MSPB            800
MET_CHAN        1
MET_NOTE        60
MET_VEL         127
MET_LEN         20


TRIGGER 3 M  37 END_EXP 0
```

5.4 sample_experiments/finn_99_275

```
# Sample  experiment from Finney (1999). This is a continuation
# paradigm  with a 4-2 patterned pacing tone.  This trial gives
# auditory feedback to keystrokes (with  275 milliseconds delay) only
# during the continuation phase (not while the pacing signal is on). Typical
# subject response is to sometimes tap groups of 5 rather than 4.

FEED_ON         0
FEED_CHAN       1
FEED_NOTE       76
FEED_VMODE      1
FEED_VEL        90
FEED_LEN        100
FEED_PMODE      1
FEED_DMODE      1
FEED_DVAL       275


METRON_ON      1
MSPB           250
MET_CHAN       1
MET_NOTE       86
MET_VEL        100
MET_LEN        30
MET_PATTERN_ARRAY   8 1 1 1 1 0 1 1 0


TRIGGER 1 M  32 FEED_ON 1
TRIGGER 2 M  32 METRON_ON 0
TRIGGER 3 M  92 END_EXP 0
```

5.5 Split keyboard

```
# Randomized  delay to keystrokes on right side of keyboard, no delay to
# keystrokes on left side of keyboard. Terminate the trial with <ctrl-C>.

FEED_ON                 3
FEED_CHAN               1
FEED_VMODE              0
FEED_LEN                0

FEED2_ON                2
FEED2_CHAN              1
FEED2_VMODE             1
FEED2_VEL               100
FEED2_LEN               100
FEED2_PMODE             0
FEED2_DMODE             2
RANDDELAY_ARRAY 6 100 150 200 250 300 350

SPLIT_POINT             60
```

6 Important Program Limits

FTAP has a number of hard-wired limits; if these are a problem, you can change the numbers in the source and recompile (see "src/ftaplimits.h"). The one which is most likely to be a problem is the number of stored events: FTAP does not do any disk I/O while a trial is running (to avoid the timing overhead). All data (keystroke, metronome, and feedback events) is stored in memory until the end of a trial. The space for this is allocated statically to avoid any overhead of memory allocation; a maximum of approximately 7500 notes can be stored with the default allocation.[6]

1. MAXNOTES = 15000: Maximum number of stored events (with separate allocations for input (keystroke) and output (feedback and metronome) events). Note that NoteOn and NoteOff are separate events. This limit (approximately 7500 notes) might be reached in a long musical performance. It's possible that on a machine with a decent amount of memory this could be increased to a much larger value without consequence; it is also possible that this default might be too large for some configurations.

2. MAX_TRIG_EVENTS = 60: Maximum number of trigger events. Should be enough for most conceivable purposes.

[6] A possible solution would be to add a new parameter to allow a user to specify a default (larger) allocation that is dynamically allocated by FTAP (and locked in memory) before the trial timer starts.

3. MAXPATTERN = 20: Maximum length of metronome pattern. Should be enough for most purposes.

4. MAXRDELAYS = 10: Maximum number of random delays. Should be enough for most conceivable purposes.

There is also a limit on the number of "active" events (that is, events in the scheduling queue) which is MAX_TE_EVENTS (currently set at 2000). This is unlikely to be a problem for FTAP itself, but *is* a problem for the "playftap" program which plays FTAP output files (see Appendix I). "playftap" is compiled with a larger limit for this value.

7 Output File Format

The output file name is hardwired to the format

```
<paramfilename>.<sub>.<block>.<trial>.abs
```

where sub, block, and trial are specified as parameters (the SUB, BLOCK, and TRIAL parameters default to the strings "sub", "block", and "trial" if not explicitly specified). The suffix ".abs" is around for historical reasons; it represents the fact that time values in the output file are absolute, rather than relative to the preceding event.[7] Since the primary use of the output is for data analysis, the output is in a columnar ASCII format rather than, say, a MIDI file (though the latter could be created from FTAP's output, with some loss of information).

The output file not only provides a full listing of all input and output events, but also provides information about the active FTAP parameters, the functioning of FTAP, and timing diagnostics. My approach has been that the output files should be self-explanatory; thus, all characteristics of the trial (the parameters) plus diagnostic information and date should be included. Unnecessary information can be removed (using, for example, PERL, Python, or |STAT), but unrecorded information can never be recovered; too much information is better than too little. However, this philosophy has been compromised a bit because the header was getting long and unwieldy. The current approach is that all parameters explicitly specified by the user are printed, as well as a subset of others that I consider to be crucial (see annotations in Section 4.2). To print out *all* parameters (recommended, e.g., if you don't understand why FTAP is behaving a certain way), set the FULL_PRINT_PARAM parameter to 1.

[7] This should probably be changed to ".ftap", but I have code and shell scripts which expect the former. The file naming format is kind of ugly, but it forces the file name to usefully identify the experiment conditions.

If the STDOUT param is set to 1, a data file is not written; see description of the STDOUT parameter. This facility is no longer supported, but probably works.

7.1 Comment Lines, Parameters, and Output Diagnostics

The data file starts with a listing of informational non-data lines. This header cannot be turned off, though there is some control over just how much gets printed via the FULL_PARAM_PRINT parameter. These non-data lines (e.g., the parameter listings, and output diagnostics) are always preceded by "# "; the marking of all these lines with "# " in the first column makes them easy to filter out. The parameter values are those at the start of the experiment, before any trigger changes.

The following output-only information values are listed along with the actual experiment parameter values:

1. TIME: Time and date that this run of FTAP started.
2. VERSION_NUMBER: FTAP version number, currently 2.1.05.
3. PARAMETER_FILE: Name of the parameter file.

Important Diagnostic Parameters (see Section 9.6 for more details). These can be ignored unless you're doing real-time data collection.:

1. SCHED_AV: Average time (in ms) between calls to the output scheduler. Less than or equal to 1 is good. More than 1 indicates a problem.
2. SCHED_MAX: The *maximum* time (in ms) between output scheduling calls. More than 3 or 4 ms is probably bad. Running FTAP as "root" and disabling networking and X Windows will probably improve things.
3. SCHED_GT1, SCHED_GT5, SCHED_GT10: Number of times that calls to the internal scheduling routine were separated by times more than 1, 5, or 10 ms respectively. Any time more than 5 indicates a problem (and there should be no more than a small number greater than 1).
4. SCHED_MAX_TIME: time of occurrence of the maximum scheduling discrepancy (SCHED_MAX). (ms from trial start).

On my system, trial runs have typically had no more than one or two scheduling times greater than 1 ms. Note that even if there are occasional long scheduling times, the data is not necessarily corrupt, as the scheduling glitch may have occurred at a time when there was nothing to do anyway!

The following secondary diagnostic variables allow looking at this. They are not included in the default printout, and may all eventually be removed.

Secondary Diagnostic Parameters:

1. AV_DELAY: The overall average delay, a possibly useful diagnostic when using random delays. Probably bogus if multiple feedback channels are in use.

2. DISC_AV: output events are scheduled to be written at a certain time, but the time they're actually written may be different if something has gone wrong. The value is the average descrepancy between actual and correct time. This can be checked in the output file, as the *actual* (not intended) time of the write() call to "/dev/midi" is provided there.

3. DISC_MAX: Maximum value of such discrepancies.

4. DISC_MAX_TIME: The time (since beginning of trial) where this discrepancy occurred. The offending event can then be looked up in the output file.

5. INPUT_MAX: in a multi-byte MIDI message, the maximum time between FTAP's accessing of the individual bytes.

6. INPUT_MAX_TIME: The time where INPUT_MAX occured.

MIDI Error Diagnostics: On one system, I would occasionally (about 1 event out of 15,000) see what appeared to be hardware MIDI errors. FTAP currently detects some obvious examples (a MIDI Note Number of 0, or larger than 127, or a MIDI velocity greater than 127), and will print a "MIDI WARNING" diagnostic to the screen, and will list information about the detected error (for up to 10 errors) in the output file, with a "parameter" of MIDI_ERROR. If you understand the MIDI data stream, how FTAP handles MIDI input and output, and the details of the MIDI data that your keyboard puts out (see Appendix D) it is usually possible to reconstruct the correct data, but it requires a bit of detective work. It is too much detail to put here (see also Section 7.6).

7.2 Data Line Format

Any line in the file not starting with the "# " character (or white space) is an 8 column data line, arranged in ascending order by time of occurrence (indication of trigger actions is included in these data lines). The data line format was originally designed for MIDI note events, such as keystrokes, metronome, and feedback. However, the data file also contains output lines for controller events and

lines showing when trigger events occurred; these both require different types of information. The output format for these events retains the same column layout as that for MIDI note events, but reuse of the fields has resulting in a somewhat inelegant mix of formatting. However, it should be easy to extract whatever information is of interest.

Data lines contain 8 columns. Column 1 always indicates the elapsed millisecond time of the recorded event relative to trial start, and column 8 always identifies the event type. The event type codes are:

Note data:

1. 'K': Keystroke (both NoteOn and NoteOff, that is, key press or release).
2. 'F': Feedback (sound in response to a keystroke. May be delayed or mapped from the keystroke itself).
3. 'M': Metronome event.
4. 'J': "Junk" event. Currently only applies to masking noise.

Other:

1. 'C': MIDI controller.
2. 'G': Output/feedback for a MIDI controller (not mnemonic, simply the letter after 'F').
3. 'T': Trigger event.

7.3 Note Data

If column 8 is 'K', 'F', 'M', or 'J', columns 1-7 have the following contents:

1. Col 1: Milliseconds since trial start.
2. Col 2: 'D'(own) for key press, 'U'(p) for key release.
3. Col 3: MIDI channel of the event.
4. Col 4: MIDI Note number of the event.
5. Col 5: Pitch name for the MIDI note number of column 4 (e.g. C#4 for 61). "#" is always used for accidentals.

6. Col 6: MIDI velocity (0 for key release; any velocity the input device may produce for NoteOff events is discarded).

7. Col 7: Sequence number for input key presses, as well as for corresponding feedback events.

8. Col 8: Event type = 'M', 'F', 'K', 'J'.

Example (a key press data line) :

```
9630    D       1       86      D5      74      12      K
```

7.4 MIDI Controller Data

If column 8 is 'C' or 'G', columns 1-7 should be interpreted as follows; understanding some of this requires knowledge of MIDI details.

1. Col 1: Milliseconds since trial start.

2. Col 2: 'X' (meaningless filler)

3. Col 3: MIDI channel of the event (?).

4. Col 4: First byte of MIDI controller data (in decimal). The interpretation depends on column 5. If column 5 is "B0" (MIDI Control Change message), then column 4 will indicate the specific controller (e.g., 64 for a sustain pedal). If column 5 is E0 (Pitch Bend), then column 4 is the least significant byte of the pitch bend value.

5. Col 5: The MIDI status byte (nibble) (in hex) of the input controller message. A0 for Polyphonic Key Pressure, B0 for MIDI Control Change, E0 for Pitch Wheel Change, ...

6. Col 6: Second byte of MIDI controller data. For Control Change messages, this will be the value for the specified controller. For Pitch Bend, it will be the most significant byte of the pitchbend value.

7. Col 7: 0 (filler).

8. Column 8: Event type: 'C' for an input event, 'G' for an output event.

Example: A sustain pedal being pressed and released might be recorded as follows:

```
8000    X       1       64      B0      127     0       C
9000    X       1       64      B0      0       0       C
```

7.5 Trigger Events

If column 8 is 'T', columns 1-7 should be interpreted as follows:

1. Col 1: Milliseconds since trial start.
2. Col 2: Trigger subtype: M(etronome), T(rigger), K(eystroke).
3. Col 3: 0 (filler).
4. Col 4: Trigger ID (second field of the TRIGGER parameter line).
5. Col 5: "- -" (filler)
6. Col 6: Internal index; ignore.
7. Col 7: 0 (filler).
8. Col 8: Event type: 'T'.

Example: A metronome trigger:

```
TRIGGER 1 M 32 FEED_ON 1
```

Corresponding output line (metronone at 250 ms MSPB):

```
8000    M       0       1       --      0       0       T
```

7.6 MIDI Hardware Errors

One of our hardware setups has gotten occasional MIDI hardware errors (e.g., a note value of 0); this occurred about 1 time out of every 15,000 input events. There is now some code in "linux_midi.c" (see the *ReadMidiEvent* routine) which detects some obvious cases of errors (e.g., a note error of 0), and a diagnostic is printed to the screen. In addition, a "MIDI_ERROR" entry is made in the output file, immediately before the data lines (a maximum of 10 errors is reported); I suggest always checking your output files for the presence of this string. An example is the following, where the first field is the time of the detected error, and the other fields are bytes of the MIDI data stream (these are not necessarily the precise raw bytes, e.g., the input routine has already interpolated MIDI running status). The velocity of 255 is bogus.

```
# MIDI_ERROR    54675   0x80    19      255
```

If you are getting MIDI errors, there's not much I can say. I was usually able to recover (by hand) what the original data should have been, but this requires a certain amount of detective work, and reasonable knowledge of the MIDI data stream and aspects of your keyboard (it is particularly difficult because FTAP does not retain an exact copy of the input stream). It should also be noted that only certain obvious MIDI errors are detected (although this accounts for about 3/4 of the situations I had problems with). Some useful information is in Appendix D.

7.7 Data Filtering And Transformations

The potentially extensive data in the output file can be filtered for analysis as necessary, though I recommend keeping the original .abs files around as a complete raw data record. It's simple to filter these events using something like the "dm" program of Gary Perlman's |*Stat* (//www.acm.org/ perlman/statinfo.html) or a shell script, or a Python or Perl program. For instance, if the only data of interest are the keystroke down times, remove any lines with the comment character ("#") in column 1, and then extract all lines with 'K' in column 8 and 'D' in column 2. The data values in column 1 of the resulting file will be the keypress times.

8 Troubleshooting

This section contains some hints if you can't get FTAP to work. Some things have certainly been overlooked in this Reference Manual; notify me if you find such things, and if you're motivated you can always look at the provided source code.

8.1 FTAP isn't working at all!

If you can't get FTAP to do anything at all, here are some suggestions.

1. Make sure your Linux MIDI system is configured correctly for FTAP use. FTAP does all MIDI I/O using "/dev/midi". A simple setup involves a keyboard attached to MIDI In and a separate tone generator attached to MIDI Out; the only sound output should be from the tone generator. When you tap on the music keyboard when FTAP is not running, you should hear no sound (FTAP wants to control all auditory feedback).[8] However, if you run the "midicopy" program in the 'bin' directory of the distribution (source is in the"utils/midicopy" directory), then your keyboard should create sounds through the tone generator. If "midicopy" isn't working, you

[8]Some MIDI/sound cards (e.g., the Roland MPU-401) default to echoing MIDI in back to MIDI out, so this is not a definitive test.

need to sort out your MIDI setup (e.g., check the MIDI channel settings on the devices you're using).

If "midicopy" works, then your basic MIDI setup should be OK. If FTAP nonethless seems to not be working, check the MIDI channels in whatever parameter file you're running, and look at the output file to see whether or not input is being recorded.

8.2 FTAP isn't doing what I think it should!

If you've created a parameter file, and it's doing something but not what you think it should be doing, here are some suggestions (in approximate order). See also Section 4.7:

1. Look at your input file, and make sure you understand what each line is supposed to do. Remove lines that are not relevant for your particular experiment. Make sure that any parameters that are important to your experiment are explicitly specified rather than using defaults.

2. Check the output file; everything FTAP has done is in it. Make sure you understand it.

3. Turn on the FULL_PARAM_PRINT param so the output file will have a full listing of all the active parameter values for your experiment.

4. Reread this Reference Manual and the User's Manual to see that you haven't missed something.

5. Look at the C source and/or contact me.

8.3 Quirks/Bugs/Features

Some quirks of FTAP are described here. These are placed here to aid in troubleshooting. You can tell me which one(s) are intolerable for future releases.

1. FTAP will print out a diagnostic on any MIDI message it can't parse, such as SYSEX messages (it used to abort). FTAP handles a lot of different MIDI messages (although it ignores irrelevant ones), but often when I try new keyboards I find something new that FTAP can't handle. FTAP does not have a smart MIDI parser and it is not clever about dealing with and recovering from input errors. See Appendix D for some discussion of MIDI.

2. MIDI controller input is subject to the FEED_ON, FEED_DMODE, and FEED_CHAN manipulations before output, but it is not subject to other manipulations.

3. File names used for input (sequence and click files) are always interpreted relative to the directory the parameter file is contained in.

4. For some (most?) tone generators, having multiple simultanous events of the same pitch may cause some notes to get cut off; this might occur if doing something like using a SAMEPITCH pitch mapping (in which all keystrokes get feedback with the same pitch) in an experiment in which subjects are using multiple fingers. If this is a problem, experiment with different tone generators and programming; play around with scale tuning to set all notes to the same pitch if you can do that on your tone generator, etc.

5. If "click files" are being used to play output during an experiment, and if there are multiple (3 or more) precisely simultaneous output events, the timing diagnostics may consistently show some inter-scheduler times of 2 or 3 milliseconds. Although I have not isolated the exact source of this, my guess is that it is due to the use of a (kernel) busy loop within the 4Front driver. It would be interesting to see whether the problem occurs with the ALSA drivers, which supposedly do not use a busy loop.

6. Forcing the file name to be in a particular format (with a ".abs" suffix) is somewhat ugly, inflexible, and un-Linux-like.

9 Installation: Millisecond-resolution Data Collection

FTAP can do data collection with reliable millisecond resolution (see Finney, 2001b, which is recommended/required reading), but there are a number of configuration issues involved in doing this. Such "real-time" use makes use of facilities requiring root privileges. In addition, not all MIDI card/Linux driver combinations are able to meet this criterion. FTAP should work fine when run as a normal user, but if you're doing serious data collection there are some additional concerns.

9.1 Root Privileges

When FTAP is being used for real-time data collection, it will use the *sched_setscheduler* and *mlockall* system calls, as well as the "/dev/rtc" real time clock, all of which require root privileges. If root privileges are not available, FTAP should run just fine, except for some small timing inaccuracies. However, real time clock support must still be enabled (for now). FTAP will print out either "Running with superuser privileges" or "Running as normal user", depending upon which is the case.

The recommended installation for running "ftap" for real-time data collection is to configure it as a setuid root program; configuring the program this way requires superuser privileges. Once this is done, any user can run FTAP as a real-time process. The "installftap" script in the "bin"

directory shows how I do this, but you and/or your system administrator should understand exactly what this script does before trying to do this. NOTE: "setuid root" programs on a Linux machine are a potential source of breached security. I don't think there are any major security issues with the FTAP program (though I am not a security expert); e.g., it is not possible to invoke a shell or execute another program from FTAP, and although FTAP may gobble system resources while it's running it is explicitly designed to be interruptible from the keyboard. If you're running on your personal machine (and it is not subject to hacker attacks), there should be no worries. If you're in a potentially hostile environment, check with your system administrator or technical support person.

This is one reason why FTAP is distributed in source code form. "setuid root" privileges should only be given to a trusted program. If you are going to run FTAP with root privileges, then I recommend that you compile it from scratch (just run the "makeftap" script in the "src" directory, though see Section C.4). Anyone who installs an unknown binary with setuid root privileges is asking for trouble. (If I really wanted to be secure, there should be some sort of security checksum on the binary and source, but I don't think FTAP is going to get the wide use of a program like *sendmail* :-) .)

FTAP requires that the "/dev/rtc" real-time clock be configured; this may or may not be the default on your system (it appears to be on RedHat, but not on Mandrake; see Section C). (This really should only be required if you're running as "root", but currently FTAP won't execute without it.) RTC is used (perhaps counterintuitively) to insert short *pauses* while FTAP is running. To prevent FTAP from hogging system resources when run with root privileges, FTAP contains a .49 ms delay every cycle (this, among other things, allows for terminating FTAP with a keyboard interrupt (DEL or ctrl-C). If run without root privileges, FTAP will run perfectly adequately for test and development purposes, and might even be adequate for collecting data; run it both ways, look at the output diagnostics (described later) and see for yourself. Counterintuitively, on the basis of the output diagnostics FTAP may appear to run *faster* when run without root privileges, as in this case there is no forced delay in each loop (it's not necessary). Crucially, however, running with root privileges avoids preemption (that is, the worst-case performance is superior).

9.2 General Concerns

Because of the real time coding in FTAP, system load is less of a problem than it might be in other cases, but data collection should still be done on a system with no other users logged in (they probably wouldn't be happy with system performance anyway), and with no heavy networking activity (e.g., not acting as a server). I have found some FTAP performance degradation when X Windows is

running (and I suggest running from a console screen), but I don't bother taking the system down to single user mode. Check the diagnostics and see what works for you.

9.3 Hardware and Linux Configuration Issues

[This section is not totally up to date, and it partially duplicates information in Section C, but I'm not bothering to fix it just now. Read that section as well.]

FTAP has been successfully run on Pentium-based hardware (200 MHz) running Redhat Linux 5.2, 6.1, and 6.2 (Linux 2.0 and 2.2 kernels), with a Creative SoundBlaster 16 MIDI card (FTAP does not depend on any particular card, but there are cards that *don't* work; see the following section). I use the 4Front MIDI driver (http://www.4front-tech.com) rather than the standard Linux drivers because they have better time resolution for MIDI output (see discussion in Section 9.3.1); and in fact, the standard drivers currently will not work with FTAP (see Section C). [This section used to say "I suggest you just experiment around with FTAP on whatever driver setup you have until you decide that FTAP will be useful for you; at that point, run the benchmarks in Section 9.6, and consider changing to the 4Front or ALSA drivers if there's a timing problem. Do not rely on precise timing for important data collection on your system until you have run these tests."]

I recommend using a separate keyboard and tone generator, with the sound amplication equipment and headphones only plugged into the tone generator. I have not tried using an internal computer sound card as the tone generator; it may (or may not) work. FTAP needs full control over feedback generation, so the keyboard should not directly control the tone generator. On some keyboards it is possible to disable direct communication between the keyboard proper and the keyboard's tone generator and treat the tone generator as logically separate from the keyboard (e.g., by turning off MIDI LOCAL mode), so you can try this if you know your equipment.

9.3.1 Car(d)s and Drivers

Of the limited hardware I've tested, my best results were with a standard, old-fashioned Creative SoundBlaster 16 card (a genuine one, not a clone). A Roland MPU-401 card (using the UART mode which allows direct access to the MIDI data stream) does not appear to handle the high throughput which I use as a benchmark test, and I never got an ES-1371 to work correctly (though I'm told that it should). Since all I/O is at a very low level (raw byte reads and writes on "/dev/midi"), it should also be possible to use a MIDI interface based on a serial or parallel port.

The standard OSS/Free drivers supplied in Linux (at least the RedHat versions I've worked with, and probably the others) have a 10 ms latency for MIDI output, at least when used with cards

without interrupt capability, such as the SoundBlaster 16. That is, the driver polls for pending MIDI output every 10 ms (1 'jiffie'), and writes out all pending I/O at that time (see the midi_poll routine in midibuf.c in the driver). Althought this is probably harmless or adequate for most music applications, it is not adequate for real-time experimental software. Section 9.6 describes a method of determining whether this is a problem on your system. The commercial 4Front driver does not suffer from this problem, though it is only available in binary form ($20 from www.4front-tech.com); running the benchmark described in Section 9.6 shows that the 4front driver will run with millisecond precision (that is, at the maximal throughput of MIDI), though it apparently uses busy loops. The free ALSA drivers (www.alsa-project.com) are also supposed to be free of this problem, though I haven't tried them.

9.4 Tested Card/Driver List

[See also Section C]

This section lists the card/driver tests that I have tested with the loop test in Section 9.6. Please let me know if you have other combinations that work so I can expand this list!

9.4.1 Success

1. SB-16 with 4Front Driver.
2. Soundblaster Live! Value with 4Front Driver.

9.4.2 Failure

1. SB-16 with standard Linux driver: only 10 ms granularity on output.
2. Roland MPU-401: time out errors.

9.5 On-line Diagnostics

When FTAP terminates, it will print out timing diagnostics, such as the following:

```
Mean time between sched()'s (ms): 0.01, schedcnt: 194077
> 1 ms: 1, > 5 ms: 0, > 10 ms: 0, max: 2 ms
```

If there are many times > 1 ms, or the maximum is greater than 3 ms or so, then your system configuration is not achieving adequate real-time. If you are trying to collect data, make sure FTAP is running as setuid root, and try to make sure your Linux system is running with as few processes as possible.

NOTE: if "click files" are being used to play output during an experiment, and if there are multiple (3 or more) precisely simultaneous output events, the above diagnostics will consistently show some inter-sched() times of 2 or 3 milliseconds. Although I have not isolated the exact source of this, my hypothesis is that it is due to the apparent use of a busy loop within the 4Front driver. It would be interesting to see whether the problem occurs with the ALSA drivers, which supposedly do not use a busy loop.

See Section E for some details on implementation that may help clarify these diagnostics.

9.6 MIDI Throughput Benchmarking

(See Finney, 2001b, for more detail.)

The above diagnostics only test the internal scheduling of FTAP; they cannot test the speed and functionality of the MIDI interface hardware and software. MIDI hardware bandwidth is approximately 30 Kbytes; at 3 bytes for each note on or note off message, this gives a maximal resolution of approximately 1 keystroke event per millisecond, which is adequate for most human behavior. I consider FTAP to have adequate performance if it can match this millisecond resolution, and it does so on my system. However, not all hardware or drivers can handle the maximal bandwidth. This section describes how to check this on your system.

Once you have a working MIDI setup (e.g., you can play from a keyboard with the "midicopy" program), and you can run simple FTAP parameter files, you may wish to verify FTAP MIDI throughput. My approach to doing this requires a cable which can connect the MIDI output port back to the MIDI input port. If your computer MIDI interface provides female MIDI connectors, the required loop can be set up with a standard MIDI cable. If the MIDI interface provides male connectors (as with most joystick MIDI interfaces), then you will need a more specialized cable or connector with female MIDI connectors (5 pin DIN) at both ends, with pin 2 connected to pin 2 (ground), pin 4 connected to pin 4, and pin 5 connected to pin 5.

Such a loop allows an FTAP feedback event to be immediately interpreted as a keystroke event. When FTAP runs the "params/benchmark/looptest" file, and the system is configured with such a cable, FTAP will write a single output priming note (both NoteOn and NoteOff), and then cycle this message in and out 2000 times. On a well performing system, this should take a total elapsed time of 4 seconds (4000 events, 1 ms/event), though there is also some startup overhead. If your Linux driver has the 10 ms polling problem mentioned in Section 9.3.1, the elapsed time will be closer to 40 seconds. Most importantly, just look at the data in the 'looptest.sub.block.trial.abs" output file; this will allow you to see the exact time course of the messages in the system.

If this benchmark appears to show a problem, and you wish to verify whether it is an FTAP problem or a MIDI interface problem, the "rawmidiloop" program in the utils directory might be of some help (this uses the same loop cable configuration, but directly reads and writes the MIDI interface without involving FTAP). This used to work, but I don't know if it still does.

References

Aschersleben, G. and Prinz, W. (1997). Delayed auditory feedback in synchronization. *Journal of Motor Behavior*, 29:35–46.

Finney, S. A. (1997). Auditory feedback and musical keyboard performance. *Music Perception*, 15:153–174.

Finney, S. A. (1999). *Disruptive Effects of Delayed Auditory Feedback on Motor Sequencing*. PhD thesis, Brown University.

Finney, S. A. (2001a). FTAP: A Linux-based program for tapping and music experiments. *Behavior Research Methods, Instruments, and Computers*, 33:63–72.

Finney, S. A. (2001b). Real-time data collection in Linux: A case study. *Behavior Research Methods, Instruments, and Computers*, 33:167–173.

Lutz, M. and Ascher, D. (1999). *Learning Python*. O'Reilly, Sebastopol.

Pfordresher, P. Q. (2001). *Auditory Feedback in Music Performance: Serial Order and Relative Timing*. PhD thesis, Ohio State University.

A Revision History

1. 2.1.00: Initial Linux version.

2. 2.1.01: Better (though still imperfect) handling of MIDI input errors; "WARNING" messages printed to screen (and "MIDI_ERROR" diagnostics to file) in case of unrecognized or incorrect MIDI bytes. Also print out the output file name when FTAP is run.

3. 2.1.02: Internal use only.

4. 2.1.03: The first public release. Changed MIDI parsing code to ignore all MIDI System Real-Time Messages (in particular, the Timing Clock message put out by some Yamaha PortaSound

keyboards). Improved (and simplified!) 'midicopy' utility. Included the *getttimeofday* benchmarks from Finney (in press) in the distribution ("utils/gettod_test"). Major revisions to this Reference Manual.

5. 2.1.04: Internal use only.

6. 2.1.05: 1 major bug fix: triggers that change a feedback response parameter between key press and key release now work correctly, fixing a bug where a note might not be terminated (this was documented in Section 4.7.1 of the 2.1.03 Reference Manual). The fix involves saving *all* parameters on key press, and using those values on key release. Also fixed a minor bug involving MIDI controllers. Minor revision to User's Guide, significant revision to this reference manual.

Future revisions wish list:

1. Fix it so FTAP can run for evaluation purposes using the standard OSS Linux drivers (i.e., support NON_BLOCK as well as select()). Also, allow non-realtime use if RTC is not configured; currently it won't run at all.

2. Add a feature so that a tone (PULSE) can be sounded on the basis of a trigger. More generally, allow calling any C function on the basis of a trigger.

3. Test and expand MIDI capabilities: test ALSA drivers, test use of internal MIDI synth for output, test MIDI on a serial port.

4. Outboard the specifications of *all* pitch mappings to an external file.

B Distribution

The distribution is a directory hierarchy (distributed in gzipped tar format) which contains the following directories:

B.1 bin

The "bin" subdirectory contains a compiled version of FTAP for a Pentium-based computer. This has been compiled under the RedHat 6.1 (or maybe 7.0) Linux distribution with the included GNU C compiler. The bin directory also contains a few other potentially useful programs (with sources in the "utils" directory).

B.2 doc

The "doc" subdirectory contains LaTeX, .dvi, .ps, and .pdf versions of the User's Guide and this Reference Manual. It also contains an "scip_summary.text" file, which is the summary of a presentation I gave at the Society for Computers in Psychology conference in 2000, which describes aspects of the "real-time" implementation of FTAP.

B.3 params

The "params" subdirectory contains sample parameter files that you can run. Each has a comment explaining its purpose. There are 3 subdirectories:

1. demo: parameter files demonstrating many of FTAP's capabilities.
2. sample_experiments: parameter files providing (approximate) replications of a number of studies in the tapping literature.
3. benchmark: parameter files for testing FTAP's performance (see Section 9.6).

B.4 src

The "src" subdirectory contains the C source code for FTAP itself. Currently, there is not a makefile, but rather a "makeftap" shell script which compiles the source. Eventually, the included (system) .h files from my Linux distribution should also go here, so that you can see what differences there are in case you have trouble compiling.

B.5 utils

The "utils" subdirectory contains the source for a few C utility programs, as well as a copy of the Python script in Section F. These programs have not been tested recently, but should still work. "playftap" is a useful utility for playing FTAP output files; see Section I for details. "midicopy" is a simple but useful diagnostic/test program mentioned in Section 9.6. "rawmidiloop" is a less useful (and perhaps non-functionaing) diagnostic program. gettod_test contains a version of the Linux *gettimeofday* tests reported in Finney (2001).

C Linux OS Configuration Issues

This section contains some minimal notes about Linux OS configuration. If you have any more information to add, please contact me.

C.1 OS Versions

FTAP has been run with various RedHat releases, including 2.0 and 2.2 kernels. It has also been run with Mandrake 8.0, using a 2.4 kernel. RedHat appears to configure the real-time clock by default ("/dev/rtc" and supporting kernel/module code). Mandrake appears not to; "insmod rtc" on Mandrake should be sufficient for installing RTC.

Although the real-time clock is needed by FTAP when run for real-time data collection (see Finney, 2001b), in an ideal world it wouldn't be necessary for evaluation use. However, currently FTAP won't run without it. This may change in the future.

C.2 Drivers

As noted elsewhere, FTAP usage currently requires the 4Front MIDI driver to run (http://www.4front-tech.com) (the ALSA drivers might work, but I haven't tried them). Here are the gory details: FTAP is implemented as a single process, and it must be able to check MIDI input without blocking (this was a somewhat arbitrary implementation decision; an alternate approach would be to have a separate input thread and output thread). My initial Linux port used the stock OSS-Free drivers supplied with RedHat and a basic SB-16 card, and did non-blocking reads using the O_NONBLOCK (O_NDELAY) flag. However, the loop test (and follow up investigation) showed that output occurred with a 10 ms poll (based on jiffies and the HZ variable), which was not sufficient for my desired "real-time" (millisecond resolution) performance (it is possible that other interrupt-driven MIDI cards would give adequate performance with the OSS-Free drivers). I then tried the commercial 4Front driver (available only as a binary). Unfortunately, at the time (this may have changed) the 4Front MIDI driver did not support NONBLOCK mode, but it *did* support the select() system call, which can accomplish the same thing. I recoded FTAP to use select(), and the tested performance showed satisfactory millisecond resolution on output.

Much to my chagrin, I didn't realize until much later that FTAP no longer worked with the standard Linux OSS drivers, even for evaluation purposes. This is probably because that driver doesn't support select(), though I haven't chased this down for sure. In fact, *both* select() and NONBLOCK really should (in my opinion) be supported on any MIDI driver. Hopefully ALSA does this, and 4Front may be incorporating NONBLOCK in a future release; I suspect the stock Linux drivers will not be fixed. In the meantime, FTAP has not been recoded so that it works with either select() or NONBLOCK.

C.3 Hardware and MIDI cards

FTAP should run on any reasonable Intel-based Linux machine (let me know if you try it on another architecture, such as Alpha or PowerPC). The two MIDI cards that have been successfully tried are a Creative SB-16 (old and simple) and a more recent Soundblaster Live! Value Digital card. In fact, most any card (except for a Roland MPU-401) is likely to work, but you really should verify MIDI throughput using the tests described earlier (Section 9.6).

C.4 Compilation

FTAP compiles fine under various RedHat releases (not surprising, since it's what I use). It does not compile on the Mandrake 8.0 release. This appears to be due to some small discrepancies involving include files. This could probably be fixed easily; I haven't done so yet.

D MIDI

FTAP does its own parsing of MIDI input, and thus has to be prepared for whatever a keyboard generates (and this differs across keyboards; MIDI allows a number of options). If you're having problems some of this information may be useful to you. This section will also attempt to document just how FTAP manipulates MIDI input and output. This section will not be a MIDI tutorial; see "http://www.borg.com/˜ jglatt/tech/midispec.htm" for an excellent technical overview of MIDI.

D.1 MIDI Data Stream

The basic data processed by FTAP are key press/release events, but there are a number of variants in how these may be encoded. In addition, keyboards may output a number of other types of MIDI events.

MIDI Status bytes are unambiguously differentiated from data bytes by having their high bit set. Key press (NoteOn) events are indicated by a 0x90 MIDI status. There are two allowable ways of signalling a NoteOff (key release) event: either a MIDI NoteOff event (0x80 MIDI status, possibly with a non-zero velocity value), or a MIDI NoteOn event (0x90) with a velocity of 0. The MIDI specification also allows optimizing data stream usage with "running status": if multiple messages in a sequence have the same status byte, the status byte can be omitted from all except the first message. Signalling NoteOff with a 0x90 message with zero velocity allows extensive use of running status.

FTAP can handle all of the above variants; internally, it places all input data into individual messages without running status, and with NoteOff Events encoded separately from NoteOn events.

Velocity in NoteOff events is discarded. If FTAP detects any MIDI errors, the MIDI_ERROR lines in the output file indicate FTAP's parsed input, not the raw input.

Keyboards can also output many other event types. Attached pedals and the modulation wheel create Control Change Events, and the Pitch Bend Wheel (or the lip sensor on a wind controller) may put out Pitch Wheel change events. There are two different types of Key Pressure events. In addition, some keyboards will send MIDI ACTIVE SENSE, MIDI CLOCK, or System Exclusive messages. FTAP cannot deal with (variable-length) System Exclusive messages (a Juno 106 keyboard puts these out for some reason), and it will quit inelegantly.[9]. FTAP will correctly process and record most most controller messages in the output file. MIDI Channel Pressure, and Program Change will be ignored and discarded, as will Timing Clock or Active Sensing messages. SYSEX messages (or other System Common messages) cause FTAP to terminate, probably ungracefully. WARNING: I have not tested all of these options, so I can't guarantee that they work. On the other hand, I don't know of any failures.

If you want to see what the raw data stream from your keyboard looks like, use the "midicopy" program in the 'bin' (or "utils") directory in the FTAP distribution.

D.2 MIDI Input Devices

Here are some specific notes on the keyboards I've tried. Some keyboards may put out MIDI sequences that FTAP is not currently prepared to handle; if FTAP gives errors with your keyboard, use the "midicopy" program to determine just what your keyboard/controller is doing.

Successfully tested input devices:

1. Roland RD-600 controller: works fine with FTAP. Encodes key release with an 0x80 status byte and a fixed velocity of 64. Uses running status. Use of the Rx and Tx keys allows effective disabling of MIDI Local mode. Pressing the pedal seems to put out multiple controller messages for some reason.

2. Fatar Studio 49: Encodes key release with an 0x80 status byte and variable velocity. Does not use running status. Input device only; requires a separate tone generator.

3. Yamaha DX-100. Uses 0x90 plus 0 velocity for NoteOff, but does not use running status. Not velocity sensitive; constant MIDI velocity of 64 for all keystrokes.

[9]FTAP's input MIDI parsing is perhaps its weakest point. SYSTEM EXCLUSIVE messages are of varying length, and are not handled. ACTIVE SENSE and MIDI CLOCK messages can occur in the middle of other messages, but FTAP just ignores them (I hope!). "linux_midi.c" should probably be rewritten from scratch...but it does work!

4. DX7-II. This worked with older versions of FTAP; I haven't tested it recently. The only likely problem would be whether Channel Pressure Messages get ignored properly (if this doesn't work, the DX7 may allow disabling transmission of Aftertouch). MIDI Local Mode can be disabled.

5. Yamaha Portasound 500M: puts out Timing Clock messages, but FTAP now ignores them.

6. Yamaha WX5 Wind controller. Both breath control and pitch bend are recorded.

Unuseable input devices:

1. Juno-106: puts out System Exclusive messages (at least with our setup); FTAP cannot deal with these.

From the point of view of tapping experiments, the length of key movement for a MIDI keyboard compared to, say, touching a metal plate may be a problem. Note also that input MIDI velocities across different keyboards are not directly comparable in terms of physical velocity. Since all that FTAP requires is that input be in MIDI format, a device other than a keyboard could also be used, e.g., a drum pad, a wind controller, or a custom device of some sort (e.g., an ICUBE system).

If a keyboard allows disabling MIDI LOCAL mode, FTAP can use it for both input and output

D.3 MIDI Output Stream

FTAP's output encodes Note Off messages with 0x80 status and 0 velocity; running status is not used.

FTAP sends a MIDI ACTIVE SENSE message to the output port every 200 ms. Most MIDI devices should have no problem with ACTIVE SENSE, but if this is a problem, you can try setting ACTIVE_SENSE_ON in metron.c to 0 and recompiling FTAP. This deals with a problem where some Linux MIDI drivers would send a MIDI ACTIVE SENSE message on device close; this can trigger some tone generators (e.g., a TX81Z) to require an ACTIVE SENSE message every 300 ms to maintain a tone. This interfered with FTAP's masking noise feature.

D.4 MIDI Output and Output Devices

Much of FTAP's flexibility derives from appropriate programming of the output MIDI device (i.e., tone generator). All of FTAP's control is in terms of MIDI information. For instance, timbre differences (e.g., having two different sounds in a structured metronome sequence) are specified in FTAP by designating different MIDI channels for different events; it is up to the experimenter

to set up the tone generator to respond in an appropriate way on the different MIDI channels. Similarly, relative loudness can be controlled by the use of MIDI velocity parameters, but the actual loudness will depend on the programming of the tone generator and the voice used. In setting up an experiment it is necessary and important for the experimenter to measure and calibrate the specific output devices.

In the simplest case, any tone generator can be used with FTAP, e.g., something which simply generates a sine wave for MIDI channel one. However, a polytimbral tone generator, combined with FTAP's ability to specify MIDI channels, allows the use of separate sounds for pacing and feedback, as well as the use of masking noise. I have used a Yamaha TX-81Z (available used for about $125).

Since output is simply sent to the "/dev/midi" device, it is possible that an internal computer tone generator set to respond to writes to this device would work; this has not been tested.

E FTAP: Design and Implementation

This section contains some discussion of the implementation of FTAP; enough (hopefully) to explain the diagnostics and benchmarking. See also Finney (2001b) and the "scip_summary.text" file in the "doc" directory for further discussion of real-time implementation; these will hopefully convince the reader that FTAP runs with millisecond precision. If you're really interested in the implementation, the source itself is liberally commented (perhaps excessively so; I haven't always had time to clean up out-of-date comments).

Timing in FTAP (time stamping of input events, and scheduing of output events) is done at the millisecond level using the Linux *gettimeofday* system call (which actually provides microsecond level precision). If FTAP is running with root privileges, *sched_setscheduler* and *mlockall* are used to get real-time priority.

Because output may be asynchronous with input (e.g., when delay is used) FTAP uses an output scheduling queue that contains an ordered list of scheduled output events (be they metronome or feedback events), along with the time in which the event should be written out to the MIDI port. This queue is normally checked at least once a millisecond, and any events scheduled at or before the current time are then written out. Since it is theoretically possible for the output queue scheduler to get delayed, the actual and the expected time are checked each time an event is written out; if the current time is different from the expected time, this discrepancy is recorded. When the program is completed, the maximum such discrepancy (as well as the mean) is written both to the screen, and to the output file. If this number is larger than 2 milliseconds or so, you should probably be concerned.

Similarly, MIDI input should be read from the input port at least once a millisecond. Time stamping of a MIDI message is determined by the arrival of the first byte. If there is to be feedback in response to the keystroke, it is written to the output scheduling queue (delayed if necessary).

The initial versions of FTAP (on SGI Irix) enforced such millisecond scheduling by using a 1 ms granularity asynchronous interrupt, but this is not provided on all systems. The current implementation relies on fast CPU speed and high-priority scheduling, and simply does a loop (until end of experiment) of:

```
ReadMidiEvent() ;
ProcOutput();
```

Although this is not provably guaranteed to give millisecond granularity, it does so in practice. If there are problems, they will show up in the output diagnostics printed to the screen at the end of a trial (and also printed to the output file). Since the high-priority scheduling of such a continuous loop (e.g., when run as root with SCHED_FIFO priority) can totally monopolize the machine (to the point of not allowing FTAP to be interrupted by a keyboard interrupt, and requiring a system reset), a .49 ms pause (implemented with "/dev/rtc") is inserted on each loop, so that the mean time between scheduler calls will tend to be .49 ms. If FTAP is run as a normal user, this pause does not occur, so that the time between scheduler calls may in fact be much less (e.g., .01 ms).

F Sample Experiment Driver in Python

FTAP runs a single trial (potentially long and complex), but an experiment typically contains multiple trials, often randomized, blocked, or counterbalanced. FTAP in such a research situation would typically be run from a script of some sort which handles these higher level aspects. A shell script could be used; I use the interpretive language Python, which is installed by default on some Linux installations (e.g., RedHat), or can be downloaded from www.python.org. Python, as a genuine programming language, is more powerful, elegant, and general than shell scripts, and I am told by those who know that it is more flexible and comprehensible than scripting languages like Perl. I like it a lot. See Lutz and Ascher (1999) for an introduction.

An example script in the Python language that I have used for running experiments is the following. For this experiment, there are 11 delay conditions, each run at each of two rates (250 ms or 400 ms); the parameter files names are constructed from a rate prefix and a delay suffix. A sample parameter file name would be 250p_100, for 100 ms of delay at the 250 ms tapping rate. Trials are blocked by rate, and the Python program handles the subject, block, and trial (override) parameters. The program prompts for a parameter file prefix (encoding the rate), subject number, block number

(whether this rate is being run as the first or second block), and runs 3 trials of each condition, in a random order. A break is given after 17 trials. All the experimenter has to do is hit <CR> when prompted to do so.

```
#!/usr/bin/env python

# Sample driver file in Python for an FTAP experiment

import os, whrandom, sys, string

print "\nEnter parameter file prefix: ",
line = sys.stdin.readline()
prefix = (string.split (line) ) [0]

print "\nEnter subject ID: ",
line = sys.stdin.readline()
subject_id = (string.split (line) ) [0]

print "\nEnter block number: ",
line = sys.stdin.readline()
block_id = (string.split (line) ) [0]


# List of all the conditions for this experiment (these are the
# parameter files suffixes).
suffixes = ["_s", "_100", "_150", "_200", "_250", "_300", "_350",
    "_400", "_450", "_500", "_550" ]

# source directory for parameter files
param_path = "/home/sf/texp10/params/"

start_trial = 1
trial_count =  3          # of trials in each condition
trial_break =  17         # give a break after 17 trials


# create a list in which each suffix/condition occurs 3 times
suffixes = trial_count * suffixes
trial_num = start_trial

# Create override parameters for FTAP invocation. Assumes that subject and
# block will not change within a single run
sub_param =   "SUB "   + `subject_id`
block_param = "BLOCK " + `block_id`
```

```
while suffixes:
        if trial_num == trial_break:
              print "\nFIRST BREAK: Hit <CR> to continue.."
              line = sys.stdin.readline()    #  experimenter can type anything


# randomly choose a condition to run...
        suffix = whrandom.choice (suffixes)
        suffixes.remove (suffix)

# construct the command line to execute...
        file = param_path + prefix + suffix
        trial_param = "TRIAL " + `trial_num`

        exec_line = "ftap " + file   + "  '" + sub_param +  " '"  + \
           "  '" + trial_param +  " '" +  "  '" + block_param +  " '"
        print "\n\nHit <CR> to start trial %d, file %s" % (trial_num, file)
        line = sys.stdin.readline()           #  experimenter can type anything

        os.system (exec_line)
        trial_num = trial_num + 1
```

G Interactive Mode

Interactive mode is an self-documenting (that is, largely undocumented) mode of FTAP that you enter if you do not provide a parameter file name of the command line. It is not currently supported; however, I haven't disabled it, so here are the commands. It was once useful for learning FTAP and exploring what parameters do, but you're probably better off editting a test file in one window and running FTAP from another window. Under no circumstances should you do serious data collection this way! Interactive mode allows running multiple trials from one invocation of FTAP. However, the system is not cleanly reset between trials; e.g., parameters which have been changed by triggers will stay changed, and internal variables may also not get reset. Unless you convince me this is a valuable facility, I will probably not respond to comments, questions, or bug reports on interactive mode.

Commands:

1. f <PARAMFILE>: read in the parameters in the named parameter file.
2. p <PARAMSTR>: set a single parameter (e.g., "p FEED_ON 1").
3. t <TRIAL>: set the trial number.

4. q: quit FTAP.

5. r: run a trial with the current set of parameters.

6. l: list the current set of parameters.

H User Enhancements/Code Changes

FTAP provides a fixed set of delay and pitch mappings, which can be chosen by integer parameter values. Additional pitch or delay mappings could be added following the structure of the routines in "map.c"; this requires C code changes. Currently, there are not clean hooks that allow simple linking in of such functions.

The necessary steps are.

1. Create a "#define " for the new feedback mode in in "params.h".

2. Create a new mapping routine modeled on the existing functions in "map.c".

3. Add a "case:" statement in the appropriate "map.c" routine (*pitchmap* or *delaymap*) calling your new mapping routine.

If you choose to make such a change, choose a PMODE or DMODE coding greater than 50 to avoid conflict with future versions of FTAP. If you create a change you think might be generally useful, let me know so I can consider including it in the distributed FTAP version.

For pitch mappings, the logical thing would be able to read mappings in from a file (similar to the current SEQFILE stuff) so that no C coding and recompilation was required. This is not in the current version of FTAP, and will require thinking about file formats for octave and whole keyboard mappings.

I 'playftap' usage

"playftap" is a utility included in the FTAP distribution (see the "utils" and "bin" directories) for "playing" FTAP output files (that is, hearing them as auditory/MIDI output). I have found this useful for getting an intuitive sense of what went on in a trial; it is also good for creating audio examples for presentations. 'playftap" currently works on my system, but I make no guarantees for you. The following is the documentation in the code header as of 9/4/00; there are a number of options to choose exactly what will be sounded.

```
   playftap.c: This is a playback program for FTAP output files, making
   it possible to listen to the data files (including
   things one didn't hear when running the experiment, like the keystrokes
   themselves!).  Various aspects of what you hear can be specified by
   command line arguments. It currently requires that the input file have
   the .abs suffix created by FTAP.  It defaults to playing keystrokes if
   nothing else is specified.

   This code has not been kept up to date, but should work. The accuracy
   of the program argument descriptions needs to be verified.

   USAGE: there are a bunch of command line arguments which allow choosing
exactly what to listen to. It will not be possible to listen to "junk"
events (masking noise), however, you may want to listen to trigger
events (to know where things changed). You may want to listen to
keystroke events, which on the original run did not (in and of
themselves) make any noise. This means that it may be necessary to
allow specifying MIDI channel for keystokes, since the input keystroke
value may not be suitable for the tone generator.

playftap usage is as follows, where all arguments (except filename)
are optional.  This program currently only plays one file at
a time.

playftap -kmft -R<ratio> -T<tchan> -U<tnote> -C<keystrokechan>
-K<keystrokeadd> -S<startpoint> filename

The "k", "m", "f", and "t" turn on playback of keystrokes, metronome,
feedback, and trigger events respectively; you can play these
simultaneously in any combination. Since trigger events in the
file  don't have a MIDI channel or note, the "T" and "U" arguments
allow overriding the built-in defaults of 1 and 120 for channel and
note, respectively. (The sounding time of 75 ms and velocity of
100 for trigger playback are hardwired).
Since keyboard midi channel input doesn't necessarily correspond to a
reasonable sound, "C" allows overriding the MIDI channel in the
input file (perhaps a note should be added, at least for tapping
(vs musical) experiments). The "K" flag provides a value you
can add to keystroke velocity values (to make them louder); its use
automatically turns on keystroke audio ouptut. The "S" flag
allows for a millisecond time (from file start) for when playback
of the file should begin. The "R" flag allows for altering the
rate of playback (slowing down is sometimes useful); 2.0 will play
at half speed, .5 will play at double speed.
```