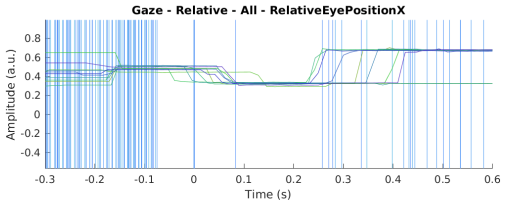
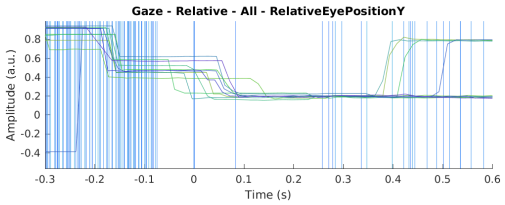
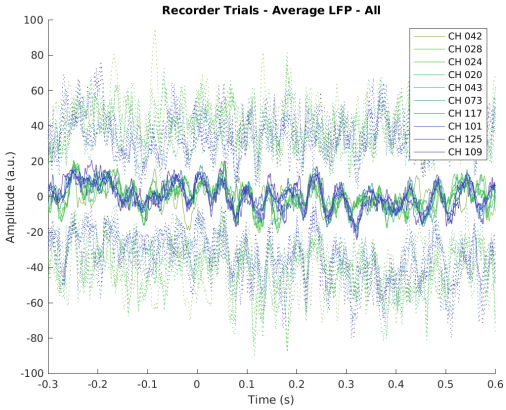
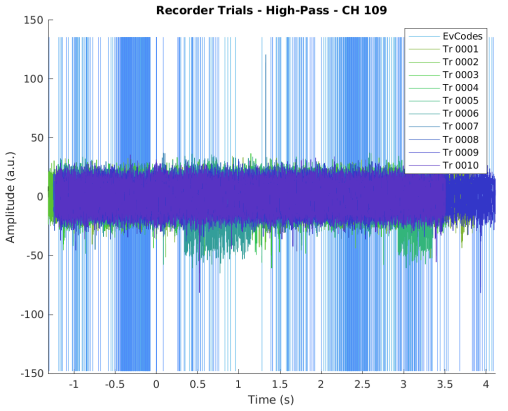
Chris's Experiment Utility Libraries – Function Reference

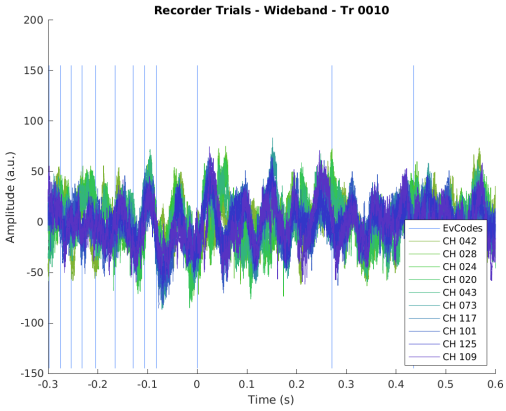
Written by Christopher Thomas – October 25, 2023.











Overview

This is a set of libraries and utilities written to support ephys experiment analyses in Thilo's lab.

This is intended to be a private project for lab-specific code that is not specific to individual experiments. Code that lends itself to reuse outside our lab can be migrated to public projects. Code that's experiment-specific should be in projects associated with those experiments.

Libraries are provided in the "`libraries`" directory. With that directory on path, call the "`addPathsExpUtilsCjt`" function to add sub-folders.

The following sets of library functions are provided:

- **"`euAlign`"** functions (Chapter 3) perform time-alignment of event lists from different sources.
- **"`euChris`"** functions (Chapter 5) manipulate datasets from Chris's experiments.
- **"`euFT`"** functions (Chapter 7) provide wrappers for Field Trip functions and combine commonly-used sets of Field Trip function calls.
- **"`euHLev`"** functions (Chapter 9) perform high-level I/O and signal processing operations. These are intended to called instead of writing commonly-duplicated low-level processing code.
- **"`euPlot`"** functions (Chapter 10) plot experiment data for testing. These don't make paper-quality plots.
- **"`euTools`"** functions (Chapter 11) are helper functions used by specific tools and scripts.
- **"`euUSE`"** functions (Chapter 13) are functions for reading and interpreting USE data (event codes, SynchBox activity, gaze data, and frame data).
- **"`euUtil`"** functions (Chapter 15) are utility functions that don't fit into other categories.

The following sample code is provided:

- **"`do_read_synchbox.m`"** – Demonstration code for reading SynchBox traffic saved by USE. This is described in Chapter 1.
- **"`ft_demo.m`"** – Simplest practical script for reading our lab's datasets into Field Trip. This is described in Chapter 2.

Contents

Part I

Examples

Chapter 1

“synchbox-demo” Example Script

1.1 README.md

```
# Chris's SynchBox Example Script

## Overview

This is a minimum working script showing how to read SynchBox data that was
saved by the USE software suite.

As long as the "exp-utils-cjt" libraries are on path, this should work.

## What This Script Does

* First, a sample folder is selected. This is ‘samples-louie‘ for a typical
ephys session or ‘samples-marcus‘ for a typical computer session. For real
data, this should point to the ‘RuntimeData‘ folder created by USE.

* Raw serial data is read using ‘euUSE_readRawSerialData‘. This saves
communications messages in a Matlab table.

* The "sent" data is parsed using ‘euUSE_parseSerialSentData‘. This
returns a table containing all reward and event code commands that were
transmitted to the SynchBox.

* The "received" data is parsed using ‘euUSE_parseSerialRecvData‘. This
returns a table containing all synchronization pulses, reward pulses,
and event codes that were actually sent by the SynchBox. __NOTE__ - Some
events that happened might not be reported in this list, if the
communications link was saturated.

* The "received" data is parsed using ‘euUSE_parseSerialRecvDataAnalog‘.
This returns a table containing analog sensor readings that the SynchBox
reported. __NOTE__ - Some readings might not be reported, if the
```

```
communications link was saturated.

* All of these data tables are then saved in the "output" folder.


*This is the end of the file.*
```

1.2 do_read_synchbox.m

```
% Quick and dirty test program for parsing USE synchbox traffic.

addpath('lib-exp-utils-cjt');
addPathsExpUtilsCjt;


%
% Configuration.

%rawfolder = 'samples-louie';
rawfolder = 'samples-marcus';

outdir = 'output';


%
% Main program.


disp('-- Reading raw serial data.');
disp(char(datetime));

[ sentdata recvdata ] = euUSE_readRawSerialData(rawfolder);

disp(char(datetime));
disp(sprintf( '.. %d lines sent, %d lines received.', ...
  height(sentdata), height(recvdata) ));


disp('-- Parsing events from sent and received data.');

[ sentrwdA sentrwdB sentcodes ] = ...
  euUSE_parseSerialSentData( sentdata, 'dupbyte' );
[ recvsynchA recvsynchB recvrwdA recvrwdB recvcodes ] = ...
  euUSE_parseSerialRecvData( recvdata, 'dupbyte' );

disp(sprintf( '.. Sent:   %d rwdA   %d rwdB   %d codes', ...
  height(sentrwdA), height(sentrwdB), height(sentcodes) ));
disp(sprintf( ...
```

```
  '.. Received:   %d synchA   %d synchB   %d rwdA   %d rwdB   %d codes', ...
  height(recvsynchA), height(recvsynchB), ...
  height(recvrwdA), height(recvrwdB), height(recvcodes) ));


disp('-- Parsing analog data from received data.');

recvanalog = euUSE_parseSerialRecvDataAnalog( recvdata );

disp(sprintf( '.. %d data points received.', height(recvanalog) ));


disp([ '-- Saving data to "' outdir '".' ]);

save( [ outdir filesep 'rawdata.mat' ], 'sentdata', 'recvdata', '-v7.3' );
save( [ outdir filesep 'sent_events.mat' ], ...
  'sentrwdA', 'sentrwdB', 'sentcodes', '-v7.3' );
save( [ outdir filesep 'recv_events.mat' ], ...
  'recvsynchA', 'recvsynchB', 'recvrwdA', 'recvrwdB', 'recvcodes', '-v7.3' );
save( [ outdir filesep 'recv_analog.mat' ], 'recvanalog', '-v7.3' );

disp('-- Done.');


%
% This is the end of the file.
```

Chapter 2

"ft-demo" Example Script

2.1 README.md

```
# Chris's Field Trip Example Script

## Overview

This is a minimum working script showing how to read data from one of our
lab's experiments into Field Trip and perform processing with it.

About half of this is done using Field Trip's functions directly, and the
remainder is done with wrapper functions that combine many common Field Trip
operations. If you're not sure what a step is doing or how it's doing it,
looking at that function's documentation and function body will help.

To run this script, you'll need Field Trip and several libraries installed;
details are below.

## Getting Field Trip

Field Trip is a set of libraries that reads ephys data and performs signal
processing and various statistical analyses on it. It's a framework that
you can use to build your own analysis scripts.

To get Field Trip:
* Check that you have the Matlab toolboxes you'll need:
    * Signal Processing Toolbox (mandatory)
    * Statistics Toolbox (mandatory)
    * Optimization Toolbox (optional; needed for fitting dipoles)
    * Image Processing Toolbox (optional; needed for MRI)
* Go to [fieldtriptoolbox.org](https://www.fieldtriptoolbox.org).
* Click "latest release" in the sidebar on the top right
(or click [here](https://www.fieldtriptoolbox.org/#latest-release)).
* Look for "FieldTrip version (link) has been released". Follow that
```

```
GitHub link (example:
[Nov. 2021 link](http://github.com/fieldtrip/fieldtrip/releases/tag/20211118)).
* Unpack the archive somewhere appropriate, and add that directory to
Matlab's search path.

Bookmark the following reference pages:
* [Tutorial list.](https://www.fieldtriptoolbox.org/tutorial)
* [Function reference.](https://www.fieldtriptoolbox.org/reference)

More documentation can be found at the
[documentation link](https://www.fieldtriptoolbox.org/documentation).

## Other Libraries Needed

You're also going to need the following libraries. Download the relevant
GitHub projects and make sure the appropriate folders from them are on path:

* [Open Ephys analysis tools](https://github.com/open-ephys/analysis-tools)
(Needed for reading Open Ephys files; the root folder needs to be on path.)
* [NumPy Matlab](https://github.com/kwikteam/npy-matlab)
(Needed for reading Open Ephys files; the "npy-matlab" subfolder needs to be
on path.)
* My [LoopUtil libraries](https://github.com/att-circ-contrl/LoopUtil)
(Needed for reading Intan files and for integrating with Field Trip; the
"libraries" subfolder needs to be on path.)
* My [experiment utility libraries](https://github.com/att-circ-contrl/exp-utils-cjt)
(Needed for processing steps that are specific to our lab, and more Field
Trip integration; the "libraries" subfolder needs to be on path.)

## Using Field Trip

A Field Trip script needs to do the following:
* Read the TTL data from ephys machines and SynchBox data from USE.
* Assemble event code information, reward triggers, and stimulation triggers
from TTL and SynchBox data.
* Time-align signals from different machines (recorder, stimulator, SynchBox,
and USE) to produce a unified dataset.
* Segment the data into trials using event code information.
* Read the analog data trial by trial (to keep memory footprint reasonable).
* Perform re-referencing and artifact rejection.
* Filter the wideband data to get clean LFP-band and high-pass signals.
* Extract spike events and waveforms from the high-pass signal.
* Extract average spike activity (multi-unit activity) from a band-pass signal.
* Stack trigger-aligned trials on to each other and get the average response
and variance of time-aligned trials (a "timelock analysis").
* Perform experiment-specific analysis.

### Reading Data with Field Trip

* `ft_read_header` reads a dataset's configuration information.
```

* ‘ft_read_event‘ reads a dataset’s event lists (TTL data is often stored as events).
* ‘ft_preprocessing‘ is a "do-everything" function. It can either read data without processing it, process data that’s already read, or read data and then process it. At minimum you’ll use it to read data.

For all of these, you can pass it a custom reading function to read data types it doesn’t know about it. We need to do this for all of our data (Intan, Open Ephys, and USE). You can tell it to use appropriate NeuroLoop functions to read these types of data, per the sample code. It can also be told to read events from tables in memory using NeuroLoop functions.

NOTE - When reading data, you pass a trial definition table as part of the configuration structure. Normally this is built using ‘ft_definetrial‘, but because of the way our event codes are set up and because we have to do time alignment between multiple devices, we build this table manually.

We’re using ‘euFT_defineTrialsUsingCodes()‘ for this.

Signal Processing with Field Trip

* Field Trip signal processing calls take the form "‘newdata = ft_XXX(config, olddata)‘. These can be made through calls to ‘ft_preprocessing‘ or by calling the relevant ‘ft_XXX‘ functions directly (these are in the ‘preproc‘ folder). The configuration structure only has to contain the arguments that the particular operation you’re performing cares about.
* ONLY call functions that begin with ‘ft_‘. In particular, anything in the "‘private‘" directory should not be called (its implementation and arguments will change as FT gets updated). The ‘ft_XXX‘ functions are guaranteed to have a stable interface.
* *Almost* all signal processing operations can be performed through ‘ft_preprocessing‘. The exception is resampling: Call ‘ft_resampledata‘ to do that.
* See the preamble of ‘ft_preprocessing‘ for a list of available signal processing operations and the parameters that need to get set to perform them.

Field Trip Data Structures

Data structures I kept having to look up were the following:

* ‘ft_datatype_raw‘ is returned by ‘ft_preprocessing‘ and other signal processing functions. Relevant fields are:
    * ‘label‘ is a {Nchans x 1} cell array of channel names.
    * ‘time‘ is a {1 x Ntrials} cell array of [1 x Nsamples] time axis vectors. Taking the lengths of these will give you the number of samples in each trial without having to load the trial data itself.
    * ‘trial‘ is a {1 x Ntrials} cell array of [Nchans x Nsamples] waveform matrices. This is the raw waveform data.

    * ‘fsample‘ is "deprecated", but it’s still the most reliable way to get the sampling rate for a data structure. Reading it from the header (which is also appended in the data) gives you the wrong answer if you’ve resampled the data (and you often will downsample it).
    * Trial metadata is also included in ‘ft_datatype_raw‘, but it’s much simpler to keep track of that separately if you’re the one who defined the trials in the first place.

* A **header** is returned by ‘ft_read_header‘, and is also included in data as the ‘ft_datatype_raw.hdr‘ field. Relevant fields are:
    * ‘Fs‘ is the *original* sampling rate, before any signal processing.
    * ‘nChans‘ is the number of channels.
    * ‘label‘ is a {Nchans x 1} cell array of channel names.
    * ‘chantype‘ is a {Nchans x 1} cell array of channel type labels. These are arbitrary, but can be useful if you know the conventions used by the hardware-specific driver function that produced them. See the LoopUtil documentation for the types used by the LoopUtil library.
    * ‘nTrials‘ is the number of trials in the raw data. This should always be 1 for continuous recordings like we’re using.

* A **trial definition matrix** is a [Ntrials x 3] matrix defining a set of trials.
    * This is passed as ‘config.trl‘ when calling ‘ft_preprocessing‘ to read data from disk.
    * Columns are ‘first sample‘, ‘last sample‘, and ‘trigger offset‘. The trigger offset is ‘(first sample - trigger sample)‘; a positive value means the trial started after the trigger, and a negative value means the trial started before the trigger.
    * Additional columns from custom trial definition functions get stored in ‘config.trialinfo‘, which is a [Ntrials x (extra columns)] matrix.
    * **NOTE** - According to the documentation, trial definitions (‘trl‘ and ‘trialinfo‘) can be Matlab tables instead of matrices, which allows column names and non-numeric data to be stored. I haven’t tested this, and I suspect that it may misbehave in some situations. Since we’re defining the trials ourselves instead of with ‘ft_definetrial‘, I just store trial metadata in a separate Matlab variable.

What This Script Does

This script performs most of the steps we’ll want to perform when pre-processing data from real experiments:

* It finds the recorder, stimulator, and USE folders and reads metadata from the recorder and stimulator.
* It reads digital/TTL events from the recorder, stimulator, and USE folders. The USE data includes what USE sent to the SynchBox (only USE timestamps) and what the SynchBox sent back (also has SynchBox timestamps).
* It reads gaze and frame data tables from USE.
* It aligns timestamps from all of these sources and translates everything into the recorder’s timeframe.

* It makes sure there's a consistent list with all of the events, since the recorder and stimulator are sometimes not set up to save all TTL inputs.
* It processes the list of event codes to find out when trials should happen. This is done using `euFT_defineTrialsUsingCodes()`, since we need metadata that `ft_definetrial()` doesn't usually look at, and since we're using `TrlStart` and `TrlEnd` to define the trial boundaries instead of fixed padding times.
* It reads the ephys data for all trials, performing filtering:
    * Power line noise and other narrow-band noise is filtered out of the wideband data.
    * The wideband data is low-pass filtered and downsampled to get LFP data.
    * The wideband data is high-pass filtered to get spike waveform data.
    * The wideband data is band-pass filtered, rectified, low-pass filtered, and downsampled to get multi-unit activity data.
* It segments USE event and gaze data per trial as well.
* It computes timelock averages of LFP and MUA data.
* It generates example plots for all of these.

Things that are missing:

* We're not identifying trials with artifacts. This is typically done manually, though automated tools are provided in the LoopUtil library.

* We're not re-referencing the data. Field Trip provides preprocessing options for this, but we'll probably have to do it probe by probe rather than across the whole dataset (common-average referencing the recording channels for each probe).

Miscellaneous Notes

* Right now, data from each device is aligned but stored separately. Eventually we'll want to use `ft_appenddata` to merge data from multiple devices (such as multiple Intan recording controllers, if we're using more than 8 headstages). This can only be done for data that's at a single sampling rate, which may not be practical for wideband and raw spike data.

* Right now, the script doesn't touch spike sorting or try to isolate single-unit spiking activity. The spike sorting pipeline presently needs to work on the entire monolithic dataset, rather than on trials, and that monolithic dataset won't fit in memory. What we're going to have to do is load monolithic data per-probe (64 or 128 channels) and re-save that for the spike sorting pipeline to process (after notch filtering and artifact removal).

This is the end of the file.

2.2 do_demo.m

```
% Short Field Trip example script.
% Written by Christopher Thomas.


%
% Configuration constants.

% Change these to specify what you want to do.


% Folders.

plotdir = 'plots';
outdatadir = 'output';


% Data folder and channels we care about.

want_channel_remap = true;

if true
  % Silicon test.
  inputfolder = 'datasets/20220504-frey-silicon';

  % These are the channels that Louie flagged as particularly interesting.
  % Louie says that channel 106 in particular was task-modulated.
  desired_recchannels = ...
    { 'CH_020', 'CH_021', 'CH_022',   'CH_024', 'CH_026', 'CH_027', ...
      'CH_028', 'CH_030', 'CH_035',   'CH_042', 'CH_060', 'CH_019', ...
      'CH_043', ...
      'CH_071', 'CH_072', 'CH_073',   'CH_075', 'CH_100', 'CH_101', ...
      'CH_106', 'CH_107', 'CH_117',   'CH_116', 'CH_120', 'CH_125', ...
      'CH_067', 'CH_123', 'CH_109',   'CH_122' };

  desired_stimchannels = {};

  % This is the code we want to be at time zero.
  % NOTE - We're adding "RwdA" and "RwdB" as codes "TTLRwdA" and "TTLRwdB".
  trial_align_evcode = 'StimOn';
%  trial_align_evcode = 'TTLRwdA';
end

if false
  % Tungsten test with stimulation.
  inputfolder = 'datasets/20220324-frey-tungsten-stim';

  % There were only three channels used in total.
```

```
  desired_recchannels = { 'AmpA_045', 'AmpA_047' };
  desired_stimchannels = { 'AmpC_011' };

  % This is the code we want to be at time zero.
  % NOTE - We're adding "RwdA" and "RwdB" as codes "TTLRwdA" and "TTLRwdB".
  trial_align_evcode = 'TTLRwdB';
end


% Where to look for event codes and TTL signals in the ephys data.

% These structures describe which TTL bit-lines in the recorder and
% stimulator encode which event signals for this dataset.

% FIXME - We want reward and stim TTLs to be cabled to both machines.
recbitsignals_openephys = struct();
recbitsignals_intan = struct();
stimbitsignals = struct('rwdB', 'Din_002');

% These structures describe which TTL bit-lines or word data channels
% encode event codes for the recorder and stimulator.
% Note that Open Ephys word data starts with bit 0 and Intan bit lines
% start with bit 1. So Open Ephys code words are shifted by 8 bits and
% Intan code words are shifted by 9 bits to get the same data.

reccodesignals_openephys = struct( ...
  'signameraw', 'rawcodes', 'signamecooked', 'cookedcodes', ...
  'channame', 'DigWordsA_000', 'bitshift', 8 );

reccodesignals_intan = struct( ...
  'signameraw', 'rawcodes', 'signamecooked', 'cookedcodes', ...
  'channame', 'Din_*', 'bitshift', 9 );

% FIXME - This might not actually be cabled in some datasets.
stimcodesignals = struct( ...
  'signameraw', 'rawcodes', 'signamecooked', 'cookedcodes', ...
  'channame', 'Din_*', 'bitshift', 9 );


% How to define trials.

% NOTE - We're setting trial_align_evcode earlier in the script now.

% These are codes that carry extra metadata that we want to save; they'll
% show up in "trialinfo" after processing (and in "trl" before that).
trial_metadata_events = ...
  struct( 'trialnum', 'TrialNumber', 'trialindex', 'TrialIndex' );

% This is how much padding we want before 'TrlStart' and after 'TrlEnd'.
padtime = 1.0;
```

```
% Narrow-band frequencies to filter out.

% We have power line peaks at 60 Hz and its harmonics, and also often
% have a peak at around 600 Hz and its harmonics.

notch_filter_freqs = [ 60, 120, 180 ];
notch_filter_bandwidth = 2.0;


% Frequency cutoffs for getting the LFP, spike, and rectified signals.

% The LFP signal is low-pass filtered and downsampled. Typical features are
% in the range of 2 Hz to 200 Hz.

lfp_maxfreq = 300;
lfp_samprate = 2000;

% The spike signal is high-pass filtered. Typical features have a time scale
% of 1 ms or less, but there's often a broad tail lasting several ms.

spike_minfreq = 100;

% The rectified signal is a measure of spiking activity. The signal is
% band-pass filtered, then rectified (absolute value), then low-pass filtered
% at a frequency well below the lower corner, then downsampled.

rect_bandfreqs = [ 1000 4000 ];
rect_lowpassfreq = 500;
rect_samprate = lfp_samprate;


% Nominal frequency for reading gaze data.

% As long as this is higher than the device's sampling rate (300-600 Hz),
% it doesn't really matter what it is.
% The gaze data itself is non-uniformly sampled.

gaze_samprate = lfp_samprate;


% Plotting configuration.

% Number of standard deviations to draw as the confidence interval.
confsigma = 2;


% Debug switches for testing.
```

```
debug_skip_gaze_and_frame = true;

debug_use_fewer_chans = true;

debug_use_fewer_trials = true;
%debug_trials_to_use = 30;
debug_trials_to_use = 10;

if debug_use_fewer_chans && (length(desired_recchannels) > 10)
  % Only drop channels if we have more than 10.
  % Take every third channel (hardcoded).
  desired_recchannels = desired_recchannels(1:3:length(desired_recchannels));
end



%
% Paths.

% Adjust these to match your development environment.

addpath('../libs-ext/lib-exp-utils-cjt');
addpath('../libs-ext/lib-looputil');
addpath('../libs-ext/lib-fieldtrip');
addpath('../libs-ext/lib-openephys');
addpath('../libs-ext/lib-npy-matlab');

% This automatically adds sub-folders.
addPathsExpUtilsCjt;
addPathsLoopUtil;



%
% Start Field Trip.

% Wrapping this to suppress the annoying banner.
evalc('ft_defaults');

% Suppress spammy Field Trip notifications.
ft_notice('off');
ft_info('off');
ft_warning('off');



%
% Other setup.

% Suppress Matlab warnings (the NPy library generates these).
```

```
oldwarnstate = warning('off');

% Limit the number of channels LoopUtil will load into memory at a time.
% 30 ksps double-precision data takes up about 1 GB per channel-hour.
nlFT_setMemChans(8);



%
% Read metadata (paths, headers, and channel lists).

% Get paths to individual devices.

[ folders_openephys folders_intanrec folders_intanstim folders_unity ] = ...
  euUtil_getExperimentFolders(inputfolder);

% FIXME - Assume one recorder dataset and 0 or 1 stimulator datasets.

have_openephys = ~isempty(folders_openephys);
if have_openephys
  folder_record = folders_openephys{1};
else
  folder_record = folders_intanrec{1};
end

have_stim = false;
if ~isempty(folders_intanstim)
  folder_stim = folders_intanstim{1};
  have_stim = true;
end

folder_game = folders_unity{1};


% Get headers.

% NOTE - Field Trip will throw an exception if this fails.
% Add a try/catch block if you want to fail gracefully.
rechdr = ft_read_header( folder_record, 'headerformat', 'nlFT_readHeader' );
if have_stim
  stimhdr = ft_read_header( folder_stim, 'headerformat', 'nlFT_readHeader' );
end


% Read Open Ephys channel mapping, if we can find it.
% FIXME - Only doing this for the recorder!

% NOTE - We're searching the entire tree, not just the recorder folder,
% for the channel map.
[ chanmap_rec_raw chanmap_rec_cooked ] = ...
```

```
  euUtil_getLabelChannelMap_OEv5(inputfolder, folder_record);
have_chanmap = ~isempty(chanmap_rec_raw);

% Forcibly disable channel mapping if we don't want it.
if ~want_channel_remap
  have_chanmap = false;
end

if have_chanmap
  % Raw labels are what we get when loading the save file.
  % Translate cooked desired channel names into raw desired channel names.

  desired_recchannels = nlFT_mapChannelLabels( desired_recchannels, ...
    chanmap_rec_cooked, chanmap_rec_raw );

  badmask = strcmp(desired_recchannels, '');
  if sum(badmask) > 0
    disp('###  Couldn''t map all requested recorder channels!');
    desired_recchannels = desired_recchannels(~badmask);
  end
end


% Figure out what channels we want.

[ pat_ephys pat_digital pat_stimcurrent pat_stimflags ] = ...
  euFT_getChannelNamePatterns();

rec_channels_ephys = ft_channelselection( pat_ephys, rechdr.label, {} );
rec_channels_digital = ft_channelselection( pat_digital, rechdr.label, {} );

stim_channels_ephys = ft_channelselection( pat_ephys, stimhdr.label, {} );
stim_channels_digital = ft_channelselection( pat_digital, stimhdr.label, {} );
stim_channels_current = ...
  ft_channelselection( pat_stimcurrent, stimhdr.label, {} );
stim_channels_flags = ft_channelselection( pat_stimflags, stimhdr.label, {} );

% Keep desired channels that match actual channels.
% FIXME - Ignoring stimulation current and flags!
desired_recchannels = ...
  desired_recchannels( ismember(desired_recchannels, rec_channels_ephys) );
desired_stimchannels = ...
  desired_stimchannels( ismember(desired_stimchannels, stim_channels_ephys) );



%
% Read events.

% Use the default settings for this.
```

```
% Read USE and SynchBox events. This also fetches the code definitions.
% This returns each device's event tables as structure fields.
% This also gives its own banner, so we don't need to print one.
[ boxevents gameevents evcodedefs ] = euUSE_readAllUSEEvents(folder_game);

% Now that we have the code definitions, read events and codes from the
% recorder and stimulator.

% These each return a table of TTL events, and a structure with tables for
% each extracted signal we asked for.

disp('-- Reading digital events from recorder.');

recbitsignals = recbitsignals_openephys;
reccodesignals = reccodesignals_openephys;
if ~have_openephys
  recbitsignals = recbitsignals_intan;
  reccodesignals = reccodesignals_intan;
end

[ recevents_ttl recevents ] = euUSE_readAllEphysEvents( ...
  folder_record, recbitsignals, reccodesignals, evcodedefs );

if have_stim
  disp('-- Reading digital events from stimulator.');
  [ stimevents_ttl stimevents ] = euUSE_readAllEphysEvents( ...
    folder_stim, stimbitsignals, stimcodesignals, evcodedefs );
end

% Read USE gaze and framedata tables.
% These return concatenated table data from the relevant USE folders.
% These take a while, so stub them out for testing.
gamegazedata = table();
gameframedata = table();
if ~debug_skip_gaze_and_frame
  disp('-- Reading USE gaze data.');
  gamegazedata = euUSE_readRawGazeData(folder_game);
  disp('-- Reading USE frame data.');
  gameframedata = euUSE_readRawFrameData(folder_game);
  disp('-- Finished reading USE gaze and frame data.');
end


% Report what we found from each device.

helper_reportEvents('.. From SynchBox:', boxevents);
helper_reportEvents('.. From USE:', gameevents);
helper_reportEvents('.. From recorder:', recevents);
helper_reportEvents('.. From stimulator:', stimevents);
```

```
%
% Clean up timestamps.

% Subtract the enormous offset from the Unity timestamps.
% Unity timestamps start at 1 Jan 1970 by default.

[ unityreftime gameevents ] = ...
  euUSE_removeLargeTimeOffset( gameevents, 'unityTime' );
% We have a reference time now; pass it as an argument to ensure consistency.
[ unityreftime boxevents ] = ...
  euUSE_removeLargeTimeOffset( boxevents, 'unityTime', unityreftime );


% Add a "timestamp in seconds" column to the ephys signal tables.

recevents = ...
  euFT_addEventTimestamps( recevents, rechdr.Fs, 'sample', 'recTime' );
stimevents = ...
  euFT_addEventTimestamps( stimevents, stimhdr.Fs, 'sample', 'stimTime' );



%
% Do time alignment.

% Default alignment config is fine.
alignconfig = struct();

% Just align using event codes. Falling back to reward pulses takes too long.


disp('.. Propagating recorder timestamps to SynchBox.');

% Use raw code bytes for this, to avoid glitching from missing box codes.
eventtables = { recevents.rawcodes, boxevents.rawcodes };
[ newtables times_recorder_synchbox ] = euUSE_alignTwoDevices( ...
  eventtables, 'recTime', 'synchBoxTime', alignconfig );

boxevents = euAlign_addTimesToAllTables( ...
  boxevents, 'synchBoxTime', 'recTime', times_recorder_synchbox );


disp('.. Propagating recorder timestamps to USE.');

% Use cooked codes for this, since both sides have a complete event list.
eventtables = { recevents.cookedcodes, gameevents.cookedcodes };
[ newtables times_recorder_game ] = euUSE_alignTwoDevices( ...
  eventtables, 'recTime', 'unityTime', alignconfig );
```

```
gameevents = euAlign_addTimesToAllTables( ...
  gameevents, 'unityTime', 'recTime', times_recorder_game );


if have_stim
  disp('.. Propagating recorder timestamps to stimulator.');

  % The old test script aligned using SynchBox TTL signals as a fallback.
  % Since we're only using codes here, we don't have a fallback option. Use
  % event codes or fail.

  eventtables = { recevents.cookedcodes, stimevents.cookedcodes };
  [ newtables times_recorder_stimulator ] = euUSE_alignTwoDevices( ...
    eventtables, 'recTime', 'stimTime', alignconfig );

  stimevents = euAlign_addTimesToAllTables( ...
    stimevents, 'stimTime', 'recTime', times_recorder_stimulator );


  % Propagate stimulator timestamps to the SynchBox, in case we need to
  % use the SynchBox's event records with the stimulator.

  disp('.. Propagating stimulator timestamps to SynchBox.');

  boxevents = euAlign_addTimesToAllTables( ...
    boxevents, 'recTime', 'stimTime', times_recorder_stimulator );
end


if ~debug_skip_gaze_and_frame

  % First, make "eyeTime" and "unityTime" columns.
  % Remember to subtract the offset from Unity timestamps.

  gameframedata.eyeTime = gameframedata.EyetrackerTimeSeconds;
  gameframedata.unityTime = ...
    gameframedata.SystemTimeSeconds - unityreftime;

  gamegazedata.eyeTime = gamegazedata.time_seconds;


  % Get alignment information for Unity and eye-tracker timestamps.
  % This information is already in gameframedata; we just have to extract
  % it.

  % Timestamps are not guaranteed to be unique, so filter them.
  times_game_eyetracker = euAlign_getUniqueTimestampTuples( ...
    gameframedata, {'unityTime', 'eyeTime'} );
```

```
  % Unity timestamps are unique but ET timestamps aren't.
  % Interpolate new ET timestamps from the Unity timestamps.

  disp('.. Cleaning up eye tracker timestamps in frame data.');

  gameframedata = euAlign_addTimesToTable( gameframedata, ...
    'unityTime', 'eyeTime', times_game_eyetracker );


  % Add recorder timestamps to game and frame data tables.
  % To do this, we'll also have to augment gaze data with unity timestamps.

  disp('.. Propagating recorder timestamps to frame data table.');

  gameframedata = euAlign_addTimesToTable( gameframedata, ...
    'unityTime', 'recTime', times_recorder_game );

  disp('.. Propagating Unity and recorder timestamps to gaze data table.');

  gamegazedata = euAlign_addTimesToTable( gamegazedata, ...
    'eyeTime', 'unityTime', times_game_eyetracker );
  gamegazedata = euAlign_addTimesToTable( gamegazedata, ...
    'unityTime', 'recTime', times_recorder_game );

end


disp('.. Finished time alignment.');



%
% Clean up the event tables.

% Propagate any missing events to the recorder and stimulator.

% We have SynchBox events with accurate timestamps, and we've aligned
% the synchbox to the ephys machines with high precision.

% NOTE - This only works if we do have accurate time alignment. If we fell
% back to guessing in the previous step, the events will be at the wrong
% times.

% Copy missing events from the SynchBox to the recorder.
disp('-- Checking for missing recorder events.');
recevents = euAlign_copyMissingEventTables( ...
  boxevents, recevents, 'recTime', rechdr.Fs );

% Copy missing events from the SynchBox to the stimulator.
```

```
if have_stim
  disp('-- Checking for missing stimulator events.');
  stimevents = euAlign_copyMissingEventTables( ...
    boxevents, stimevents, 'stimTime', stimhdr.Fs );
end


% Copy TTL events into the event code tables, if present.

disp('-- Copying TTL events into event code streams.');

% NOTE - Not copying into "gameevents" for now.

if isfield(recevents, 'cookedcodes')

  if isfield(recevents, 'rwdA')
    recevents.cookedcodes = euFT_addTTLEventsAsCodes( ...
      recevents.cookedcodes, recevents.rwdA, ...
      'recTime', 'codeLabel', 'TTLRwdA' );
  end

  if isfield(recevents, 'rwdB')
    recevents.cookedcodes = euFT_addTTLEventsAsCodes( ...
      recevents.cookedcodes, recevents.rwdB, ...
      'recTime', 'codeLabel', 'TTLRwdB' );
  end

end

if have_stim
  if isfield(stimevents, 'cookedcodes')

    if isfield(stimevents, 'rwdA')
      stimevents.cookedcodes = euFT_addTTLEventsAsCodes( ...
        stimevents.cookedcodes, stimevents.rwdA, ...
        'stimTime', 'codeLabel', 'TTLRwdA' );
    end

    if isfield(stimevents, 'rwdB')
      stimevents.cookedcodes = euFT_addTTLEventsAsCodes( ...
        stimevents.cookedcodes, stimevents.rwdB, ...
        'stimTime', 'codeLabel', 'TTLRwdB' );
    end

  end
end



%
```

```
% Get trial definitions.


disp('-- Segmenting data into trials.');

% Get event code sequences for "valid" trials (ones where "TrialNumber"
% increased afterwards).

% NOTE - We have to use the recorder code list for this.
% Using the Unity code list gets about 1 ms of jitter.

[ trialcodes_each trialcodes_concat ] = euUSE_segmentTrialsByCodes( ...
  recevents.cookedcodes, 'codeLabel', 'codeData', true );


% Get trial definitions.
% This replaces ft_definetrial().

[ rectrialdefs rectrialdeftable ] = euFT_defineTrialsUsingCodes( ...
  trialcodes_concat, 'codeLabel', 'recTime', rechdr.Fs, ...
  padtime, padtime, 'TrlStart', 'TrlEnd', trial_align_evcode, ...
  trial_metadata_events, 'codeData' );

if have_stim
  % FIXME - We're assuming that we'll get the same set of trials for the
  % recorder and stimulator. That's only the case if the event code
  % sequences received by each are the same and start at the same time!

  trialcodes_concat = euAlign_addTimesToTable( trialcodes_concat, ...
    'recTime', 'stimTime', times_recorder_stimulator );

  [ stimtrialdefs stimtrialdeftable ] = euFT_defineTrialsUsingCodes( ...
    trialcodes_concat, 'codeLabel', 'stimTime', stimhdr.Fs, ...
    padtime, padtime, 'TrlStart', 'TrlEnd', trial_align_evcode, ...
    trial_metadata_events, 'codeData' );
end


% FIXME - For debugging (faster and less memory), keep only a few trials.

if debug_use_fewer_trials
  % Don't touch trialcodes_each and trialcodes_concat; they're not used
  % past here.

  % FIXME - We're assuming the recorder and stimulator trial tables are
  % consistent with each other.

  trialcount = height(rectrialdeftable);
  if trialcount > debug_trials_to_use
    firsttrial = round(0.5 * (trialcount - debug_trials_to_use));
```

```
    lasttrial = firsttrial + debug_trials_to_use - 1;
    firsttrial = max(firsttrial, 1);
    lasttrial = min(lasttrial, trialcount);

    rectrialdeftable = rectrialdeftable(firsttrial:lasttrial,:);
    rectrialdefs = rectrialdefs(firsttrial:lasttrial,:);
  end

  trialcount = height(stimtrialdeftable);
  if trialcount > debug_trials_to_use
    firsttrial = round(0.5 * (trialcount - debug_trials_to_use));
    lasttrial = firsttrial + debug_trials_to_use - 1;
    firsttrial = max(firsttrial, 1);
    lasttrial = min(lasttrial, trialcount);

    stimtrialdeftable = stimtrialdeftable(firsttrial:lasttrial,:);
    stimtrialdefs = stimtrialdefs(firsttrial:lasttrial,:);
  end
end


% FIXME - Sanity check.

if isempty(rectrialdefs)
  error('No valid recorder trial epochs defined!');
end

if have_stim && isempty(stimtrialdefs)
  error('No valid stimulator trial epochs defined!');
end


% NOTE - You'd normally discard known artifact trials here.



%
% Read the ephys data.

% NOTE - We're reading everything into memory at once. This will only work
% if we have few enough channels to fit in memory. To process more data,
% either read it a few trials at a time or a few channels at a time or at
% a lower sampling rate.

% NOTE - For demonstration purposes, I'm just processing recorder series
% here. For stimulator series, use "stimtrialdefs" and "desired_stimchannels".


% First step: Get wideband data into memory and remove any global ramp.
```

```
preproc_config_rec = struct( ...
  'headerfile', folder_record, 'datafile', folder_record, ...
  'headerformat', 'nlFT_readHeader', 'dataformat', 'nlFT_readDataDouble', ...
  'trl', rectrialdefs, 'detrend', 'yes', 'feedback', 'text' );

preproc_config_rec.channel = ...
  ft_channelselection( desired_recchannels, rechdr.label, {} );

disp('.. Reading wideband recorder data.');
recdata_wideband = ft_preprocessing( preproc_config_rec );

% NOTE - We need to turn raw channel labels into cooked channel labels here.

if have_chanmap
  newlabels = nlFT_mapChannelLabels( recdata_wideband.label, ...
    chanmap_rec_raw, chanmap_rec_cooked );

  badmask = strcmp(newlabels, '');
  if sum(badmask) > 0
    disp('###  Couldn''t map all recorder labels!');
    newlabels(badmask) = {'bogus'};
  end

  % There are at least three places where the labels are stored.
  % Update all copies.
  recdata_wideband.label = newlabels;
  recdata_wideband.hdr.label = newlabels;
  rechdr.label = newlabels;
end


% NOTE - You'd normally do re-referencing here.


% Second step: Do notch filtering using our own filter, as FT's brick wall
% filter acts up as of 2021.

disp('.. Performing notch filtering (recorder).');
recdata_wideband = euFT_doBrickNotchRemoval( ...
  recdata_wideband, notch_filter_freqs, notch_filter_bandwidth );


% Third step: Get derived signals (LFP, spike, and rectified activity).

disp('.. Getting LFP, spike, and rectified activity signals.');

[ recdata_lfp, recdata_spike, recdata_activity ] = euFT_getDerivedSignals( ...
  recdata_wideband, lfp_maxfreq, lfp_samprate, spike_minfreq, ...
  rect_bandfreqs, rect_lowpassfreq, rect_samprate, false);
```

```
% Fourth step: Pull in gaze data as well.

gazedata_ft = struct([]);

if (~debug_skip_gaze_and_frame) && (~isempty(gameframedata))

  disp('.. Reading and resampling gaze data.');

  % We're reading this from the USE FrameData table.
  % The cooked gaze information gives XY positions.
  % The raw gaze information in "gamegazedata" uses three different
  % eye-tracker-specific coordinate systems. We don't want to deal with that.

  % Trick Field Trip into reading nonuniform tabular data as if it was a file.

  gaze_columns_wanted = { 'EyePositionX', 'EyePositionY', ...
    'RelativeEyePositionX', 'RelativeEyePositionY' };
  gazemaxrectime = max(gameframedata.recTime);
  nlFT_initReadTable( gameframedata, gaze_columns_wanted, ...
    'recTime', 0.0, 10.0 + gazemaxrectime, gaze_samprate, gaze_samprate );

  % Adjust trial definition sample numbers.
  % Time 0 is consistent between recorder and gaze, since we're using
  % "recTime" as the timestamp in both. So all we're doing is resampling.

  gazetrialdefs = ...
    euFT_resampleTrialDefs( rectrialdefs, rechdr.Fs, gaze_samprate );


  % We're not reading from a file, but Field Trip wants it to still
  % exist, so give it a real folder.

  gazehdr = ...
    ft_read_header( folder_game, 'headerFormat', 'nlFT_readTableHeader' );

  preproc_config_gaze = struct( ...
    'headerfile', folder_game, 'datafile', folder_game, ...
    'headerformat', 'nlFT_readTableHeader', ...
    'dataformat', 'nlFT_readTableData', ...
    'trl', gazetrialdefs, 'feedback', 'text' );
  preproc_config_gaze.channel = ...
    ft_channelselection( gaze_columns_wanted, gazehdr.label, {} );

  gazedata_ft = ft_preprocessing( preproc_config_gaze );

end
```

```
%
% Plot trial data.

% NOTE - Just plotting recorder, not stimulator, for the demo.
% NOTE - Per "trial_metadata_events", we've saved trial number in the
% "trialnum" table column. Pass it to the plotting routine.

disp('-- Plotting ephys signals.');

rectrialnums = rectrialdeftable.trialnum;

euPlot_plotFTTrials( recdata_wideband, rechdr.Fs, ...
  rectrialdefs, rectrialnums, rechdr.Fs, recevents, rechdr.Fs, ...
  { 'oneplot', 'perchannel', 'pertrial' }, ...
  { [], [ -0.3 0.6 ], [ -0.03 0.06 ] }, { 'full', 'zoom', 'detail' }, inf, ...
  'Recorder Trials - Wideband', [ plotdir filesep 'rec-wb' ] );

euPlot_plotFTTrials( recdata_lfp, lfp_samprate, ...
  rectrialdefs, rectrialnums, rechdr.Fs, recevents, rechdr.Fs, ...
  { 'oneplot', 'perchannel', 'pertrial' }, ...
  { [], [ -0.3 0.6 ], [ -0.03 0.06 ] }, { 'full', 'zoom', 'detail' }, inf, ...
  'Recorder Trials - LFP', [ plotdir filesep 'rec-lfp' ] );

euPlot_plotFTTrials( recdata_spike, rechdr.Fs, ...
  rectrialdefs, rectrialnums, rechdr.Fs, recevents, rechdr.Fs, ...
  { 'oneplot', 'perchannel', 'pertrial' }, ...
  { [], [ -0.3 0.6 ], [ -0.03 0.06 ] }, { 'full', 'zoom', 'detail' }, inf, ...
  'Recorder Trials - High-Pass', [ plotdir filesep 'rec-hpf' ] );

euPlot_plotFTTrials( recdata_activity, rect_samprate, ...
  rectrialdefs, rectrialnums, rechdr.Fs, recevents, rechdr.Fs, ...
  { 'oneplot', 'perchannel', 'pertrial' }, ...
  { [], [ -0.3 0.6 ], [ -0.03 0.06 ] }, { 'full', 'zoom', 'detail' }, inf, ...
  'Recorder Trials - Multi-Unit Activity', [ plotdir filesep 'rec-mua' ] );


if (~debug_skip_gaze_and_frame) && (~isempty(gameframedata))

  disp('-- Plotting gaze.');

  % The gaze trial matrix is derived from the recorder trial matrix, so
  % it has the same trial numbers as the recorder.
  gazetrialnums = rectrialnums;

  gaze_chans_abs = { 'EyePositionX', 'EyePositionY' };
  gaze_chans_rel = { 'RelativeEyePositionX', 'RelativeEyePositionY' };

  % Per-trial doesn't tell us much when glancing at it, so just do the stack.

  euPlot_plotAuxData( gazedata_ft, gaze_samprate, ...
```

```
    gazetrialdefs, gazetrialnums, gaze_samprate, recevents, rechdr.Fs, ...
    gaze_chans_abs, { 'oneplot' }, ...
    'Gaze - Absolute', [ plotdir filesep 'gaze-abs' ] );

  euPlot_plotAuxData( gazedata_ft, gaze_samprate, ...
    gazetrialdefs, gazetrialnums, gaze_samprate, recevents, rechdr.Fs, ...
    gaze_chans_rel, { 'oneplot' }, ...
    'Gaze - Relative', [ plotdir filesep 'gaze-rel' ] );

end

disp('-- Finished plotting.');



%
% Do timelock analysis and plot the results.

% NOTE - Just working with the recorder data, not the stimulator data.
% We didn't load and segment the stimulator data for the demo script.


disp('-- Computing time-locked average and variance of ephys signals.')

% For now, just looing at LFP and MUA. Wideband/HPF is less useful.
% It won't be meaningful to compute this for eye movements, I don't think.

% Default configuration is fine.

recavg_activity = ft_timelockanalysis(struct(), recdata_activity);
recavg_lfp = ft_timelockanalysis(struct(), recdata_lfp);


disp('-- Plotting time-locked average data.');

euPlot_plotFTTimelock( recavg_activity, confsigma, ...
  { 'oneplot', 'perchannel' }, ...
  { [], [ -300 600 ], [ -30 60 ] }, { 'full', 'zoom', 'detail' }, inf, ...
  'Recorder Trials - Average Multi-Unit Activity', ...
  [ plotdir filesep 'rec-avg-mua' ] );

euPlot_plotFTTimelock( recavg_lfp, confsigma, ...
  { 'oneplot', 'perchannel' }, ...
  { [], [ -300 600 ], [ -30 60 ] }, { 'full', 'zoom', 'detail' }, inf, ...
  'Recorder Trials - Average LFP', ...
  [ plotdir filesep 'rec-avg-lfp' ] );

disp('-- Finished plotting.');
```

```
%
% Write data to disk.

% FIXME - NYI.



%
% Done.


%
% Helper functions.


% This writes event counts from a specific device to the console.
% Input is a structure containing zero or more tables of events.

function helper_reportEvents(prefix, eventstruct)
  msgtext = prefix;

  evsigs = fieldnames(eventstruct);
  for evidx = 1:length(evsigs)
    thislabel = evsigs{evidx};
    thisdata = eventstruct.(thislabel);
    msgtext = [ msgtext sprintf('  %d %s', height(thisdata), thislabel) ];
  end

  disp(msgtext);
end



%
% This is the end of the file.
```

Part II

Library Functions

Chapter 3

"euAlign" Functions

3.1 euAlign_addTimesToAllTables.m

```
% function newtables = euAlign_addTimesToAllTables( ...
%   oldtables, oldcolumn, newcolumn, corresptimes )
%
% This function augments several tables with an additional timestamp column.
% these new timestamps are derived from an existing timestamp column using a
% correspondence table produced by euAlign_alignTables().
%
% This is a wrapper for euAlign_addTimesToTable().
%
% "oldtables" is a structure with zero or more fields. Each field contains
%   an event table that is to be augmented.
% "oldcolumn" is the name of the existing table column to use to generate
%   timestamps.
% "newcolumn" is the name of the new table column to generate.
% "corresptimes" is a table containing old and new column timestamps at
%   known-corresponding time points, produced by euAlign_alignTableS().
%
% "newtables" is a copy of "oldtables"; all tables in "newtables" have the
%   new timestamp column added. New timestamp values are linearly
%   interpolated between known points of correspondence.
```

3.2 euAlign_addTimesToTable.m

```
% function newtable = euAlign_addTimesToTable( ...
%   oldtable, oldcolumn, newcolumn, corresptimes )
%
% This function augments a table with an additional timestamp column. These
% new timestamps are derived from an existing timestamp column using a
% correspondence table produced by euAlign_alignTables().
```

```
%
% If the new timestamp column already exists or if it can't be generated,
% "newtable" is a copy of "oldtable".
%
% "oldtable" is the table to augment.
% "oldcolumn" is the name of the existing table column to use to generate
%   timestamps.
% "newcolumn" is the name of the new table column to generate.
% "corresptimes" is a table containing old and new column timestamps at
%   known-corresponding time points, produced by euAlign_alignTables().
%
% "newtable" is a copy of "oldtable" with the new column added. New timestamp
%   values are linearly interpolated between known points of correspondence.
```

3.3 euAlign_alignByShiftAndPad.m

```
% function newdata = ...
%   euAlign_alignByShiftAndPad( olddata, oldtimes, reftimes, padmethod )
%
% This function aligns and pads or crops a waveform to match a reference
% time series.
%
% NOTE - The time series are assumed to be uniformly sampled and have the
% same sampling rate.
%
% Waveform extents that do not overlap the reference time series are cropped,
% and the waveform is padded to span the entire reference time series if it
% doesn't already do so.
%
% "olddata" is a vector containing the waveform data to align/crop/pad.
% "oldtimes" is a vector containing the corrsponding time series.
% "reftimes" is a vector containing the desired target time series.
% "padmethod" is 'nan', 'copy', or 'zero', indicating what padded regions
%   are to be filled with. 'copy' duplicates the nearest "olddata" sample.
%
% "newdata" is a shifted cropped/padded version of "olddata" with sample
%   times corresponding to "reftimes".
```

3.4 euAlign_alignTables.m

```
% function [ newfirsttab, newsecondtab, ...
%   firstmatchmask, secondmatchmask, timecorresp ] = ...
%   euAlign_alignTables( firsttab, secondtab, ...
%     firsttimelabel, secondtimelabel, firstdatalabel, seconddatalabel, ...
%     coarsewindows, medwindows, finewindow, outliersigma, verbosity )
%
```

```
% This function time-aligns data tables from two sources, using both the
% two sources' timestamps and the recorded data values. Timestamps are
% expected to be monotonic (tables get sorted to guarantee this).
%
% If data field names are empty strings, alignment is performed based on
% timestamps alone.
%
% NOTE - This takes O(n) memory but O(n^3) time vs the number of events.
% Using data values helps a lot, as it reduces the number of potential
% matches. Using small window sizes also helps a lot.
%
% Timestamps are typically in seconds, but this will tolerate integer
% timestamps, and will preserve data types for interpolated values.
%
% Each input table is augmented with a new column containing aligned
% timestamp values from the other table. A table with known corresponding
% timestamps is also returned, along with vectors indicating which rows in
% the input tables had matching events in the other table.
%
%
% "Coarse" alignment is performed using a sliding window that moves in steps
% equal to the window radius. We pick one representative sample near the
% middle of the window and consider all samples in the window as alignment
% candidates for it.
%
% An alignment using a constant global delay is performed using the first
% coarse window value, before sliding-window coarse alignment is performed.
%
% "Medium" alignment uses each sample in turn as the center of a sliding
% window. All samples in the window are considered as alignment candidates
% for the central sample.
%
% "Fine" alignment uses each sample in turn as the center of a sliding
% window. Within each window, all samples are considered to be aligned with
% their closest corresponding candidates. A fixed time offset is applied to
% all samples and optimized to minimize cost given these matchings. The
% result is used as a final correction for the central sample's time offset.
%
%
% "firsttab" is a table containing event tuples from the first source.
% "secondtab" is a table containing event tuples from the second source.
% "firsttimelabel" specifies the first source's column containing timestamps.
% "secondtimelabel" specifies the second source's column containing timestamps.
% "firstdatalabel" if non-empty specifies the first source's column containing
%   data samples to compare.
% "seconddatalabel" if non-empty specifies the second source's column
%   containing data samples to compare.
% "coarsewindows" is a vector containing window half-widths for the first
%   pass of sliding-window time shifting. These are applied from widest to
%   narrowest.
```

```
% "medwindows" is a vector containing window half-widths for the second pass
%   of sliding-window time shifting. These are applied from widest to
%   narrowest.
% "finewindow" is the window half-width for final non-uniform time shifting.
%   this freezes the event mapping from the previous step and optimizes delay.
% "outliersigma" is used to reject spurious matches. If, within a window, a
%   match's time delay is this many deviations from the mean, it's squashed.
% "verbosity" is 'verbose', 'normal', or 'quiet'.
%
% "newfirsttab" is a copy of "firsttab" with a "secondtimelabel" column added.
% "newsecondtab" is a copy of "secondtab" with a "firsttimelabel" column.
% "firstmatchmask" is a vector indicating which rows in "firsttab" had
%   matching partners in "secondtab".
% "secondmatchmask" is a vector indicating which rows in "secondtab" had
%   matching partners in "firsttab".
% "timecorresp" is a table with "firsttimelabel" and "secondtimelabel"
%   columns, containing tuples with known-good time alignment. This is the
%   "canonical" set of timestamps used to interpolate the time values in
%   "newfirsttab" and "newsecondtab".
```

3.5 euAlign_alignUsingSlidingWindow.m

```
% function [ bestdeltalist bestcostlist ] = ...
%   euAlign_alignUsingSlidingWindow( ...
%     firsttimes, secondtimes, firstdata, seconddata, ...
%     firstcandidates, windowrad, costmethod )
%
% This performs sliding window time alignment of two event series, optionally
% matching data between series. This uses the squared distance cost function,
% and assumes that one matching pair of events will have ideal time alignment.
% Alignment is performed with windows centered around each "candidate" event.
%
% NOTE - This may take a while! There are O(c*w) tests, and each test
% takes O(w2) time for local optimization. So, total time is O(c*w3).
% For global optimization, each test takes O(n*w) time, for O(c*n*w2) time
% in total.
%
% Timestamps are typically in seconds, but this will tolerate integer
% timestamps.
%
% "firsttimes" is a vector of timestamps for the first set of events being
%   matched. This is expected to be monotonic (sorted in increasing order).
% "secondtimes" is a vector of timestamps for the second set of events.
%   This is expected to be monotonic (sorted in increasing order).
% "firstdata" is a vector containing data values for each event in the first
%   set. Set this to [] to skip data matching.
% "seconddata" is a vector containing data values for each event in the second
%   set. Set this to [] to skip data matching.
```

```
% "firstcandidates" is a vector containing indices of elements within
%   "firsttimes" to find optimal time deltas for.
% "windowrad" is the search distance to use when looking for matching
%   elements. This is the sliding window radius.
% "costmethod" is 'global' or 'local'. If it's 'global', all elements of
%   "firsttimes" and "secondtimes" are used when computing the cost function.
%   If it's 'local', only elements within the window range are used.
%
% "bestdeltalist" is a vector containing time deltas such that
%   tfirst + tdelta = tsecond. Each element in "firstcandidates" has a
%   corresponding element in this list (which is NaN if no matching event
%   was found in the second list).
% "bestcostlist" is a vector containing cost function values corresponding to
%   the elements in "bestdeltalist". Deltas are chosen to minimize cost.
```

3.6 euAlign_alignWithFixedMapping.m

```
% function [ bestdeltalist bestcostlist ] = ...
%   euAlign_alignWithFixedMapping( ...
%     firsttimes, secondtimes, firstdata, seconddata, ...
%     firstcandidates, windowrad )
%
% This performs sliding window time alignment of two event series, optionally
% matching data between series. This uses the squared distance cost function.
% Alignment is performed with windows centered around each "candidate" event.
% A single time shift is applied within the window and optimized to minimize
% the cost function.
%
% NOTE - Each event within the window in the first list is assumed to map to
% the nearest event within the window in the second list. So, approximate
% alignment must already have been performed.
%
% NOTE - This takes O(c*w2) time, so use a small window size.
%
% Timestamps are typically in seconds, but this will tolerate integer
% timestamps.
%
% "firsttimes" is a vector of timestamps for the first set of events being
%   matched. This is expected to be monotonic (sorted in increasing order).
% "secondtimes" is a vector of timestamps for the second set of events.
%   This is expected to be monotonic (sorted in increasing order).
% "firstdata" is a vector containing data values for each event in the first
%   set. Set this to [] to skip data matching.
% "seconddata" is a vector containing data values for each event in the second
%   set. Set this to [] to skip data matching.
% "firstcandidates" is a vector containing indices of elements within
%   "firsttimes" to find optimal time deltas for.
% "windowrad" is the search distance to use when looking for matching
```

```
%   elements. This is the sliding window radius.
%
% "bestdeltalist" is a vector containing time deltas such that
%   tfirst + tdelta = tsecond. Each element in "firstcandidates" has a
%   corresponding element in this list (which is NaN if no matching event
%   was found in the second list).
% "bestcostlist" is a vector containing cost function values corresponding to
%   the elements in "bestdeltalist". Deltas are chosen to minimize cost.
```

3.7 euAlign_copyMissingEventTables.m

```
% function newtarget = ...
%   euAlign_copyMissingEventTables( evsource, oldtarget, timelabel, samprate )
%
% This function augments a structure containing event tables with new event
% tables copied from a different structure. Timestamps from the specified
% column are translated to sample counts in the new event tables.
%
% "evsource" is a structure with zero or more fields. Each field contains
%   an event table that is to be copied. Empty tables are not copied.
% "oldtarget" is a structure with zero or more fields. Each field contains
%   an event table. Event tables that are missing or empty are replaced.
% "timelabel" is the name of the table column in "evsource" to use when
%   generating sample counts in "newtarget".
% "samprate" is the sampling rate to use when translating timestamps in
%   seconds into sample counts.
%
% "newtarget" is a copy of "oldtarget" with any missing or empty tables
%   overwritten with translated non-empty tables from "evsource".
```

3.8 euAlign_evalAlignCostFunctionSquare.m

```
% function totalcost = euAlign_evalAlignCostFunctionSquare( ...
%   firsttimes, secondtimes, firstdata, seconddata, windowrad )
%
% This evaluates a cost function for an attempted alignment between two
% event lists. Optionally event data codes are presented that also have to
% match. Only events within the window radius of each other can match.
%
% This particular cost function is the sum of the squared distances between
% matching events. This takes O(n*w) time to compute.
%
% "firsttimes" is a vector of timestamps for the first set of events being
%   matched. This is expected to be monotonic (sorted in increasing order).
% "secondtimes" is a vector of timestamps for the second set of events.
%   This is expected to be monotonic (sorted in increasing order).
```

```
% "firstdata" is a vector containing data values for each event in the first
%   set. Set this to [] to skip data matching.
% "seconddata" is a vector containing data values for each event in the second
%   set. Set this to [] to skip data matching.
% "windowrad" is the search distance to use when looking for matching
%   elements.
%
% "totalcost" is the value of the cost function (smaller is better).
```

3.9 euAlign_fakeAlignmentWithExtents.m

```
% function timecorresp = euAlign_fakeAlignmentWithExtents( ...
%   firsttimelabel, firsttimeextents, secondtimelabel, secondtimeextents )
%
% This function produces a fake time alignment reference table using the
% extents of two timestamp series. The fake alignment centers the series
% on each other.
%
% This is intended to be used when there isn't enough information to call
% "euAlign_alignTables()".
%
% NOTE - Time ranges shorter than 1 microsecond are assumed to be bogus.
%
% "firsttimelabel" is a column label for timestamps from the first series.
% "firsttimeseries" contains timestamp values from the first series. This
%   typically just holds the minimum and maximum timestamp value.
% "secondtimelabel" is a column label for timestamps from the second series.
% "secondtimeseries" contains timestamp values from the second series. This
%   typically just holds the minimum and maximum timestamp value.
%
% "timecorresp" is a table with "firsttimelabel" and "secondtimelabel"
%   columns, containing tuples with corresponding timestamps. This may be
%   used to interpolate time values from one time base to the other.
```

3.10 euAlign_findMatchingEvents.m

```
% function [ firstmatches secondmatches ] = euAlign_findMatchingEvents( ...
%   firsttimes, secondtimes, firstdata, seconddata, windowrad )
%
% This performs sliding-window matching between two event series, finding
% corresponding events from the two input series. Matching events are those
% with the smallest squared time distance. This optionally requires matching
% data values between corresponding events.
%
% NOTE - Matches are a 1:1 mapping. Any given first-list event will match at
% most one second-list event, and vice-versa.
```

```
%
% Timestamps are typically in seconds, but this will tolerate integer
% timestamps.
%
% NOTE - This takes O(n*w) time, so it should be fairly fast.
%
% "firsttimes" is a vector of timestamps for the first set of events being
%   matched. This is expected to be monotonic (sorted in increasing order).
% "secondtimes" is a vector of timestamps for the second set of events.
%   This is expected to be monotonic (sorted in increasing order).
% "firstdata" is a vector containing data values for each event in the first
%   set. Set this to [] to skip data matching.
% "seconddata" is a vector containing data values for each event in the second
%   set. Set this to [] to skip data matching.
% "windowrad" is the search distance to use when looking for matching
%   elements. This is the sliding window radius.
%
% "firstmatches" is a vector with one element per entry in the first list
%   containing the index of the matching entry in the second list, or NaN
%   if there was no match.
% "secondmatches" is a vector with one element per entry in the second list
%   containing the index of the matching entry in the first list, or NaN if
%   there was no match.
```

3.11 euAlign_getDefaultAlignConfig.m

```
% function newconfig = euAlign_getDefaultAlignConfig( oldconfig )
%
% This function fills in missing time alignment parameters with reasonable
% default values.
%
% These configuration parameters are intended to be used with
% "euAlign_alignTables()".
%
% "oldconfig" is a structure with zero or more of the following fields:
%   "coarsewindows" is a vector with coarse alignment window half-widths.
%   "medwindows" is a vector with medium alignment window half-widths.
%   "finewindow" is the window half-width for final non-uniform alignment.
%   "outliersigma" is the threshold for rejecting spurious matches.
%   "verbosity" is 'verbose', 'normal', or 'quiet'.
%
% "newconfig" is a copy of "oldconfig" with missing fields added.
```

3.12 euAlign_getResampledOverlappedWaves.m

```
% function [ aligned_time, aligned_first, aligned_second ] = ...
```

```
%   euAlign_getResampledOverlappedWaves( ...
%     first_time, first_wave, second_time, second_wave, time_bin_size )
%
% This processes two input waveforms that were sampled using different time
% series, and produces versions of these waveforms sampled using a common
% time series for the portions of the waveforms that overlap in time.
%
% This does quick and dirty resampling via time binning, averaging samples
% within each bin. There must be at least one time sample from each input
% waveform within each bin. Since the bin window has a top-hat profile, high
% frequencies may be aliased down during resampling.
%
% "first_time" is a vector with sampling times for the first waveform.
% "first_wave" is a vector with sample values for the first waveform.
% "second_time" is a vector with sampling times for the second waveform.
% "second_wave" is a vector with sample values for the second waveform.
% "time_bin_size" is the duration of the time bins to use when resampling.
%
% "aligned_time" is a vector containing the centre values of each time bin.
% "aligned_first" is a vector containing sample values from the first
%   waveform, averaged within each time bin.
% "aligned_second" is a vector containing sample values from the second
%   waveform, averaged within each time bin.
```

3.13 euAlign_getSlidingWindowIndices.m

```
% function [ firstindices lastindices ] = ...
%   euAlign_getSlidingWindowIndices( firsttimes, secondtimes, windowrad )
%
% This compiles a list of spans within "secondtimes" that are within a
% search range of any given event within "firsttimes". For each time in
% "firsttimes", a span of indices is found such that times in "secondtimes"
% within that span are in the range (t-radius) to (t+radius).
%
% "firsttimes" is a list of event times to use as window centers. This must
%   be sorted in ascending order.
% "secondtimes" is a list of event times to find windows within. This must
%   be sorted in ascending order.
% "windowrad" is the maximum acceptable difference between a window center
%   time from "firstimes" and an event time within "secondtimes".
%
% "firstindices" and "lastindices" are vectors containing the indices of the
%   first and last valid events in "secondtimes", for each time in
%   "firsttimes". For time firsttimes(k), the valid span in secondtimes is
%   from firstindices(k) to lastindices(k).
%
% NOTE - Entries with no matching elements have "NaN" stored. Check for this.
```

3.14 euAlign_getUniqueTimestampTuples.m

```
% function corresptimes = ...
%   euAlign_getUniqueTimestampTuples( evtable, timecolumns )
%
% This function searches a table of events that are labelled with one or more
% different timestamp columns. Timestamps within a column may repeat; for
% these cases, rows with any timestamp that has already been seen are
% discarded.
%
% "evtable" is the data table to read timestamps from.
% "timecolumns" is a cell array containing timestamp column labels.
%
% "corresptimes" is a table containing _only_ timestamp columns, with
%   timestamp values that are guaranteed to be unique.
```

3.15 euAlign_getWindowExemplars.m

```
% function [ mostidxlist medianidxlist leastidxlist ] = ...
%   euAlign_getWindowExemplars( ...
%     firsttimes, secondtimes, firstdata, seconddata, windowrad )
%
% This picks "exemplar" events from an event series, for purposes of alignment
% between two event series.
%
% The "first" event series is segmented into windows whose overlap is less
% than the window radius. Corresponding windows within the "second" event
% series are found. For each such pair of windows, each event within the first
% window is considered, and the number of potential matches it has in the
% second window is computed (typically requiring matching "data" values).
%
% For each "first" event window, the events having the most potential matches,
% least potential matches, and median number of potential matches are chosen.
% The indices of these events within the "first" event list are returned.
%
% NOTE - This may take a while. It's O(n*w).
%
% "firsttimes" is a vector of timestamps for the first set of events being
%   matched. This is expected to be monotonic (sorted in increasing order).
% "secondtimes" is a vector of timestamps for the second set of events.
%   This is expected to be monotonic (sorted in increasing order).
% "firstdata" is a vector containing data values for each event in the first
%   set. Set this to [] to skip data matching.
% "seconddata" is a vector containing data values for each event in the second
%   set. Set this to [] to skip data matching.
% "windowrad" is the window radius (half-width).
%
```

```
% "mostidxlist" is a vector containing indices of elements within
%   "firsttimes" that are window exemplars with large numbers of matches.
% "medianidxlist" is a vector containing indices of elements within
%   "firsttimes" that are window exemplars with typical numbers of matches.
% "leastidxlist" is a vector containing indices of elements within
%   "firsttimes" that are window exemplars with small numbers of matches.
```

3.16 euAlign_squashOutliers.m

```
% function newdata = euAlign_squashOutliers( ...
%   timeseries, olddata, windowrad, outliersigma )
%
% This performs sliding-window outlier rejection. Data elements that are
% more than the specified number of deviations from the mean within the
% window are replaced with NaN in the output.
%
% There must be at least 6 non-NaN data elements in the window for squashing
% to be performed; otherwise all samples are kept.
%
% "timeseries" is a vector of timestamps for the data values being processed.
% "olddata" is a vector containing data values to remove outliers from.
% "windowrad" is the window half-width to use when evaluating statistics.
% "outliersigma" is the rejection threshold, in deviations from the mean.
%
% "newdata" is a copy of "olddata" with outlier values replaced with NaN.
```

Chapter 4

"euChris" Notes

4.1 CHRISBATCHDEFS.txt

```
A "batch definition" describes a list of channels to be processed.

This is either a cell array or a character vector.

If it's a cell array, it contains a list of names of channels to be read.

If it's a character vector, it's one of the following:

  'trig' indicates the trigger channel from the experiment.
  'hint' indicates the channels listed in the "extrachans" hint field.
  'all' indicates that all channels are to be read.

(This is the end of the file.)
```

4.2 CHRISCASEMETA.txt

```
The "euHLev_getChrisMetadata()" function accepts "case" definition
structures as arguments; these are described below. The returned metadata
structures (per CHRISEXPMETA.txt) are augmented with these case definitions.


Case definition structures have the following fields:

"folder" is the path to extract experiment metadata from.
"prefix" is a character vector with a filename-safe prefix to use with plots.
"title" is a character vector with a human-readable description to use in
  plots.
"setprefix" is a character vector with a filename-safe prefix to use with
  plots for the group of cases of which this case is a part.
```

```
"settitle" is a character vector with a human-readable description to use
  in plots for the group of cases of which this case is a part.
"exptype" is a character vector defining the type of dataset being processed
  (e.g. 'loop2302').
"hint" is a structure containing hints for parsing and/or data processing.
  This may be a structure with no fields or an empty structure array.


Hint fields recognized by library functions are as follows:

"extrachans" is a cell array containing Field Trip channel labels of
  additional channels to plot.
"extrachanprefixes" is a cell array with filename-safe prefixes to use when
  plotting data derived from the channels listed in "extrachans".
"extrachantitles" is a cell array with human-readable titles to use when
  plotting data derived from the channels listed in "extrachans".

"chanbatches" is a cell array defining groups of channels to process. Each
  group is stored as five successive entries in "chanbatches":
  1) a filename-safe label
  2) a human-readable title
  3) a sprintf pattern for channel names that contains a channel number
     pattern
  4) a vector containing channel numbers to iterate through
  5) a vector containing channel numbers to blacklist


(This is the end of the file.)
```

4.3 CHRISCOOKEDMETA.txt

```
The "euChris_getCookedMetadata_XXX()" functions transform a raw metadata
structure for one folder (per RAWFOLDERMETA.txt) into an interpreted
("cooked") metadata structure, with the fields described below.


Fields specific to 'loop2302' type metadata:

(These are NaN or empty vectors/arrays if undefined.)

  "samprate" is the acquisition sampling rate.

  "rawchans" is a list of input channel labels before channel mapping.
    This is extracted from the Intan recorder, not FT, if possible.
  "cookedchans" is a list of input channel labels after channel mapping.

  "readfromfile" is true if a File Reader node was used and false otherwise.
    Channel labels are unreliable in this situation (the reader takes them
```

```
  from the file being read but doesn't save them anywhere).

"chanmapoldnums" is a vector indexed by new channel number containing the
  corresponding old channel numbers. This is taken from the Open Ephys
  channel mapping node, and is [] if that node wasn't found.
"chanmapftraw" is a cell array containing FT channel names before mapping.
  This may be {} if the FT channel map couldn't be extracted.
"chanmapftcooked" is a cell array containing FT channel names after mapping.
  This may be {} if the FT channel map couldn't be extracted.

"torteband" [ low high ] has the Phase Calculator's band corners.
"tortemode" is the output mode. We want a value of 'both'.
"torteinchans" is a vector of channel numbers used as the Phase
  Calculator's input. We usually want exactly one channel here.
"torteextrachan" is the channel number that the Phase Calculator adds
  for magnitude output, in mag+phase mode (this is 1-based).

"crossmagchan" is the channel number of the magnitude detector's input.
"crossmagthresh" is the multiplier used to get the magnitude threshold
  from the RMS average magnitude.
"crossmagtau" is the decay time in seconds for computing this RMS average.
"crossmagbank" is the bank index (0-based) of the detector's TTL output.
"crossmagbit" is the bit number (0-based) of the detector's TTL output.

"crossphasechan" is the channel number of the phase detector's input.
"crossphaseval" is the detector's target value (phase threshold in degrees).
"crossphasebank" is the bank index (0-based) of the detector's TTL output.
"crossphasebit" is the bit number (0-based) of the detector's TTL output.

"crossrandchan" is the channel number of the random phase comparator's
  input.
"crossrandbank" is the bank index (0-based) of the detector's TTL output.
"crossrandbit" is the bit number (0-based) of the detector's TTL output.

"randwasjitter" is true if the conditional trigger faked random phase
  targets by adding jitter to a fixed phase target.

"trigbank" is the bank index (0-based) of the conditional trigger's TTL
  outputs.
"trigphasebit" is the bit index (0-based) of the phase-aligned trigger
  output, or NaN if this wasn't generated.
"trigrandbit" is the bit index (0-based) of the random-phase trigger
  output, or NaN if this wasn't generated.
"trigmagbit" is the bit index (0-based) of the power excursion trigger
  output, or NaN if this wasn't generated.
"trignowbit" is the bit index (0-based) of the "trigger immediately when
  RwdB goes high" trigger output, or NaN if this wasn't generated.

"ardinbit" is the bit index (0-based) of the Arduino's triggering input.
```

```
  "filewritenodes" is a vector containing node ID numbers of file writers.
  "filewritechanmasks" is a cell array containing logical vectors indicating
    which channels were saved for each file writing node.
  "filewritehasevents" is a logical vector indicating whether each node
    saved TTL events or not.

FIXME - Channel labels will be iffy. They use OE's naming conventions rather
than FT's, and may be in a different order (due to aux channels etc).



(This is the end of the file.)
```

4.4 CHRISEXPCONFIGS.txt

```
High-level experiment configuration is stored as a structure, with fields
depending on the experiment type.


For experiments of type 'loop2302', fields are as follows:

"chan_wb_num" is the Open Ephys channel number of the wideband signal.
"chan_mag_num" is the Open Ephys channel number with the analytic magnitude.
"chan_phase_num" is the Open Ephys channel number with the analytic phase.

"chan_wb_oelabel" is the Open Ephys name of the wideband channel.

"trigtype" is 'phase', 'random', 'power', 'immediate', or 'none'. This
  indicates how the Arduino output, if any, was generated.

"file_path_first" is the folder containing the wideband data and loopback
  trigger signal.
"file_path_second" is the folder containing derived and internally generated
  signals, or '' if these weren't saved.


(Signals saved by the primary file writing node:)

"chan_wb_ftlabel" is the Field Trip name of the saved wideband signal,
  or '' if this wasn't saved.

"chan_ttl_loopback_trig_ftlabel" is the Field Trip name of the saved
  stimulation trigger TTL signal as read back by the Intan recorder,
  or '' if this wasn't saved.


(Signals saved by the secondary file writing node:)
```

```
"chan_mag_ftlabel" is the Field Trip name of the saved magnitude signal,
  or '' if this wasn't saved.
"chan_phase_ftlabel" is the Field Trip name of the saved phase signal,
  or '' if this wasn't saved.

"chan_ttl_detect_mag_ftlabel" is the Field Trip name of the saved magnitude
  detection TTL signal, or '' if this wasn't saved.
"chan_ttl_detect_phase_ftlabel" is the Field Trip name of the saved
  desired-phase detection TTL signal, or '' if this wasn't saved.
"chan_ttl_detect_rand_ftlabel" is the Field Trip name of the saved
  random-phase detection TTL signal, or '' if this wasn't saved.

"chan_ttl_trig_phase_ftlabel" is the Field Trip name of the saved stimulation
  trigger TTL signal for fixed-phase stimulation, or '' if this wasn't saved.
"chan_ttl_trig_rand_ftlabel" is the Field Trip name of the saved stimulation
  trigger TTL signal for random-phase stimulation, or '' if this wasn't saved.
"chan_ttl_trig_power_ftlabel" is the Field Trip name of the saved stimulation
  trigger TTL signal for magnitude excursion stimulation, or '' if this wasn't
  saved.
"chan_ttl_trig_immed_ftlabel" is the Field Trip name of the saved stimulation
  trigger TTL signal for immediate stimulation, or '' if this wasn't saved.

"chan_ttl_trig_selected_ftlabel" is the Field Trip name of the saved
  stimulation trigger TTL signal that was actually sent to the stimulator.



(This is the end of the file.)
```

4.5 CHRISEXPMETA.txt

```
The "euChris_parseExperimentConfig()" function returns metadata structures
describing the configuration of experiments that used Chris's signal chains.
The contents of these metadata structures is described below.



Fields common to all types of metadata structure:

"summary" is a cell array of character vectors containing a short
  human-readable summary of the experiment configuration.
"details" is a cell array of character vectors containing a human-readable
  detailed description of the experiment config.

"diagmsgs" is a cell array containing diagnostic messages _and_ error and
  warning messages generated during processing.
"errmsgs" is a cell array containing _only_ error and warning messages
  generated during processing.
```

"rawmetalist" is a copy of the "rawmetalist" cell array passed to
  "euChris_parseExperimentConfig()". This contains metadata structures
  returned by euHLev_getAllMetadata_XXX, per RAWFOLDERMETA.txt.

"exptype" is a label defining the type of signal chain used; e.g. 'loop2023'.
"hintdata" is a structure containing hints for processing metadata. If there
  are no hints, this will be a structure with no fields.

"cookedmeta" is a structure containing type-specific derived metadata, per
  CHRISCOOKEDMETA.txt.

"expconfig" is a structure describing the experiment configuration, per
  CHRISEXPCONFIGS.txt.

"casemeta" (optional) is a structure added by euChris_getChrisMetadata()
  containing case metadata. Fields are per CHRISCASEMETA.txt.

(This is the end of the file.)

4.6 CHRISMUAPARAMS.txt

Multi-unit activity analysis parameters are stored as a structure with the
following fields:

Common to most functions:

"time_window_ms" is the duration in milliseconds of the time windows used
  for extracting average statistics.

"time_before_ms" is a timestamp in milliseconds specifying where the middle
  of the "before stimulation" time window should be.

"timelist_after_ms" is a vector containing timestamps in milliseconds
  specifying where the middle of the "after stimulation" time windows
  should be.

Used by (FIXME - Cross-correlation is not yet a library function):

"xcorr_range_ms" [min max] is the range of values in milliseconds to use
  for time lag when performing cross-correlation.

(This is the end of the file.)

4.7 CHRISOSCPARAMS.txt

```
Oscillation fit parameters are stored as a structure with the following
fields:


Fields relating to finding the global dominant oscillation frequency:

"window_search" [ min max ] is a time range (in seconds) to look at when
  measuring the dominant oscillation frequency.

"freq_search" [ min max ] is the frequency range to search for the
  dominant oscillation frequency.

"freq_drift" [ min max ] is the minimum and maximum multiple of the
  dominant frequency to accept when curve-fitting in time windows.
  E.g. [ 0.5 1.5 ] accepts from 0.5 * fdominant to 1.5 * fdominant.


Fields relating to measuring local oscillations within specific windows:

"min_before_strength" is the minimum oscillation magnitude a channel
  has to have, as a fraction of the strongest channel's magnitude, to
  be considered to be oscillating before stimulation.

"window_lambda" is the width of the curve-fitting time window, as a
  multiple of the dominant oscillation wavelength.

"time_before" is the desired timestamp (in seconds) of the middle of the
  before-stimulation curve-fitting window.

"timelist_after" is a vector containing desired timestamps (in seconds) of
  the middle of after-stimulation curve fitting windows.


Debugging fields:

"use_line_fit" (optional) is true to subtract a line fit before performing
  a cosine fit, and false or absent to just subtract the mean.

"debug_save_waves" (optional) is true to save the waveforms being curve
  fit into the results data structure, and false or absent to discard them.


(This is the end of the file.)
```

4.8 CHRISSIGNALS.txt

```
Signals extracted by euChris_extractSignals_XXX are stored in a structure,
with fields depending on experiment type.


For experiments of type 'loop2302', fields are as follows:

NOTE - Wideband and signals derived from it should always exist, but
other fields aren't gauranteed to be present.

"wb_time" and "wb_wave" are the wideband signal.
"lfp_time" and "lfp_wave" are the wide-cut LFP signal.
"band_time" and "band_wave" are the ideal (acausal) narrow-band signal.
"delayband_time" and "delayband_wave" are a version of the narrow-band
  signal produced with a causal filter. This won't match TORTE's filter.

"canon_time", "canon_mag", "canon_phase", and "canon_rms" are derived
  from the acausal band-pass signal. These are the time, analytic
  magnitude and phase, and the acausal moving RMS average of the
  magnitude, respectively.
"delayed_time", "delayed_mag", "delayed_phase", and "delayed_rms"
  are derived from the delayed (causal) band-pass signal. These are the
  time, analytic magnitude and phase, and delayed (causal) moving RMS
  average of the magnitude, respectively.
"canon_magflag", "canon_phaseflag", "delayed_magflag", and
  "delayed_phaseflag" are the magnitude excursion detection flag and
  the phase target match flag derived from the acausal and delayed
  (causal) signals described above.
"canon_magflag_edges", "canon_phaseflag_edges", "delayed_magflag_edges",
  and "delayed_phaseflag_edges" are vectors holding timestamps of rising
  edges of the corresponding magnitude detection flags and of the
  delayed phase detection flag, and timestamps of the high pulse
  midpoints of the acausal phase detection flag.

"torte_time", "torte_mag", and "torte_phase" are the recorded values
  of the TNE Lab Phase Calculator plugin's estimates of instantaneous
  magnitude and instantaneous phase.
  NOTE - These signals are not guaranteed to exist!
"torte_wave" is a reconstruction of the narrow-band signal using
  "torte_mag" and "torte_phase". This should look like "delayband_wave".

"XXX_ftevents", "XXX_wave", "XXX_time", and "XXX_edges" are stored for
  each of several TTL signals.
  NOTE - These signals are not guaranteed to exist!
  "XXX_ftevents" holds a Field Trip event structure array for events
    associated with this TTL signal, per ft_read_event().
  "XXX_wave" is a logical vector holding time-series waveform data for
    this TTL signal.
```

```
  "XXX_time" is a vector holding waveform timestamp data for this signal.
  "XXX_edges" is a vector holding timestamps of rising signal edges.
  Signals saved (values of "XXX") are "loopback", "detectmag",
  "detectphase", "detectrand", "trigphase", "trigrand", "trigpower",
  "trigimmed", and "trigused" (a duplicate of one of the other "trig"
  signals).



(This is the end of the file.)
```

4.9 CHRISSTIMFEATURES.txt

```
Stimulation response features are stored as a structure with the following
fields. Each response describes one trial (or one timelock-average from a
set of trials); responses for multiple trials are typically returned as a
cell array of per-trial responses.



== Fields common to all types of feature extraction:


General metadata:

"trialnum" is the trial number.

"winbefore" is a scalar containing the timestamp of the midpoint of the
  before-stimulation time window, in seconds.

"winafter" is a 1 x Nwindows matrix containing the timestamps of the
  midpoints of the after-stimulation time windows, in seconds.

(Time window durations are not stored in this structure.)



Metadata labels expected by euPlot_plotResponseStatistics:

"sessionlabel" is a character vector containing a filename-safe label for
  the recording session (e.g. '20230808a').

"caselabel" is a character vector containing a filename-safe label for the
  group of trials that was aggregated (typically corresponding to one set
  of test conditions; e.g. 'be150rand').

"probelabel" is a character vector containing a filename-safe label for the
  group of channels being processed (typically corresponding to one probe;
```

```
  e.g. 'pr1acc').

"chanlabels" is a cell array containing Field Trip channel labels for each
  of the channels in this response.

"chanidx" is a Nchans x 1 matrix containing sequentially numbered channel
  indices. This is used as a data series to plot data against channel index.

"winidx" is a 1 x Nwindows matrix containing sequentially numbered window
  indices. This is used as a data series to plot data against window nuimber.



Normalized data fields that may be provided by test scripts:

"FOOnormCASE" a Nchans x Nwindows matrix containing an extracted feature
  value (FOO) normalized against that feature's value from a baseline test
  case, for each channel and each requested window location.

"FOObaseCASE" is a Nchans x Nwindows matrix containing the extracted
  feature value that was normalized against, corresponding to each
  "FOOnormCASE" entry above. This is the feature value from a different
  case (the baseline case).

These are typically generated by "euChris_calcNormalizedResponse".

NOTE - Any normalized data with negligible or noisy baseline values will be
meaningless. Pruning such cases is strongly advised.



== Fields for local field potential oscillations:


Fields provided by euChris_extractStimOscillationResponse:

"oscfreq" is the dominant frequency detected in the trial.

"magbefore" is a Nchans x 1 matrix containing magnitudes of the dominant
  oscillation before stimulation for each channel.

"freqbefore" is a Nchans x 1 matrix containing frequencies of the dominant
  oscillation before stimulation for each channel.

"phasebefore" is a Nchans x 1 matrix containing the phase (in radians) of
  the dominant oscillation before stimulation for each channel at the
  window midpoint.

"meanbefore" is a Nchans x 1 matrix containing the DC offset (mean) of the
```

```
  wave in the "before stimulation" curve fit window.

"rampbefore" (optional) is a Nchans x 1 matrix containing the line-fit slope
  of the wave in the "before stimulation" curve fit window.

"magafter" is a Nchans x Nwindows matrix containing magnitudes of the
  dominant oscillation after stimulation for each channel and each
  requested window location.

"freqafter" is a Nchans x Nwindows matrix containing frequencies of the
  dominant oscillation after stimulation for each channel and each
  requested window location.

"phaseafter" is a Nchans x Nwindows matrix containing the phase (in
  radians) of the dominant oscillation after stimulation for each
  channel at each requested window midpoint.

"meanafter" is a Nchans x Nwindows matrix containing the DC offset (mean)
  of the wave in each "after stimulation" curve fit window.

"rampafter" (optional) is a Nchans x Nwindows matrix containing the line-fit
  slope of the wave in each "after stimulation" curve fit window.

"relafter" is a Nchans x Nwindows matrix containing relative magnitudes
  of the dominant oscillation after stimulation dividied by the magnitude
  before stimulation, for each channel and each requested window
  location. If the oscillation before stimulation was below-threshold,
  NaN is recorded instead of the relative magnitude.



Debugging fields that may be provided (if requested):

"origtimebefore" is a 1 x Nsamples matrix containing timestamps from the
  "before stimulation" curve fit window.

"origwavebefore" is a Nchans x Nsamples matrix with waveform data from the
  "before stimulation" curve fit window.

"rawphasebefore" is a Nchans x 1 matrix storing the phase returned by
  nlProc_fitCosine() before adjustment.

"origtimeafter" is a cell array with one entry per window size, containing
  1 x Nsamples matrices holding timestamps from the "after stimulation"
  curve fit windows.

"origwaveafter" is a cell array with one entry per window size, containing
  Nchans x Nsamples matrices holding waveform data from the "after
  stimulation" curve fit windows.
```

"rawphaseafter" is a cell array with one entry per window size, containing
  Nchans x 1 matrices storing the phase returned by nlProc_fitCosine()
  before adjustment.

Normalized data fields that are generated by test scripts:

"magafternormrand" and "magafterbaserand" contain magnitudes of the dominant
  oscillation after phase-specific stimulation normalized against the
  corresponding random-phase stimulation cases.

"magafternormphase" and "magafterbasephase" contain magnitudes of the dominant
  oscillation after stimulation at one specific phase (typically 180 degrees)
  normalized against another specific phase (typically 0 degrees).

"magafternormcurrent" and "magafterbasecurrent" contain magnitudes of the
  dominant oscillation after stimulation at higher current normalized against
  lower current (typically sham if available).

"relafternormcurrent" and "relafterbasecurrent" contain the after-vs-before
  relative oscillation magnitudes at higher current normalized against the
  relative oscillation magnitudes at lower current.

== Fields for multi-unit activity:

Raw statistics provided by euChris_extractStimMUAResponse:

"meanbefore" is a Nchans x 1 vector holding the mean of the MUA before
  stimulation.

"meanafter_list" is a Nchans x Nwindows matrix containing means of the MUA
  after stimulation.

"devbefore" is a Nchans x 1 vector holding the standard deviation of the
  MUA before stimulation.

"devafter_list" is a Nchans x Nwindows matrix containing standard deviations
  of the MUA after stimulation.

Derived statistics provided by euChris_extractStimMUAResponse:

"basemult_list" is a Nchans x Nwindows matrix containing
  meanafter / meanbefore. This is the relative increase in background

activity.

"devmult_list" is a Nchans x Nwidnows matrix containing
devafter / devbefore. This is the relative increase in activity
variability.

"zbaseshift_list" is a Nchans x Nwindows matrix containing
(meanafter - meanbefore) / devbefore. This is the z-scored _displacement_
in background activity.

If noise is negligible and spiking activity is the dominant component of
the background, then "basemult" is a z-scored measure of activity change.

If noise is the dominant component of the background and real spiking
activity is intermittent, "devmult" is a z-scored measure of activity change.

If the noise component of the background is the same before and after
stimulation, and if spiking activity is a significant component of the
background, then "zbaseshift" is a z-scored measure of activity change.

Normalized data fields that are generated by test scripts:

These fields have names with the form:
{ basemult | devmult | zbaseshift }{ norm | base }{ rand | phase | current }

"basemultFOO", "devmultFOO", and "zbaseshiftFOO" are derived from the
corresponding statistics described above.

"FOOnormCASE" and "FOObaseCASE" are the normalized statistic value and the
baseline statistic value (used to compute the normalized value),
respectively.

"FOOrand" uses the random-phase test case as the baseline, dividing other
phase cases' statistic values by the random-phase case's value.
"FOOphase" uses the "n90" phase test case as the baseline, dividing the
other fixed-phase cases' statistic values by the "n90" case's value.
"FOOcurrent" uses the sham case as the baseline, dividing the other current
cases' statistic values by the sham case's value.

(This is the end of the file.)

4.10 CHRISSTIMRESPONSE.txt

```
Stimulation response data is a structure containing the following fields.
This can be used for alignment from any type of event, not just stimulation:


"ftdata_wb" is a Field Trip data structure containing the wideband data
  after artifact rejection and notch filtering, segmented into trials
  triggered by stimulation events.
"ftdata_lfp" (optional) is a Field Trip data structure containing broad-band
  LFP data, segmented into trials triggered by stimulation events.
"ftdata_band" (optional) is a Field Trip data structure containing narrow-band
  LFP data, segmented into trials triggered by stimulation events.
"ftdata_hp" (optional) is a Field Trip data structure containing high-pass
  spike waveforms, segmented into trials triggered by stimulation events.
"ftdata_mua" (optional) is a Field Trip data structure containing rectified
  band-pass spiking activity, segmented into trias triggered by stimulation
  events.

"tortecidx" is the index of the TORTE input channel in ftdata_XXX.
"extracidx" is a vector with indices of the hint-specified extra channels
  in ftdata_XXX.

"trainpos" is a vector with one entry per trial, holding the relative
  position of each trial in its associated event train (1 for the first
  event of a train, 2 for the next, and so forth).
"trainrevpos" is a vector with one entry per trial, holding the relative
  position of each trial with respect to the _end_ of its event train
  (1 for the last event of a train, 2 for the second-last, and so forth).


(This is the end of the file.)
```

4.11 SIGNALCONFIG.txt

```
A signal configuration structure is used by several of the
"euChris_extractXXXX" functions. This structure has the following fields:


Common:

"notch_freqs" is a vector of frequencies to notch filter (may be empty).
"notch_bandwidth" is the bandwidth of the notch filter.

"lfp_band" [ min max ] is the broad-band LFP frequency range. NOTE - Only
  the upper corner is used when filtering, since FT and Matlab both have
  trouble with very low frequency filters.
```

"spike_cutoff" is the corner frequency of the high-pass filter used to
  extract spike waveforms for spike shape analysis.

"mua_band" [min max] is the band-pass frequency range used to extract
  rectified spiking activity.
"mua_cutoff" is the corner frequency of the low-pass filter used to smooth
  rectified spiking activity.

"artifact_method" is a character vector specifying the artifact suppression
  method (per ARTIFACTCONFIG.txt).

"artparams_sigma", if present, is a configuration structure for standard
  deviation based artifact rejection, per ARTIFACTCONFIG.txt.

"artparams_expknown", if present, is a configuration structure for artifact
  rejection using exponential curve fits at known locations, per
  ARTIFACTCONFIG.txt.

"artparams_expguess", if present, is a configuration structure for artifact
  rejection using exponential curve fits at guessed locations, per
  ARTIFACTCONFIG.txt.

"squash_config", if present, is a configuration structure for NaN squashing
  and interpolation and step removal, per SQUASHCONFIG.txt.

For euChris_extractSignalsAnalog_loop2302:

"head_tail_trim_fraction" is the relative amount to trim from the head
  and tail of the data (as a fraction of the total length). This should be
  in the range of 0 to 0.5.
"canon_detect_phase_width_degrees" is the width of the response window
  to use when estimating what the phase detector signal should look like.

For euChris_extractStimResponses_loop2302:

Artifact suppression can be specified for this function.

(This is the end of the file.)

Chapter 5

"euChris" Functions

5.1 euChris_calcNormalizedFeatureResponse.m

```
% function newfeatures = euChris_calcNormalizedFeatureResponse( ...
%   oldfeatures, oldfield, normfield, baselinefield, ...
%   case_compare_lut, casefield, matchfields )
%
% This function normalizes responses from the specified test cases against
% baseline responses from different test cases.
%
% An example use-case is normalizing phase-specific responses against
% the random-phase response.
%
% "oldfeatures" is a cell array containing one or more stimulation response
%   feature extraction structures, per CHRISSTIMFEATURES.txt.
% "oldfield" is the name of the structure field containing data to normalize.
% "normfield" is the name of the normalized structure field to create, or ''
%   to not store normalized data.
% "baselinefield" is the name of the field to store a copy of the baseline
%   data in, or '' to not store baseline data.
% "case_compare_lut" is a 2xNcases cell array. The first row contains labels
%   of test cases to be normalized, and the second row contains corresponding
%   labels of baseline cases to normalize against.
% "casefield" is the name of the field containing the case label.
% "matchfields" is a cell array containing zero or more field names (such as
%   'session' or 'probe')that have to match between cases that are normalized
%   or averaged.
%
% "newfeatures" is a copy of "oldfeatures". Responses that could be
%   normalized have a new field added (with the specified name). This is a
%   matrix with the same dimensions as the source field containing the
%   that record's response normalized by the baseline case's response
%   (averaged across compatible records).
```

5.2 euChris_evalTorte.m

```
% function tortefoms = euChris_evalTorte( signaldata, evalconfig )
%
% This function compiles figure-of-merit statistics for TORTE's estimation
% of magnitude and phase.
%
% "signaldata" is a structure returned by euChris_extractSignals_loop2302(),
%   with fields as described in CHRISSIGNALS.txt.
% "evalconfig" is a structure specifying analysis parameters. Missing fields
%   are set to reasonable default values. Fields and defaults are:
%   "resample_ms" (default: 1.0) is the sampling interval for comparing
%     estimated and ground truth waves (which were recorded from different
%     nodes, and so need to be aligned and resampled).
%   "average_tau" (default: 10.0) is the smoothing time constant used for
%     computing the (acausal) running average of magnitude.
%   "magcategory_edges" (default: [ 0.1 0.3 0.7 1.5 3.0 10.0 ]) is a set of
%     bin edges used for binning by average-normalized canon magnitude
%     before computing magnitude estimation statistics.
%   "phasecategory_count" (default: 8) is the number of phase bins to use
%     when binning canon phase before computing phase estimation statistics.
%
% "tortefoms" is a structure with the following fields:
%   "canon_v_torte_mag_ideal" is a vector containing sampled acausal ground
%     truth magnitudes, normalized to the ground truth running average.
%   "canon_v_torte_mag_torte" is a vector containing sampled estimated
%     magnitudes, normalized to the acausal ground truth running average.
%   "canon_v_torte_mag_rel_error" is a vector containing sampled relative
%     error of estimated magnitude vs acausal ground truth magnitude.
%   "canon_v_torte_mag_cat_vals" is a vector containing magnitude error bin
%     midpoint values corresponding to each of these relative error samples.
%   "canon_v_torte_phase_error_deg" is a vector containing sampled phase
%     error of estimated phase vs acausal ground truth phase.
%   "canon_v_torte_phase_cat_vals" is a vector containing phase error bin
%     midpoint values corresponding to each of these phase error samples.
%   "delayed_v_torte_mag_ideal" is a vector containing sampled delayed
%     (causal) ground truth magnitudes, normalized to the ground truth
%     running average.
%   "delayed_v_torte_mag_torte" is a vector containing sampled estimated
%     magnitudes, normalized to the delayed (causal) ground truth average.
%   "delayed_v_torte_mag_rel_error" is a vector containing sampled relative
%     error of estimated magnitude vs delayed (causal) ground truth magnitude.
%   "delayed_v_torte_mag_cat_vals" is a vector containing magnitude error bin
%     midpoint values corresponding to each of these relative error samples.
%   "delayed_v_torte_phase_error_deg" is a vector containing sampled phase
%     error of estimated phase vs delayed (causal) ground truth phase.
%   "delayed_v_torte_phase_cat_vals" is a vector containing phase error bin
%     midpoint values corresponding to each of these phase error samples.
```

5.3 euChris_extractSignalsAnalog_loop2302.m

```
% function casesignals = euChris_extractSignals_loop2302( ...
%   oldcasesignals, expmeta, signalconfig, verbosity )
%
% This function reads saved analog signals associated with one experiment
% and computes derived signals.
%
% This function works with 'loop2302' type experiments.
%
% "oldcasesignals" is a structure containing some or all of the signals
%   described in CHRISSIGNALS.txt. New signals are added to this.
% "expmeta" is one of the metadata structures returned by
%   euChris_getChrisMetadata(), with the fields described in CHRISEXPMETA.txt
%   (including the "casemeta" additional field).
% "signalconfig" is a structure with the fields noted in SIGNALCONFIG.txt,
%   including the following:
%   - Trimming configuration.
%   - Notch filtering configuration.
%   - LFP band specification.
%   - Phase detection width for estimating ground-truth phase detection.
%   - Artifact suppression settings.
%   - Squash settings.
% "verbosity" is 'normal' or 'quiet'.
%
% "casesignals" is a copy of "oldcasesignals" with the following signals
%   added, per CHRISSIGNALS.txt:
%
%   "wb_time" and "wb_wave" are the wideband signal.
%   "lfp_time" and "lfp_wave" are the wide-cut LFP signal.
%   "band_time" and "band_wave" are the ideal (acausal) narrow-band signal.
%   "delayband_time" and "delayband_wave" are a version of the narrow-band
%     signal produced with a causal filter. This won't match TORTE's filter.
%
%   "canon_time", "canon_mag", "canon_phase", and "canon_rms" are derived
%     from the acausal band-pass signal. These are the time, analytic
%     magnitude and phase, and the acausal moving RMS average of the
%     magnitude, respectively.
%   "delayed_time", "delayed_mag", "delayed_phase", and "delayed_rms"
%     are derived from the delayed (causal) band-pass signal. These are the
%     time, analytic magnitude and phase, and delayed (causal) moving RMS
%     average of the magnitude, respectively.
%   "canon_magflag", "canon_phaseflag", "delayed_magflag", and
%     "delayed_phaseflag" are the magnitude excursion detection flag and
%     the phase target match flag derived from the acausal and delayed
%     (causal) signals described above.
%   "canon_magflag_edges", "canon_phaseflag_edges", "delayed_magflag_edges",
%     and "delayed_phaseflag_edges" are vectors holding timestamps of rising
%     edges of the corresponding magnitude detection flags and of the
```

```
%     delayed phase detection flag, and timestamps of the high pulse
%     midpoints of the acausal phase detection flag.
%
%   "torte_time", "torte_mag", and "torte_phase" are the recorded values
%     of the TNE Lab Phase Calculator plugin's estimates of instantaneous
%     magnitude and instantaneous phase.
%     NOTE - These signals are not guaranteed to exist!
%   "torte_wave" is a reconstruction of the narrow-band signal using
%     "torte_mag" and "torte_phase". This should look like "delayband_wave".
```

5.4 euChris_extractSignalsDigital_loop2302.m

```
% function casesignals = euChris_extractSignalsDigital_loop2302( ...
%   oldcasesignals, expmeta, signalconfig, verbosity )
%
% This function reads saved digital signals associated with one experiment
% and computes derived signals.
%
% This function works with 'loop2302' type experiments.
%
% "oldcasesignals" is a structure containing some or all of the signals
%   described in CHRISSIGNALS.txt. New signals are added to this.
% "expmeta" is one of the metadata structures returned by
%   euChris_getChrisMetadata(), with the fields described in CHRISEXPMETA.txt
%   (including the "casemeta" additional field).
% "signalconfig" is a structure with the following fields, per
%   SIGNALCONFIG.txt:
%   "notch_freqs" is a vector of frequencies to notch filter (may be empty).
%   "notch_bandwidth" is the bandwidth of the notch filter.
%   "artifact_suppression_level" is 0 for normal suppression, positive for
%     more suppression, or NaN to disable suppression.
%   "head_tail_trim_fraction" is the relative amount to trim from the head
%     and tail of the data (as a fraction of the total length).
%   "lfp_band" [ min max ] is the broad-band LFP frequency range.
%   "canon_detect_phase_width_degrees" is the width of the response window
%     to use when estimating what the phase detector signal should look like.
% "verbosity" is 'normal' or 'quiet'.
%
% "casesignals" is a copy of "oldcasesignals" with the following signals
%   added, per CHRISSIGNALS.txt:
%
%   "XXX_ftevents", "XXX_wave", "XXX_time", and "XXX_edges" are stored for
%     each of several TTL signals.
%     NOTE - These signals are not guaranteed to exist!
%     "XXX_ftevents" holds a Field Trip event structure array for events
%       associated with this TTL signal, per ft_read_event().
%     "XXX_wave" is a logical vector holding time-series waveform data for
%       this TTL signal.
```

```
%     "XXX_time" is a vector holding waveform timestamp data for this signal.
%     "XXX_edges" is a vector holding timestamps of rising signal edges.
%     Signals saved (values of "XXX") are "loopback", "detectmag",
%     "detectphase", "detectrand", "trigphase", "trigrand", "trigpower",
%     "trigimmed", and "trigused" (a duplicate of one of the other "trig"
%     signals).
```

5.5 euChris_extractStimMUAResponse.m

```
% function muafeatures = euChris_extractStimMUAResponse( ...
%   timeseriesbefore, trialdatabefore, timeseriesafter, trialdataafter, ...
%   mua_params, meta_fields )
%
% This performs feature extraction of multi-unit activity before and after
% stimulation, for one or more trials.
%
% For each trial, this finds the magnitude and standard deviation of the
% MUA in user-specified time windows before and after stimulation. Several
% derived statistics are computed, per CHRISMUAFEATURES.txt.
%
% One set of trials is used for curve-fitting before stimulation and another
% set of trials for curve-fitting after stimulation. These may be the same
% trials (to fit before and after each event), or may be trials corresponding
% to the first and last event in a train (to fit before and after the train).
%
% The number of trials and channels must be the same for both trial sets.
% Durations (sample counts) may vary.
%
% "timeseriesbefore" is a 1xNtrials cell array containing time series for
%   trials to fit before stimulation.
% "trialdatabefore" is a 1xNtrials cell array containing Nchans x Nsamples
%   matrices of waveform data for trials to fit before stimulation.
% "timeseriesafter" is a 1xNtrials cell array containing time series for
%   trials to fit after stimulation.
% "trialdataafter" is a 1xNtrials cell array containing Nchans x Nsamples
%   matrices of waveform data for trials to fit after stimulation.
% "mua_params" is a structure containing the following fields,
%   per CHRISMUAPARAMS.txt:
%   "time_window_ms" is the duration in milliseconds of the time windows
%     used for extracting average statistics.
%   "time_before_ms" is a timestamp in milliseconds specifying where the
%     middle of the "before stimulation" time window should be.
%   "timelist_after_ms" is a vector containing timestamps in milliseconds
%     specifying where the middle of the "after stimulation" time windows
%     should be.
% "meta_fields" is a structure with arbitrary fields. Each output structure
%   in "muafeatures" is initialized with a copy of "meta_fields".
%
```

```
% "muafeatures" is a 1xNtrials cell array. Each cell contains a copy of the
%   "meta_fields" structure with the following fields added, per
%   CHRISMUAFEATURES.txt:
%   "trialnum" is the trial number.
%   "winbefore" is a scalar containing the timestamp in seconds of the
%     midpoint of the before-stimulation time window.
%   "winafter" is a 1 x Nwindows vector containing the timestamps in seconds
%     of the midpoints of the after-stimulation time windows.
%   "meanbefore" is a Nchans x 1 vector holding the mean of the MUA before
%     stimulation.
%   "meanafter_list" is a Nchans x Nwindows matrix containing means of the
%     MUA after stimulation.
%   "devbefore" is a Nchans x 1 vector holding the standard deviation of
%     the MUA before stimulation.
%   "devafter_list" is a Nchans x Nwindows matrix containing standard
%     deviations of the MUA after stimulation.
%   "basemult_list" is a Nchans x Nwindows matrix containing
%     meanafter / meanbefore. This is the relative increase in background
%     activity.
%   "devmult_list" is a Nchans x Nwindows matrix containing
%     devafter / devbefore. This is the relative increase in activity
%     _variability_.
%   "zbaseshift_list" is a Nchans x Nwindows matrix containing
%     (meanafter - meanbefore) / devbefore. This is the z-scored
%     _displacement_ in background activity.
%
% If noise is negligible and activity dominates the background, basemult is
% a z-scored measure of activity change.
% If noise dominates the background and real activity is intermittent,
% devmult is a z-scored measure of activity change.
% If noise is the same before and after stimulation and real activity is a
% significant part of the background, then zbaseshift is a z-scored measure
% of activity change.
```

5.6 euChris_extractStimOscillationResponse.m

```
% function oscfeatures = euChris_extractStimOscillationResponse( ...
%   timeseriesbefore, trialdatabefore, timeseriesafter, trialdataafter, ...
%   oscfit_params, meta_fields )
%
% This performs feature extraction of oscillations before and after
% stimulation, for one or more trials.
%
% For each trial, this finds the dominant oscillation frequency within a
% specified time window, and then curve-fits oscillation magnitude at or
% near that frequency within one time window before stimulation and
% multiple time windows after stimulation.
%
```

```
% One set of trials is used for curve-fitting before stimulation and another
% set of trials for curve-fitting after stimulation. These may be the same
% trials (to fit before and after each event), or may be trials corresponding
% to the first and last event in a train (to fit before and after the train).
%
% The number of trials and channels must be the same for both trial sets.
% Durations (sample counts) may vary.
%
% "timeseriesbefore" is a 1xNtrials cell array containing time series for
%   trials to fit before stimulation.
% "trialdatabefore" is a 1xNtrials cell array containing Nchans x Nsamples
%   matrices of waveform data for trials to fit before stimulation.
% "timeseriesafter" is a 1xNtrials cell array containing time series for
%   trials to fit after stimulation.
% "trialdataafter" is a 1xNtrials cell array containing Nchans x Nsamples
%   matrices of waveform data for trials to fit after stimulation.
% "oscfit_params" is a structure containing the following fields,
%   per CHRISOSCPARAMS.txt:
%   "window_search" [ min max ] is a time range to look at when measuring
%     the dominant oscillation frequency.
%   "freq_search" [ min max ] is the frequency range to search for the
%     dominant oscillation frequency.
%   "freq_drift" [ min max ] is the minimum and maximum multiple of the
%     dominant frequency to accept when curve-fitting in time windows.
%     E.g. [ 0.5 1.5 ] accepts from 0.5 * fdominant to 1.5 * fdominant.
%   "min_before_strength" is the minimum oscillation magnitude a channel
%     has to have, as a fraction of the strongest channel's magnitude, to
%     be considered to be oscillating before stimulation.
%   "window_lambda" is the width of the curve-fitting time window, as a
%     multiple of the dominant oscillation wavelength.
%   "time_before" is the desired time of the middle of the before-stimulation
%     curve-fitting window.
%   "timelist_after" is a vector containing desired times of the middle of
%     after-stimulation curve fitting windows.
%   "use_line_fit" (optional) is true to subtract a line fit before doing
%     the cosine fit, and false or absent to just subtract the mean.
%   "debug_save_waves" is true to save raw signals used for curve fits.
% "meta_fields" is a structure with arbitrary fields. Each output structure
%   in "oscfeatures" is initialized with a copy of "meta_fields".
%
% "oscfeatures" is a 1xNtrials cell array. Each cell contains a copy of the
%   "meta_fields" structure with the following fields added, per
%   CHRISSTIMFEATURES.txt:
%   "trialnum" is the trial number.
%   "oscfreq" is the dominant frequency detected in the trial.
%   "winbefore" is a scalar containing the timestamp of the midpoint of the
%     before-stimulation time window.
%   "magbefore" is a Nchans x 1 matrix containing magnitudes of the dominant
%     oscillation before stimulation for each channel.
%   "freqbefore" is a Nchans x 1 matrix containing frequencies of the dominant
```

```
%     oscillation before stimulation for each channel.
%   "phasebefore" is a Nchans x 1 matrix containing the phase (in radians) of
%     the dominant oscillation before stimulation for each channel at the
%     window midpoint.
%   "meanbefore" is a Nchans x 1 matrix containing the DC offset (mean) of the
%     wave in the "before stimulation" curve fit window.
%   "rampbefore" (optional) is a Nchans x 1 matrix containing the line-fit
%     slope of the wave in the "before stimulation" curve fit window.
%   "winafter" is a 1 x Nwindows matrix containing the timestamps of the
%     midpoints of the after-stimulation time windows.
%   "magafter" is a Nchans x Nwindows matrix containing magnitudes of the
%     dominant oscillation after stimulation for each channel and each
%     requested window location.
%   "freqafter" is a Nchans x Nwindows matrix containing frequencies of the
%     dominant oscillation after stimulation for each channel and each
%     requested window location.
%   "phaseafter" is a Nchans x Nwindows matrix containing the phase (in
%     radians) of the dominant oscillation after stimulation for each
%     channel at each requested window midpoint.
%   "meanafter" is a Nchans x Nwindows matrix containing the DC offset (mean)
%     of the wave in each "after stimulation" curve fit window.
%   "rampafter" (optional) is a Nchans x Nwindows matrix containing the
%     line-fit slope of the wave in each "after stimulation" curve fit window.
%   "relafter" is a Nchans x Nwindows matrix containing relative magnitudes
%     of the dominant oscillation after stimulation dividied by the magnitude
%     before stimulation, for each channel and each requested window
%     location. If the oscillation before stimulation was below-threshold,
%     NaN is recorded instead of the relative magnitude.
```

5.7 euChris_extractStimResponses_loop2302.m

```
% function responsedata = euChris_extractStimResponses_loop2302( ...
%   expmeta, signalconfig, trigtimes, trig_window_ms, train_gap_ms, ...
%   chans_wanted, want_lfp, want_narrowband, want_highpass, want_mua, ...
%   verbosity )
%
% This function reads ephys data in segments centered around stimulation
% events.
%
% This function works with 'loop2302' type experiments.
%
% Wideband data has artifact regions replaced with NaN. Broad-band LFP and
% narrow-band LFP have smoothed replacement segments in these regions.
%
% "expmeta" is one of the metadata structures returned by
%   euChris_getChrisMetadata(), with the fields described in CHRISEXPMETA.txt
%   (including the "casemeta" additional field).
% "signalconfig" is a structure with fields noted in SIGNALCONFIG.txt,
```

```
%   including the following:
%   - Notch filtering configuration.
%   - LFP band configuration.
%   - Spike and MUA configuration.
%   - Artifact suppression settings.
%   - Squash settings.
% "trigtimes" is a vector containing trigger timestamps in seconds. If this
%   is [], the trials' t=0 times are used.
% "trig_window_ms" [ start stop ] is the window around stimulation events
%   to save, in milliseconds. E.g. [ -100 300 ].
% "train_gap_ms" is a duration in milliseconds. Stimulation events with this
%   separation or less are considered to be part of a pulse train.
% "chans_wanted" is either a character array or a cell array, per
%   CHRISBATCHDEFS.txt. If it's a character array, it's 'trig' to read the
%   trigger channel, 'hint' to read the hint channels (if any; trigger
%   channel if not), and 'all' to read all ephys channels. If it's a cell
%   array, it's treated as a list of FT channels to read.
% "want_lfp" is true if the broad-band LFP is to be extracted.
% "want_narrowband" is true if the narrow-band LFP is to be extracted.
% "want_highpass" is true if the high-pass spike waveform is to be extracted.
% "want_mua" is true if the rectified band-pass spiking activity is to be
%   extracted.
% "verbosity" is 'normal' or 'quiet'.
%
% "responsedata" is a structure containing the following fields, per
%   CHRISSTIMRESPONSE.txt:
%
%   "ftdata_wb" is a Field Trip data structure containing the wideband data
%     after artifact rejection and notch filtering. This contains NaN spans.
%   "ftdata_lfp" is a Field Trip data structure with the broad-band LFP data.
%     This is only present if "want_lfp" was true.
%   "ftdata_band" is a Field Trip data structure with the narrow-band LFP
%     data. This is only present if "want_narrowband" was true.
%   "ftdata_hp" is a Field Trip data structure with the high-pass spike data.
%     This is only present if "want_highpass" was true.
%   "ftdata_mua" is a Field Trip data structure with the rectified spike
%     activity data. This is only present if "want_mua" was true.
%
%   "tortecidx" is the index of the TORTE input channel in ftdata_XXX.
%   "extracidx" is a vector with indices of hint channels in ftdata_XXX.
%
%   "trainpos" is a vector with one entry per trial, holding the relative
%     position of each trial in an event train (1 for the first event of
%     a train, 2 for the next, and so forth).
%   "trainrevpos" is a vector with one entry per trial, holding the relative
%    position of each trial with respect to the _end_ of its event train
%    (1 for the last event of a train, 2 for the second-last, and so forth).
```

5.8 euChris_fillDefaultsEvalTorte.m

```
% function newconfig = euChris_fillDefaultsEvalTorte( oldconfig )
%
% This function fills in any missing fields in the supplied configuration
% structure.
%
% This function builds configuration structures for euChris_evalTorte().
%
% "oldconfig" is the structure to augment. It may be struct() or struct([]).
%
% "newconfig" is a copy of "oldconfig" with any missing fields added and set
%   to reasonable default values.
```

5.9 euChris_getArtifactConfigFromSignalConfig.m

```
% function [ artmethod artconfig ] = ...
%   euChris_getArtifactConfigFromSignalConfig( signalconfig )
%
% This extracts the appropriate artifact rejection configuration structure
% from a signal configuration structure.
%
% "signalconfig" is a signal configuration structure, per SIGNALCONFIG.txt.
%
% "artmethod" is an artifact rejection method, per ARTIFACTCONFIG.txt.
% "artconfig" is an artifact rejection configuration structure, per
%   ARTIFACTCONFING.txt.
```

5.10 euChris_getChanBatchDefs.m

```
% function [ batchdefs batchlabels batchtitles ] = ...
%   euChris_getChanBatchDefs( hintdata, want_batch_chans )
%
% This processes case definition hint data, building a list of channel
% groups (batches) to process.
%
% "hintdata" is a hint structure from a case definition (per
%   CHRISCASEMETA.txt).
% "want_batch_chans" is true to return channel lists based on the
%   "chanbatches" hint field and false otherwise (just returning trigger
%   channel and the channels decribed in the "extrachans" hint field).
%   If batches are wanted but none are defined, 'all' is returned.
%
% "batchdefs" is a cell array containing batch definitions, per
%   CHRISBATCHDEFS.txt. Each batch definition is either a cell array or a
```

```
%   character vector. If it's a cell array, it contains a list of names of
%   channels to be read. If it's a character vector, it's 'trig', 'hint', or
%   'all'.
% "batchlabels" is a cell array containing filename-safe labels for each
%   batch.
% "batchtitles" is a cell array containing human-readable plot-safe names
%   for each batch.
```

5.11 euChris_getChrisMetadata.m

```
% function casemetalist = euChris_getChrisMetadata( caselist )
%
% This iterates through a list of experiment cases for Chris's tests, and
% extracts signal chain metadata and Open Ephys metadata for each case.
%
% "caselist" is a struct array with the following fields (per
%   CHRISCASEMETA.txt):
%   'folder' is a character array with the top-level save file path.
%   'prefix' is a filename-safe label for per-case filenames.
%   'title' is a plot-safe human-readable per-case label.
%   'setprefix' is a filename-safe label for the set of cases this case
%     belongs to.
%   'settitle' is a plot-safe human-readable title for the set of cases
%     this case belongs to.
%   'exptype' is a character array specifying the parser to use for
%     processing this configuration, or '' for the default.
%   'hint' is a structure containing hints for parsing. This may be a
%     structure with no fields or an empty structure array.
%
% "casemetalist" is a cell array with one element per case, containing
%   the metadata structures returned when parsing each case. These have
%   the fields defined in CHRISEXPMETA.txt, augmented with a "casemeta"
%   field containing the associated case's metadata per CHRISCASEMETA.txt.
```

5.12 euChris_getCookedMetadata_loop2302.m

```
% function cookedmeta = euChris_getCookedMetadata_loop2302( rawmeta, hintdata )
%
% This function accepts raw configuration metadata for a "loop2302"
% experiment and produces derived (cooked) metadata.
%
% "rawmeta" is a metadata structure for the Open Ephys v5 folder
%   associated with the experiment, per RAWFOLDERMETA.txt.
% "hintdata" is a structure containing hints for processing metadata. This
%   may be a structure with no fields or an empty structure array.
%
```

```
% "cookedmeta" is a metadata structure containing derived configuration
% information, per CHRISCOOKEDMETA.txt.
```

5.13 euChris_getExpConfig_loop2302.m

```
% function [ config summary details diagmsgs errmsgs ] = ...
%   euChris_getExpConfig_loop2302( rawmetalist, cookedmeta, hintdata )
%
% This examines an experiment session's metadata and builds an experiment
% configuration structure describing the experiment.
%
% This function works with 'loop2302' type metadata.
%
% "rawmetalist" is a cell array containing per-folder raw metadata
%   structures for this experiment, per RAWFOLDERMETA.txt.
% "cookedmeta" is the cooked (derived) metadata for this experiment, per
%   CHRISEXPMETA.txt.
% "hintdata" is a structure containing hints for processing metadata. If
%   there are no hints, this will be a structure with no fields.
%
% "config" is a structure describing the experiment configuration, per
%   CHRISEXPCONFIGS.txt. It will be empty if parsing failed.
% "summary" is a cell array of character vectors containing a short
%   human-readable summary of the configuration.
% "details" is a cell array of character vectors containing a
%   human-readable detailed description of the configuration.
% "diagmsgs" is a cell array containing diagnostic messages _and_ error
%   and warning messages generated during processing.
% "errmsgs" is a cell array containing _only_ error and warning messages
%   generated during processing.
```

5.14 euChris_getResponseDataMetadata.m

```
% function datameta = euChris_getResponseDataMetadata( statdata, labelfields )
%
% This extracts many pieces of metadata from a series of stimulation response
% feature statistics structures.
%
% NOTE - window times are expected to be consistent across all records!
% Window metadata content may be incorrect if this is not the case.
%
% NOTE - If requested label fields are absent, associated metadata fields
% will be absent or empty.
%
% "statdata" is a cell array containing stimulation response feature data
%   structures, per CHRISSTIMFEATURES.txt, typically also including
```

```
%   session/case/probe labels.
% "labelfields" is a cell array containing label field names to collect
%   metadata for. Typical names include 'sessionlabel', 'caselabel', and
%   'probelabel'.
%
% "datameta" is a structure with the following fields:
%
%   "winbefore" is a scalar containing the timestamp of the midpoint of the
%     before-stimulation time window.
%   "winafter" is a vector containing the timestamps of the midpoints of the
%     after-stimulation time windows.
%   "winbeforetext" is a character vector containing a plot-safe
%     human-readable version of the absolute value of the "winbefore" time.
%   "winbeforelabel" is a character vector containing a filename-safe
%     label derived from the absolute value of the "winbefore" time.
%   "winaftertext" is a cell array containing character vectors with
%     plot-safe human-readable versions of the "winafter" times.
%   "winafterlabels" is a cell array containing character vectors with
%     filename-safe labels derived from the "winafter" times.
%
%   "labelraw" is a structure with one field per entry in "labelfields",
%     holding cell arrays with lists of all raw label values encountered.
%   "labeltext" is a structure with one field per entry in "labelfields",
%     holding cell arrays containing plot-safe human-readable versions of
%     the raw labels.
%   "labelshort" is a structure with one field per entry in "labelfields",
%     holding cell arrays containing plot-safe versions of the raw labels.
%   "labelkey" is a structure with one field per entry in "labelfields",
%     holding cell arrays containing versions of the raw labels that are
%     safe to use as structure field names.
```

5.15 euChris_makeBatchChanLabels.m

```
% function chanlabels = euChris_makeBatchChanLabels( chanpattern, channums )
%
% This produces a cell array containing channel labels, by iterating
% through a list of channel numbers and applying a sprintf pattern.
%
% "chanpattern" is a spritnf pattern for channel names that accepts a
%   channel number as an argument.
% "channums" is a vector containing channel numbers to format.
%
% "chanlabels" is a cell array containing formatted channel labels.
```

5.16 euChris_parseCaseLabel_loop2302.m

```
% function [ band current phase ] = ...
%   euChris_parseCaseLabel_loop2302( caselabel )
%
% This attempts to parse a case label ("prefix" in the set definitions)
% using Chris's 2023 naming conventions.
%
% These have the form: (band)(current)(phase)
%
% (band) is letters, (current) is digits, and (phase) is both.
%
% If (band) is omitted, it's assumed to be beta.
% Target phases of 0 were recorded as both "cal" and "0deg".
% Current is in uA, and is converted to a number. Other values are kept as
% character vectors.
%
% "caselabel" is a character vector holding the label to convert.
%
% "band" is a character vector indicating band.
% "current" is a number indicating the stimulation current in uA.
% "phase" is a character vector indicating target phase.
%
% Values that couldn't be extracted are returned as '' (char) or NaN (number).
```

5.17 euChris_parseExperimentConfig.m

```
% function [ expmeta errmsgs ] = ...
%   euChris_parseExperimentConfig( rawmetalist, exptype, hintdata )
%
% This is a top-level entry point for parsing configuration data for
% one of Chris's experiment runs.
%
% "rawmetalist" is a cell array containing metadata structures returned by
%   euHLev_getAllMetadata_XXX, per RAWFOLDERMETA.txt.
% "exptype" is a label defining the type of dataset being processed.
%   E.g.: 'loop2302'
% "hintdata" is a structure containing hints for processing metadata. This
%   may be a structure with no fields or an empty structure array.
%
% "expmeta" is a structure containing aggregated metadata for this run.
%   This is an empty structure if the run metadata couldn't be parsed.
%   Detailed contents are per "CHRISEXPMETA.txt".
% "errmsgs" is a cell array containing warning messages and error messages
%   generated while parsing.
```

5.18 euChris_parseSetLabel_loop2302.m

```
% function [ datenum suffix ] = euChris_parseSetLabel_loop2302( setlabel )
%
% This attempts to parse a set label ("setprefix" in the set definitions)
% using Chris's 2023 naming convention.
%
% These have the form: (date)(suffix)
%
% (date) is digits, (suffix) is letters.
% For tests that didn't repeat, (suffix) is omitted.
%
% "setlabel" is a character vector holding the label to convert.
%
% "datenum" is a number indicating the date the dataset was recorded.
% "suffix" is a character vector indicating which cycle or session from
%   that date this set belongs to.
%
% Values that couldn't be extracted are returned as '' (char) or NaN (number).
```

5.19 euChris_plotCaseDataBoxes.m

```
% function euChris_plotCaseDataBoxes( ...
%   casedata, plotpercase, yfield, xfield, ...
%   casetitles, caselabels, plottype, xstr, ystr, ...
%   titleprefix, titlesuffix, fnameprefix, fnamesuffix )
%
% This makes a series of box plots with individual and aggregated data.
%
% This is intended to work with the output of "euChris_evalXXX" functions,
% which generate structures with various statistics stored as fields. This
% functiton is passed a cell array of such structures (for different
% experiment cases), and plots individual and aggregate statistics.
%
% "casedata" is a cell array containing per-case structures with data.
% "plotpercase" is true if per-case plots are to be generated in additon to
%   combined/aggregated plots.
% "yfield" is the structure field name containing data values to plot.
% "xfield" is the structure field name containing per-sample category/bin
%   values/labels. This is a vector for numeric values and a cell array for
%   labels.
% "casetitles" is a cell array containing plot-safe human-readable case titles.
% "caselabels" is a cell array containing filename-safe per-case labels.
% "plottype" is a label giving hints about how to set up the plot. Use '' for
%   defaults. Known values are: 'tortemagrel', 'tortemagabs', 'tortphase'
% "xstr" is a character vector containing the X axis label.
% "ystr" is a character vector containing the Y axis label.
```

```
% "titleprefix" is a character vector with plot-safe text to add before the
%   case title (including whitespace and delimiters).
% "titlesuffix" is a character vector with plot-safe text to append after the
%   case title (including whitespace and delimiters).
% "fnameprefix" is a character vector with filename-safe text to add before
%   the case label.
% "fnamesuffix" is a character vector with filename-safe text to append after
%   the case label.
%
% No return value.
```

5.20 euChris_plotOscillationFits.m

```
% function euChris_plotOscillationFits( ...
%   oscfitdata, oscparams, trialtimes, trialwaves, timeswanted, ...
%   window_sizes, size_labels, max_count_per_size, titleprefix, fnameprefix )
%
% This generates waveform plots of stimulation events and oscillation fits
% near those events.
%
% "oscfitdata" is a cell array with one entry per trial. Each cell contains
%   a structure with detected oscillation features, per CHRISSTIMFEATURES.txt.
%   If the "chanlabels" field is present, it's used.
% "oscparams" is a structure containing oscillation fit parameters, per
%   CHRISOSCPARAMS.txt.
% "trialtimes" is a cell array with one entry per trial, containing the
%   timestamp series for each trial.
% "trialwaves" is a cell array with one entry per trial. Each cell contains
%   a Nchans x Nsamples matrix with waveform data.
% "timeswanted" is 'before', 'after', or 'both' (before/after stimulation).
% "window_sizes" is a cell array. Each cell contains a plot time range
%   [ begin end ] in seconds, or [] for the full data extent.
% "size_labels" is a cell array containing filename-safe labels used when
%   creating filenames and annotating titles for plots of each window size.
% "max_count_per_size" is a scalar indicating the maximum number of plots
%   to emit at a given size level. Set to inf to not decimate plots.
% "titleprefix" is the prefix used when generating figure titles.
% "fnameprefix" is the prefix used when generating output filenames.
```

5.21 euChris_plotResponseStatistics.m

```
% function euChris_plotResponseStatistics( ...
%   statdata, global_filter, plotdefs, aggr_only, fnameprefix )
%
% This plots statistics for changes in extracted features induced by
% stimulation.
```

```
%
% "statdata" is a cell array containing stimulation response feature data
%   structures, per CHRISSTIMFEATURES.txt, including labels for
%   session/case/probe and (if plotted) channel indices/labels.
% "global_filter" is a structure array containing a filter list, per
%   nlUtil_pruneStructureList(). This is applied to "statdata" before plotting.
% "plotdefs" is a structure array with the following fields:
%   "label" is a filename-safe identifier for this plot definition.
%   "type" is 'xy', 'box', 'line', or 'timeheat'.
%   "xaxis" is the statdata field to use for the independent axis.
%   "xtitle" is the title to use for the independent axis.
%   "yaxis" is the statdata field to use for the dependent axis.
%   "ytitle" is the title to use for the dependent axis.
%   "titleprefix" is a prefix to use when building plot titles.
%   "titlesuffixes" is a cell array which may include the following:
%     'before' emits "NN ms Before" in the title.
%     'after' emits "NN ms After" in the title.
%   "decorations" is a cell array which may include the following:
%     'diag' draws a diagonal line (typically for XY plots).
%     'hunity' draws a horizontal line at Y=1 (typically for relative data).
%     'hzero' draws a horizontal line at Y=0 (typically for absolute data).
%   "caseblacklist" is a cell array with case labels to ignore.
%   "casewhitelist" is a cell array; only cases in this list are plotted. If
%     this is {}, all cases may be plotted (subject to the blacklist).
% "aggr_only" is true to suppress per-session plots (only plotting aggregate).
% "fnameprefix" is a prefix used when building output filenames.
%
% No return value.
```

5.22 euChris_plotTorteWaves.m

```
% function euChris_plotTorteWaves( signaldata, window_sizes, size_labels, ...
%   max_count_per_size, want_debug_plots, titlebase, fnamebase )
%
% This generates several sets of time-series plots of TORTE's estimates of
% an ephys signal's analytic components, with detection and trigger times
% overlaid.
%
% "signaldata" is a structure returned by euChris_extractSignals_loop2302(),
%   with fields as described in CHRISSIGNALS.txt.
% "window_sizes" is a vector containing plot durations in seconds, stepped
%   across the time series.
% "size_labels" is a cell array containing filename-safe labels used when
%   creating filenames and annotating titles for plots of each window size.
% "max_count_per_size" is a scalar indicating the maximum number of plots
%   to emit at a given size level. Plots are spaced evenly within and are
%   centered on the full time span.
% "want_debug_plots" is true to render additional plots (against the
```

```
%   full-range LFP and against the raw wideband signal), and false otherwise.
% "titlebase" is a prefix to use when constructing human-readable titles.
% "fnamebase" is a prefix to use when constructing filenames.
%
% No return value.
```

5.23 euChris_reconstructOscCosine.m

```
% function [ recontime reconchanwaves ] = euChris_reconstructOscCosine( ...
%   wintime, winperiods, samprate, extendmethod, ...
%   oscmag, oscfreq, oscphase, oscmean, oscramp )
%
% This reconstructs per-channel cosine oscillations based on their detection
% parameters.
%
% This is mostly intended as a plotting aid.
%
% NOTE - Since there's only one time series (following Field Trip trial
% conventions), it's sized for the largest window (lowest frequency).
% Reconstructions past the requested number of periods may be extended or
% set to NaN, per "extendmethod".
%
% "wintime" is the timestamp of the middle of the reconstruction window.
% "winperiods" is the number of oscillation periods to reconstruct.
% "samprate" is the sampling rate for the reconstructed waves.
% "extendmethod" is 'extend' or 'crop', for windows larger than the desired
%   number of periods.
% "oscmag" is a Nchans x 1 vector with per-channel cosine magnitudes.
% "oscfreq" is a Nchans x 1 vector with per-channel cosine frequencies.
% "oscphase" is a Nchans x 1 vector with per-channel cosine phases at the
%   window midpoint.
% "oscmean" is a Nchans x 1 vector with per-channel DC offsets.
% "oscramp" (optional) is a Nchans x 1 vector with per-channel line-fit
%   slopes. If this is missing or [], a slope of 0 is used (no line fit).
%
% "recontime" is a 1 x Nsamples timestamp series.
% "reconchanwaves" is a Nchans x Nsamples set of reconstructed waveforms.
```

5.24 euChris_sortCasesBySet.m

```
% function setcases = euChris_sortCasesBySet( caselist )
%
% This accepts a structure array containing case definition structures
% (per CHRISCASEMETA.txt), and sorts them into multiple lists that each
% have the same "setprefix" field within the list.
%
```

```
% "caselist" is a structure array containing case definitions, per
%   CHRISCASEMETA.txt.
%
% "setcases" is a cell array. Each cell contains its own structure array
%   containing case definitions that share a common "setprefix" value.
%   While "setcases" may be empty (if "caselist" is empty), the lists
%   contained within "setcases" are guaranteed to be non-empty.
```

5.25 euChris_summarizeSignalsPresent_loop2302.m

```
% function [ summary details ] = ...
%   euChris_summarizeSignalsPresent_loop2302( sigstruct )
%
% This function generates a human-readable summary describing which signals
% are present or absent in a 'loop2302' extracted signals structure.
%
% "sigstruct" is a structure containing extracted signals, per
%   CHRISSIGNALS.txt.
%
% "summary" is a cell array containing character vectors holding lines of
%   text for a human-readable summary of which signals are present/absent.
% "details" is a cell array containing character vectors holding lines of
%   text for a detailed human-readable description of signals present/absent.
```

Chapter 6

"euFT" Notes

6.1 FT_ITERFUNC_DERIVED.txt

```
A derived-signal iteration processing function handle is called to perform
signal processing on wideband, LFP, spike, and MUA signals when iterating
across trials and channels within a Field Trip data structure.

As of 2023 this type of iterator is only used when processing FT data in
batches (to allow separation of FT and non-FT operations and to hide
channel batch logic).


A derived-signal iteration processing function has the form:

result = iterfunc_derived( ...
  wbseries, wbtimes, wbrate, lfpseries, lfptimes, lfprate, ...
  spikeseries, spiketimes, spikerate, muaseries, muatimes, muarate, ...
  trialidx, chanidx, chanlabel )

"wbseries" is a vector containing wideband samples (from FT's "trial").
"wbtimes" is a vector containing wideband sample times (from FT's "time").
"wbrate" is the wideband sampling rate (from FT's "fsample").
"lfpseries" is a vector containing low-pass-filtered LFP samples.
"lfptimes" is a vector containing low-pass-filtered LFP sample times.
"lfprate" is the low-pass-filtered LFP sampling rate.
"spikeseries" is a vector containing high-pass-filtered spike samples.
"spiketimes" is a vector containing high-pass-filtered spike sample times.
"spikerate" is the high-pass-filtered spike series sampling rate.
"muaseries" is a vector containing rectified activity samples.
"muatimes" is a vector containing rectified activity sample times.
"muarate" is the rectified multi-unit activity sampling rate.
"trialidx" is the trial number.
"chanidx" is the channel number.
"chanlabel" is the corresponding channel label (from the Field Trip data's
```

```
  "label" field).

"result" is an arbitrary data type containing the output of processing this
  trial and channel's data. This is typically a struct aggregating several
  different processing results.


A typical derived-signal iteration processing function definition would be
as follows. This example wraps a helper function that is passed additional
arguments set at the time the processing function is defined, and that only
operates on the LFP portion of the input data.

tuning_parameters = (stuff);
other_parameters = (stuff);
iterfunc_derived = @( ...
  wbseries, wbtimes, wbrate, lfpseries, lfptimes, lfprate, ...
  spikeseries, spiketimes, spikerate, muaseries, muatimes, muarate, ...
  trialidx, chanidx, chanlabel ) ...
  helper_do_iteration_processing( lfpseries, lfptimes, lfprate, ...
    trialidx, chanidx, chanlabel, tuning_parameters, other_parameters );


This is the end of the file.
```

Chapter 7

"euFT" Functions

7.1 euFT_addEventTimestamps.m

```
% function newsignals = ...
%   euFT_addEventTimestamps( oldsignals, samprate, samplabel, timelabel )
%
% This function processes a number of event tables or event structure arrays,
% adding a timestamp column or field derived from the "sample" column or
% field.
%
% "oldsignals" is a structure with zero or more fields. Each field contains
%   either a table with event data or a struct array with event data.
% "samprate" is the sampling rate to use when calculating timestamps.
% "samplabel" is the name of the column or field with the sample number. In
%   Field Trip event lists, this is "sample".
% "timelabel" is the name of the new column or field to add.
%
% "newsignals" is a copy of "oldsignals" where each table or struct array
%   is augmented with a column or field containing timestamps in seconds.
```

7.2 euFT_addTTLEventsAsCodes.m

```
% function newtable = ...
%   euFT_addTTLEventsAsCodes( oldtable, ttltable, timecol, codecol, codelabel )
%
% This adds TTL events to an event code event table.
%
% "oldtable" is the event code event table to add events to.
% "ttltable" is the event table containing TTL events.
% "timecol" is the name of the column containing timestamps. This must be
%   present in both tables.
% "codecol" is the name of the column in "oldtable" that contains code
```

```
%   identification labels.
% "codelabel" is the code identification label to use for added TTL events.
%
% "newtable" is a copy of "oldtable" with TTL events added. Columns from
%   "ttltable" that are present in "oldtable" are copied to the new rows;
%   columns in "ttltable" that are not in "oldtable" are discarded. Columns
%   in "oldtable" that are not in "ttltable" are set to NaN or {''} in the
%   new rows.
```

7.3 euFT_assembleEventWords.m

```
% function wordevents = ...
%   euFT_assembleEventWords( bitlabels, bitevents, wordlabel, firstbit )
%
% This assembles events associated with individual bit changes into a
% sequence of events associated with code word changes.
%
% Channel labels are assumed to end in the bit number. The named bit number
% is treated as the least-significant bit in the code word (this is usually
% 0 or 1, depending on channel label conventions).
%
% NOTE - This only works for LoopUtil events! Those have the channel names
% stored in the event records' "type" field.
%
% NOTE - This uses unsigned 64-bit event words.
%
% "bitlabels" is a cell array containing channel labels for individual bits.
% "bitevents" is an event structure array to search for bit-change events in.
% "wordlabel" is the value to store in the derived list's "type" field.
% "firstbit" is the channel bit number corresponding to the least-significant
%   bit in the output word. This is usually 0 or 1.
%
% "wordevents" is an event structure array containing word-change events.
```

7.4 euFT_defineTrialsUsingCodes.m

```
% function trl = euFT_defineTrialsUsingCodes( ...
%   eventtable, labelfield, timefield, samprate, padbefore, padafter, ...
%   startlabel, stoplabel, alignlabel, codemetatosave, codevaluefield )
%
% This function generates a Field Trip "trl" matrix from a USE event code
% table. The first three columns of "trl" are the starting sample, ending
% sample, and trigger offset (per ft_definetrial()). Remaining columns are
% metadata, which ft_preprocess() will move to a "trialinfo" matrix.
%
% The "trltab" table is a table containing the same data as "trl". This
```

```
% may be used to get column labels for "trl" (as metadata would otherwise
% be hard to identify).
%
% NOTE - The first three columns in 'trltab' are named 'sampstart',
% 'sampend', and 'sampoffset'. The original timestamps for the start, stop,
% and alignment events are saved in 'timestart', 'timeend', and 'timetrigger'.
%
% This can tolerate event labels that are numbers or character arrays.
%
% "eventtable" is a table containing event information. This must at minimum
%   contain event labels and timestamps.
% "labelfield" is the name of the event table column holding event labels.
% "timefield" is the name of the event table column holding timestamps.
% "samprate" is the sampling rate to use for converting timestamps into sample
%   indices.
% "padbefore" is the number of seconds to add before the trial-start event.
% "padafter" is the number of seconds to add after the trial-end event.
% "startlabel" is the event label that indicates the start of a trial.
% "stoplabel" is the event label that indicates the end of a trial.
% "alignlabel" is the event label that indicates the trigger to align trials
%    with.
% "codemetatosave" is a structure describing event code value information to
%    be saved as trial metadata. Each field is an output metadata column
%    name, and that field's value is the event label to look for. When that
%    event label is seen, the event value (from "valuefield") is saved as
%    metadata.
% "codevaluefield" is the name of the event table column holding event values.
%    This is only needed if "codemetatosave" is non-empty.
%
% "trl" is a Field Trip trial definition matrix, per ft_definetrial(). This
%   includes additional metadata columns.
% "trltab" is a table containing the same trial definition data as "trl",
%   with column names.
```

7.5 euFT_doBrickNotchRemoval.m

```
% function newdata = ...
%   euFT_doBrickNotchRemoval( olddata, notch_list, notch_bw )
%
% This performs band-stop filtering in the frequency domain by squashing
% frequency components (a "brick wall" filter). This causes ringing near
% large disturbances (a top-hat in the frequency domain gives a sinc
% function impulse response).
%
% NOTE - This uses the LoopUtil brick-wall filter implementation. To use
% Field Trip's implementation, call euFT_getFiltPowerBrick() to get a FT
% filter configuration structure.
%
```

```
% "olddata" is the FT data structure to process.
% "notch_list" is a vector containing notch center frequencies to remove.
% "notch_bw" is the width of the notch. All notches have the same width.
%
% "newdata" is a copy of "olddata" with trial data waveforms filtered.
```

7.6 euFT_doBrickPowerFilter.m

```
% function newdata = ...
%   euFT_doBrickPowerfilter( olddata, notch_freq, notch_modes, notch_bw )
%
% This performs band-stop filtering in the frequency domain by squashing
% frequency components (a "brick wall" filter). This causes ringing near
% large disturbances (a top-hat in the frequency domain gives a sinc
% function impulse response).
%
% NOTE - This uses the LoopUtil brick-wall filter implementation. To use
% Field Trip's implementation, call euFT_getFiltPowerBrick() to get a FT
% filter configuration structure.
%
% "olddata" is the FT data structure to process.
% "notch_freq" is the fundamental frequency of the family of notches.
% "notch_modes" is the number of frequency modes to remove (1 = fundamental,
%   2 = fundamental and first harmonic, etc).
% "notch_bw" is the width of the notch. Harmonics have the same width.
%
% "newdata" is a copy of "olddata" with trial data waveforms filtered.
```

7.7 euFT_getChannelNamePatterns.m

```
% function [ names_ephys names_digital names_stimcurrent names_stimflags ] = ...
%   euFT_getChannelNamePatterns()
%
% This function returns cell arrays of channel name patterns, suitable for
% use with ft_channelselection(). These will identify different types of
% channel in Open Ephys and Intan data read using the LoopUtil library.
%
% "names_ephys" contains patterns for analog ephys recording channels.
% "names_digital" contains patterns for TTL bit-line and word channels.
% "names_stimcurrent" has patterns for Intan stimulation current channels.
% "names_stimflags" hass patterns for Intan stimulation flag status channels.
```

7.8 euFT_getCodeWordEvent.m

```
% function [ evtable have_events ] = euFT_getCodeWordEvent( ...
%   namelut, wordsigname, bitsignames, firstbit, shiftbitsname, ...
%   headerlabels, allevents )
%
% This looks up channel label patterns for a given signal, looks up channels
% that match those patterns, picks appropriate channels, and finds events
% that are from these channels, merges them into event words, and, returns
% the result as a table.
%
% NOTE - This only works for LoopUtil events! Those have the channel names
% stored in the event records' "type" field.
%
% "namelut" is a structure indexed by signal name that has cell arrays of
%   Field Trip channel label specifiers (per ft_channelselection()), and
%   that also has a field storing the number of bits to shift code words.
% "wordsigname" is the class label to look for for whole-word data. This
%   should only match one Field Trip label.
% "bitsignames" is the wildcard class label to look for for single-bit data.
%   These are treated as word bits, and labels are assumed to end in the bit
%   number.
% "firstbit" is the bit number of the least-significant bit in the word. This
%   is usually 0 or 1, depending on channel label conventions.
% "shiftbitsname" is the LUT field to look for for the bit shift. If this
%   isn't found, a bit shift of 0 is assumed.
% "headerlabels" is the "label" cell array from the Field Trip header.
% "allevents" is the event list to search.
%
% "evtable" is a table containing the filtered event list. This may be empty.
% "have_events" is true if at least one matching event was detected.
```

7.9 euFT_getDerivedSignals.m

```
% function [ datalfp dataspike datarect ] = euFT_getDerivedSignals( ...
%   datawide, lfp_corner, lfp_rate, spike_corner, ...
%   rect_band, rect_lowpass, rect_rate, want_quiet )
%
% This performs filtering to turn a wideband signal into an LFP signal
% (low-pass filtered and downsampled), a spike signal (high-pass filtered),
% and a rectified activity signal (band-pass filtered, rectified, low-pass
% filtered, and then downsampled).
%
% This is a wrapper for ft_preprocessing() and ft_resampledata().
%
% "datawide" is the wideband Field Trip data structure.
% "lfp_corner" is the low-pass corner frequency for the LFP signal.
```

```
% "lfp_rate" is the desired sampling rate of the LFP signal, or NaN to use
%   the wideband sampling rate.
% "spike_corner" is the high-pass corner frequency for the spike signal.
% "rect_band" [low high] are the band-pass corners for the rectified signal.
% "rect_lowpass" is the low-pass smoothing corner for the rectified signal.
% "rect_rate" is the desired sampling rate of the rectified signal, or NaN
%   to use the wideband sampling rate.
% "want_quiet" is an optional argument. If set to true, progress output is
%   suppressed. If omitted, it defaults to true.
%
% "datalfp" is the LFP signal Field Trip data structure.
% "dataspike" is the spike signal Field Trip data structure.
% "datarect" is the rectified activity signal Field Trip data structure.
```

7.10 euFT_getFiltPowerBrick.m

```
% function cfg = euFT_getFiltPowerBrick( powerfreq, modecount )
%
% This generates a Field Trip ft_preprocessing() configuration structure for
% power-line filtering in the frequency domain using a "brick wall" filter
% (squashing unwanted frequency components). This causes ringing near large
% disturbances (a top-hat in the frequency domain gives a sinc function
% impulse response).
%
% FIXME - We're using the undocumented "brickwall" bsfilttype option to get
% this filter.
%
% FIXME - This doesn't seem to be working properly; it boosts amplitude
% and only slightly attenuates unwanted frequencies.
%
% "powerfreq" is the power-line frequency (typically 50 Hz or 60 Hz).
% "modecount" is the number of modes to include (1 = fundamental,
%   2 = fundamental plus first harmonic, etc).
%
% "cfg" is a Field Trip configuration structure for this filter.
```

7.11 euFT_getFiltPowerCosineFit.m

```
% function cfg = euFT_getFiltPowerCosineFit( powerfreq, modecount )
%
% This generates a Field Trip ft_preprocessing() configuration structure for
% power-line filtering in the frequency domain (DFT filter), using a cosine
% fit (subtracting the specified components).
%
% NOTE - This will work best for short trials. For longer trials, we may be
% able to pick up the fact that we're not exactly at the nominal frequency.
```

```
%
% "powerfreq" is the power-line frequency (typically 50 Hz or 60 Hz).
% "modecount" is the number of modes to include (1 = fundamental,
%   2 = fundamental plus first harmonic, etc).
%
% "cfg" is a Field Trip configuration structure for this filter.
```

7.12 euFT_getFiltPowerDFT.m

```
% function cfg = euFT_getFiltPowerDFT( powerfreq, modecount )
%
% This generates a Field Trip ft_preprocessing() configuration structure for
% power-line filtering in the frequency domain (DFT filter), using the
% "dftbandwidth" option to get a band-stop filter with known bandwidth.
%
% NOTE - This is for short signals only (segmented trials)! For anything
% longer than a few seconds, the type of filter this uses consumes a very
% large amount of memory.
%
% "powerfreq" is the power-line frequency (typically 50 Hz or 60 Hz).
% "modecount" is the number of modes to include (1 = fundamental,
%   2 = fundamental plus first harmonic, etc).
%
% "cfg" is a Field Trip configuration structure for this filter.
```

7.13 euFT_getFiltPowerFIR.m

```
% function cfg = euFT_getFiltPowerFIR( powerfreq, modecount, samprate )
%
% This generates a Field Trip ft_preprocessing() configuration structure for
% power-line filtering, using the "bsfreq" option to get a time-domain
% multi-notch band-stop filter.
%
% NOTE - This is intended for long continuous data, where frequency-domain
% filtering might introduce numerical noise.
%
% NOTE - We're using the FIR implementation of this; the IIR implementation
% is unstable and FT flags it as such.
%
% FIXME - The FIR filter takes forever due to needing a very wide filter and
% Matlab doing convolution in the time domain. We also need to use the
% undocumented "bsfiltord" configuration option.
%
% "powerfreq" is the power-line frequency (typically 50 Hz or 60 Hz).
% "modecount" is the number of modes to include (1 = fundamental,
%   2 = fundamental plus first harmonic, etc).
```

```
% "samprate" is the sampling rate of the signal being filtered (needed to
%   calculate the number of points needed for the FIR).
%
% "cfg" is a Field Trip configuration structure for this filter.
```

7.14 euFT_getFiltPowerTW.m

```
% function cfg = euFT_getFiltPowerTW( powerfreq, modecount )
%
% This generates a Field Trip ft_preprocessing() configuration structure for
% power-line filtering, using Thilo's old configuration. For each mode,
% Thilo specified a comb of frequencies to get something close to a
% band-stop filter. Implementation uses the "DFT" filter, which does a
% cosine fit at each requested frequency in the frequency domain.
%
% NOTE - This will only approximate a band-stop filter for short trials with
% relatively low sampling rates, I think. Frequency uncertainty should be
% comparable to the step size (0.1 Hz).
%
% "powerfreq" is the power-line frequency (typically 50 Hz or 60 Hz).
% "modecount" is the number of modes to include (1 = fundamental,
%   2 = fundamental plus first harmonic, etc).
%
% "cfg" is a Field Trip configuration structure for this filter.
```

7.15 euFT_getSingleBitEvent.m

```
% function [ evtable have_events ] = ...
%   euFT_getSingleBitEvent( namelut, thissigname, headerlabels, allevents )
%
% This looks up a channel label pattern for a given signal, looks up channels
% that match that pattern, picks the first such channel, and then finds
% events that are from that channel, returning the result as a table.
%
% NOTE - This only works for LoopUtil events! Those have the channel labels
% stored in the event records' "type" field.
%
% "namelut" is a structure indexed by signal name that has cell arrays of
%   Field Trip channel label specifiers (per ft_channelselection()).
% "thissigname" is the signal label to look for in "namelut".
% "headerlabels" is the "label" cell array from the Field Trip header.
% "allevents" is the event list to search.
%
% "evtable" is a table containing the filtered event list's fields. This
%   table may be empty.
% "have_events" is true if at least one matching event was detected.
```

7.16 euFT_getTrainTrialDefs.m

```
% function trialdefs = euFT_getTrainTrialDefs( ...
%   samprate, sampcount, evtimes, trig_window_ms, train_gap_ms )
%
% This builds a list of Field Trip trial definitions for events with the
% specified times. Events are grouped into trains of closely-spaced events,
% and relative position of each event within each train is recorded.
%
% The first sample in the continuous waveform is assumed to be time 0.
%
% "samprate" is the sampling rate.
% "sampcount" is the total number of samples in the continuous data.
% "evtimes" is a vector containing trigger event timestamps in seconds.
% "trig_window_ms" [ start stop ] is the window around trigger events
%   to save, in milliseconds. E.g. [ -100 300 ].
% "train_gap_ms" is a duration in milliseconds. Trigger events with this
%   separation or less are considered to be part of an event train.
%
% "trialdefs" is a "trl" matrix per ft_definetrial(). It is a Nx7 matrix.
%   The first three columns are the starting sample index, the ending sample
%   index, and the relative position of the first sample in the trial with
%   respect to the trigger (0 = on trigger, + = after trigger, - = before
%   trigger). Column 4 is the relative position of each trial's event within
%   its associated train. Columns 5, 6, and 7 are the trigger time, trial
%   starting time, and trial ending time in seconds.
```

7.17 euFT_iterateAcrossFolderBatchingDerived.m

```
% function auxdata = euFT_iterateAcrossFolderBatchingDerived( ...
%   config_load, iterfunc_derived, chanbatchsize, trialbatchsize, ...
%   artparams, notch_freqs, notch_bw, lfp_corner, lfp_rate, ...
%   spike_corner, mua_band, mua_corner, mua_rate, ...
%   verbosity )
%
% This iterates across a Field Trip dataset, loading a few channels at a time
% and performing built-in and user-specified processing. Processing output
% is aggregated and returned.
%
% The idea is to be able to process a dataset much larger than can fit in
% memory. At 30 ksps the footprint is typically about 1 GB per channel-hour.
%
% Built-in processing is artifact rejection followed by notch filtering
% followed by calling euFT_getDerivedSignals() to get LFP, spike, and MUA
% waveforms. These are passed (with the wideband waveform) to a user-specified
% function for futher processing.
%
```

```
% The user-specified iteration function handle is of the type described in
% FT_ITERFUNC_DERIVED.txt. Return values are aggregated in "auxdata".
%
% "config_load" is a Field Trip configuration structure to be passed to
%   ft_preprocessing() to load the data. The "channel" field is split into
%   batches when iterating.
% "iterfunc_derived" is a function handle used to transform derived waveform
%   data into "result" data, per FT_ITERFUNC_DERIVED.txt.
% "chanbatchsize" is the number of channels to process at a time. Set this to
%   inf to process all channels at once.
% "trialbatchsize" is the number of trials to process at a time. Set this to
%   inf to process all trials at once.
% "artparams" is a structure containing configuration parameters for
%   nlChan_applyArtifactReject(). Pass "struct([])" to disable artifact
%   rejection.
%   FIXME - This needs to be overhauled to support other artifact options.
% "notch_freqs" is a vector containing frequencies to notch-filter. This can
%   be empty to disable notch filtering.
% "notch_bw" is the bandwidth of the notch filter to use.
% "lfp_corner" is the low-pass corner frequency to use when generating the LFP.
% "lfp_rate" is the rate to downsample the LFP waveform to.
% "spike_corner" is the high-pass corner frequency to use when generating the
%   "spikes only" signal.
% "mua_band" [low high] is the band-pass frequency range to use when
%   generating the multi-unit activity signal prior to rectification.
% "mua_corner" is the low-pass corner frequency to use when generating the
%   multi-unit activity signal after rectification.
% "mua_rate" is the rate to downsample the rectified MUA signal to.
% "verbosity" is an optional argument. It can be set to 'none' (no console
%   output), 'terse' (reporting the number of channels processed), or 'full'
%   (reporting the names of channels processed). The default is 'none'.
%
% "auxdata" is a cell array indexed by {trial,channel} containing the
%   output returned by iterfunc_derived.
```

7.18 euFT_resampleTrialDefs.m

```
% function newtrialdefs = ...
%   euFT_resampleTrialDefs( oldtrialdefs, oldrate, newrate );
%
% This function alters a trial definition table to account for a changed
% sampling rate. Time zero is expected to be consistent between the
% two representations (i.e. just a scaling, without shifting).
%
% "oldtrialdefs" is a matrix or table containing trial definitions, per
%   ft_definetrial().
% "oldrate" is the sampling rate used with "oldtrialdefs".
% "newrate" is the desired new sampling rate.
```

```
%
% "newtrialdefs" is a copy of "oldtrialdefs" with the start, end, and
%   offset columns modified to reflect the new sampling rate.
```

Chapter 8

"euHLev" Notes

8.1 ARTIFACTCONFIG.txt

```
Artifact removal is specified using a method label and a configuration
structure.



Method labels are character vectors with the following values:

'' or 'none' specifies no artifact suppression.

'sigma' specifies standard deviation based artifact rejection, per
  nlProc_removeArtifactsSigma.

'expknown' specifies exponential curve fits across known time ranges,
  per nlArt_removeMultipleExpDecays.

'expguess' specifies multiple exponential curve fits guessing at locations,
  per nlArt_removeExpDecaysByGuessing.



Standard deviation based artifact rejection configuration is a structure with
the following fields:

"threshold_adjust" is a scalar added to the default threshold levels. A
  positive value raises the absolute and derivative thresholds by the
  specified number of standard deviations, and a negative value lowers them.



Exponential fits with known locations are configured using a structure with
the following fields:
```

```
"fit_fenceposts_ms" is a vector containing times to be used as span endpoints
  for exponential curve fitting. The span between the last two times is curve
  fit and its contribution subtracted, then the next-last span, and so forth.
  If this is [], no curve fits are performed.

"fit_method" is a character vector or cell array specifying the algorithms
  to use for curve fitting all segments (if a cahracter vector) or for
  each segment individually (if a cell array of character vectors). If this
  is '' or absent, a default algorithm is used.



Exponential fits with guessed locations are configured using a structure
with the following fields:

(The fields described in EXPGUESSCONIG.txt from lib-nloop-artifact.)

The following additional fields:

"plot_config" is a debug plot/report configuration structure per
  EXPGUESSPLOT.txt, with the following additional fields:

  "want_debug_plots" is true to emit debugging plots of curve fits, and false
    to suppress plots.

  "want_reports" is true to save summarized reports of curve fits, and false
    to suppress plots.

  "tattle_verbosity" is 'quiet', 'terse', 'normal', or 'verbose'. This
    controls how many debugging/progress messages are sent to the console.

  "report_verbosity" is 'quiet', 'terse', 'normal', or 'verbose'. This
    controls how many debugging/progress messages are written to log files.



(This is the end of the file.)
```

8.2 RAWFOLDERMETA.txt

```
Folder metadata as returned by euHLev_getAllMetadata_XXX is encapsulated as
a structure with the fields described below.


Common to all folder types:

"folder" is the folder path (as returned by euUtil_getExperimentFolders()).
```

```
"type" is 'openephys', 'intanrec', or 'intanstim'.

"header_ft" is the Field Trip header structure for this folder.
"chans_an" is a cell array containing analog channel FT labels.
"chans_dig" is a cell array containing digital channel FT labels.

"chanmap_raw" is a cell array with FT channel names before mapping.
"chanmap_cooked" is a cell array with FT channel names after mapping.

...The channel map is a "best guess", and might not be defined if the
helper function couldn't find appropriate configuration files.


Fields for type 'openephys' (Open Ephys v0.5):

"settings" is a copy of the XML parse of "settings.xml", as returned by
  readstruct().
"oebufinfo" is Open Ephys buffer metadata as returned by
  nlOpenE_parseConfigAudioBufferInfo_v5().
"oesigchain" is a cell array containing metadata for all processor nodes
  found by nlOpenE_parseConfigProcessorsXML_v5().


(This is the end of the file.)
```

8.3 SQUASHCONFIG.txt

```
Artifact squashing, step correction, and interpolation are specified using
a configuration structure with the following fields:


"squash_window_ms" [ start stop ] is a window around t=0 to replace with
  NaN, in milliseconds (e.g. [ -0.5 1.5 ]). If this is [] or absent, no
  squashing is performed.

"auto_squash_threshold" is the amount by which the signal must depart
  from the DC level to be automatically squashed as a residual artifact,
  if adjacent to an existing NaN span. This is a multiple of
  the median-to-quartile distance (about two thirds of a standard
  deviation). Typical values are 6-12 for clear exponential excursions.
  If this is absent or Inf or NaN, no squashing is performed.

"ramp_span_ms" [ start stop ] is a duration around t=0 to apply a ramp to
  remove any DC step present over NaN regions (squashed regions). This
  If this is [] or absent, no ramping is performed.

"want_interp" is true to interpolate within squashed regions, false or
  absent to leave NaN regions in place.
```

(This is the end of the file.)

Chapter 9

"euHLev" Functions

9.1 euHLev_applyChannelBlacklist.m

```
% function keepmask = ...
%   euHLev_applyChannelBlacklist( chanlist, classfile, userconfig )
%
% This reads a file containing hand-annotated channel classes and applies
% a blacklist or whitelist to identify channels that should be kept. The
% supplied channel list is masked using the resulting rules.
%
% Only channels mentioned in the file are kept. If no such file exists,
% all channels are kept.
%
% "chanlist" is a cell array containing the list of channel names to filter.
% "classfile" is the name of a CSV file containing hand-annotated channel
%   types, per euUtil_readChannelClasses().
% "userconfig" is an optional argument containing a structure with key/value
%   pairs to override. Relevant fields (keys) are:
%   "chancolumn" is the name of the column containing channel names.
%   "typecolumn" is the name of the column to read annotated types from.
%   "whitelist" is a cell array containing acceptable type values.
%   "blacklist" is a cell array containing unacceptable type values.
%
% "keepmask" is a vector of boolean values that's true for entries in
%   "chanlist" that should be kept.
%
% If a non-empty whitelist is supplied, it's used. Otherwise the blacklist
% is used.
```

9.2 euHLev_checkChannelsExist.m

```
% function found = euHLev_checkChannelsExist( header_ft, headername, chanlist )
```

```
%
% This checks the "label" field of the specified FT header to see if it
% contains the requested channel names. Error messages are optionally
% displayed if channels are missing.
%
% "header_ft" is the Field Trip header to search.
% "headername" is a human-readable name for identifying the header when
%   reporting missing channels (e.g. 'FooHeader', for "FooHeader doesn't
%   contain..."). If this is [], no messages are produced.
% "chanlist" is a cell array containing the channel names to search for.
%   Channel names that are empty character arrays ('') are skipped.
%
% "found" is true if all of the requested channels are found and false
%   otherwise.
```

9.3 euHLev_doSquashAndFill.m

```
% function newftdata = euHLev_doSquashAndFill( oldftdata, squashconfig )
%
% This function squashes specified time ranges, squashes excursions beyond
% a certain threshold from the median, adds a ramp to compensate for stepwise
% discontinuities across NaN ranges, and interpolates NaN ranges.
%
% These functions can be enabled, disabled, and tuned via a config structure.
%
% "oldftdata" is a ft_datatype_raw structure to modify.
% "squashconfig" is a configuration structure, per SQUASHCONFIG.txt.
%
% "newftdata" is a copy of "oldftdata" with the desired modifications made.
```

9.4 euHLev_getAllMetadata_OpenEv5.m

```
% function foldermeta = euHLev_getAllMetadata_OpenEv5(thisfolder)
%
% This function processes a folder that contains a single structure.oebin
% file, and calls helper functions to return all relevant metadata
% (field trip information, channel list and channel map, open ephys signal
% chain metadata, etc).
%
% "thisfolder" is the name of the folder that contains structure.oebin, as
%   returned by euUtil_getExperimentFolders().
%
% "foldermeta" is a structure with contents described in RAWFOLDERMETA.txt.
```

9.5 euHLev_getArtifactSigmaDefaults.m

```
% function paramstruct = euHLev_getArtifactSigmaDefaults( samprate )
%
% This function returns a structure containing reasonable default tuning
% parameters for nlArt_removeArtifactsSigma and nlFT_removeArtifactsSigma.
%
% "samprate" is either a sampling rate (to get durations in samples, for
%   nlArt_removeArtifactsSigma) or NaN (to get durations in milliseconds
%   or seconds, for nlFT_removeArtifactsSigma).
%
% "paramstruct" is a structure with the following fields:
%   "ampdetect" is the threshold for flagging amplitude excursions.
%   "derivdetect" is the threshold for flagging derivative excursions.
%   "ampturnoff" is the turn-off threshold for amplitude excursion detection.
%   "derivturnoff" is the turn-off threshold for derivative excursion
%     detection.
%   "squashbefore" is the added number of samples or milliseconds ahead of
%     the excursion to squash.
%   "squashafter" is the added number of samples or milliseconds after the
%     excursion to squash.
%   "derivsmooth" is the size of the derivative smoothing window in samples
%     or milliseconds.
%   "dcsmooth" is the size of the dc-level smoothing window in samples or
%     seconds.
```

9.6 euHLev_getDefaultChannelBlacklistConfig.m

```
% function config = euHLev_getDefaultChannelBlacklistConfig()
%
% This returns a structure containing default configuration parameters for
% euHLev_applyChannelBlacklist().
%
% "config" is a structure of key/value pairs that includes the following:
%   "chancolumn" is the name of the column containing channel names.
%   "typecolumn" is the name of the column to read annotated types from.
%   "whitelist" is a cell array containing acceptable type values.
%   "blacklist" is a cell array containing unacceptable type values.
```

9.7 euHLev_getOpenEHeaderChannels.m

```
% function [ header chans_ephys chans_digital chanmap_raw chanmap_cooked ] = ...
%   euHLev_getOpenEHeaderChannels( configfolder, ephysfolder )
%
% This fetches the FT header and relevant channel names from the specified
```

```
% ephys folder, and also searches the config folder and ephys folder to
% fetch the channel map that was used.
%
% This is a wrapper for various Field Trip and euUtil_ functions.
%
% NOTE - "ephysfolder" can be a cell array ("configfolder" shouldn't be).
% If "ephysfolder" is a cell array, each entry is treated as a folder to be
% checked, and all return values are also per-folder cell arrays.
%
% "configfolder" is the folder to search for the channel map. Empty to skip.
% "ephysfolder" is the folder to search for ephys metadata.
%
% "header" is the Field Trip ephys header.
% "chans_ephys" is a list of channels containing ephys data.
% "chans_digital" is a list of channels containing digital event data.
% "chanmap_raw" is a list of raw ephys channel names to be mapped.
% "chanmap_cooked" is a list of corresponding cooked ephys channel names.
```

9.8 euHLev_readAndCleanSignals.m

```
% function ftdata_ephys = euHLev_readAndCleanSignals( folder, ephys_chans, ...
%   trial_config, artifact_method, artifact_config, squash_config, ...
%   notch_freqs, notch_bw )
%
% This reads Field Trip trial data from the specified folder, performing
% artifact rejection, NaN squashing/paving, and notch filtering in that order.
%
% This can be used to read continuous data or event-segmented data, depending
% on what's passed as "trial_config".
%
% NOTE - Channel names specified as '' are skipped (removed from the list
% before calling FT's read functions).
%
% NOTE - For large datasets, call euFT_iterateAcrossFolderBatchingDerived
% instead. This is intended for quick and dirty processing of datasets that
% fit in memory.
%
% NOTE - This does _not_ detrend or demean data, so it's safe for reading
% magnitudes, phase angles, event codes, and so forth.
%
% NOTE - Events and waveforms have timestamps relative to the start of the
% recording (or to the trial's t=0 point), not the start of the trim window.
%
% "folder" is the folder to read from.
% "ephys_chans" is a cell array containing channel names to read. If this is
%   empty, no ephys data is read (an empty struct array is returned).
% "trial_config" is a scalar (for monolithic data), a structure (for
%   automatic segmentation around events), or a matrix (for manually
```

```
%   supplied trial definitions).
%   As a scalar (in the range 0 to 0.5) it's the amount of time to trim from
%   the start and end of the ephys data as a fraction of the untrimmed length.
%   As a matrix, it's a Field Trip trial definition structure passed as "trl"
%   when reading the data.
%   As a structure, it has the following fields:
%     "trigtimes" is a series of trigger event times, in seconds.
%     "trig_window_ms" [ start stop ] is a duration span in milliseconds
%       to capture for each event, with the trigger at time zero.
%     "train_gap_ms" is a duration in milliseconds. Events that are separated
%       by no more than this time are considered part of an event train, per
%       euFT_getTrainTrialDefs().
% "artifact_method" is an artifact rejection method, per ARTIFACTCONFIG.txt.
%  This can be '' or 'none' to disable artifact rejection.
% "artifact_config" is a structure containing configuration information for
%   the chosen artifact rejection method, per ARTIFACTCONFIG.txt. This may
%   be struct() or struct([]) if no method is used.
% "squash_config" is a structure describing artifact squashing, step
%   correction, and interpolation to be performed, per SQUASHCONFIG.txt.
%   This may be struct() or struct([]) if none of these are to be done.
% "notch_freqs" is a vector containing notch filter frequencies. An empty
%   vector disables notch filtering.
% "notch_bw" is the notch filter bandwidth in Hz. NaN disables filtering.
%
% "ftdata_ephys" is a Field Trip data structure containing ephys data for
%   the specified ephys channels. NOTE - This will be an empty struct array
%   if there were no ephys channels specified or no matching channels found.
```

9.9 euHLev_readEvents.m

```
% function [ event_ftdata event_waves wave_times event_edges ] = ...
%   euHLev_readEvents( folder, event_chans, trim_fraction )
%
% This reads Field Trip event data from the specified folder, trimming to
% the specified window, and returning events in several formats.
%
% NOTE - Channel names specified as '' are skipped (removed from the list
% before calling FT's read functions).
%
% NOTE - This assumes all of the requested data can fit in memory. Since
% this includes waveform reconstructions, that might be a poor assumpion.
%
% NOTE - Events and waveforms have timestamps relative to the start of the
% recording, not the start of the trim window.
%
% "folder" is the folder to read from.
% "event_chans" is a cell array containing event channel names to read. If
%   this is empty, no events are read (an empty cell array is returned).
```

```
% "trim_fraction" (in the range 0 to 0.5) is the amount of time to trim from
%   the start and end of the ephys data as a fraction of the untrimmed length.
%
% "event_ftdata" is a cell array with the same number of elements as
%   "event_chans". Each cell contains a vector of Field Trip event records
%   (per "nlFT_readEvents") from the associated event channel.
% "event_waves" is a cell array with the same number of elements as
%   "event_chans". Each cell contains a logical vector holding sampled values
%   of the event signal interpreted as a TTL waveform.
% "wave_times" is a vector holding sample timestamps. This is the same for
%   all waves contained in "event_waves".
% "event_edges" is a cell array with the same number of elements as
%   "event_chans". Each cell contains a structure with fields "risetimes",
%   "falltimes", and "bothtimes", containing vectors holding event timestamps
%   for rising edges, falling edges, and all edges, respectively.
```

9.10 euHLev_selectLouieChannels.m

```
% function newchans_raw = euHLev_selectLouieChannels( ...
%   logfile, logdate, oldchans_raw, chanmap_raw, chanmap_cooked )
%
% This selects the subset of channels that Louie flagged as "good" in a
% dataset.
%
% If a non-empty channel map is supplied Louie's channel numbers are assumed
% to be cooked channel numbers and the old/new channel lists are assumed to
% be raw channel numbers.
%
% If the channel map is empty then the old/new channel lists and the log file
% are assumed to use the same numbering scheme.
%
% "logfile" is the log file to read.
% "logdate" is the datestamp to look for in the log file.
% "oldchans_raw" is the channel list before pruning.
% "chanmap_raw" is a list of raw channel names to be translated.
% "chanmap_cooked" is a list of corresponding cooked channel names.
%
% "newchans_raw" is a copy of "oldchans_raw" that only contains channels
%   that Louie identified as "good".
```

Chapter 10

"euPlot" Functions

10.1 euPlot_axesPlotFTTimelock.m

```
% function euPlot_axesPlotFTTimelock( thisax, ...
%   timelockdata_ft, chans_wanted, spread_fraction, bandsigma, ...
%   plot_timerange, plot_yrange, legendpos, figtitle )
%
% This plots the mean and variance of one or more channel waveforms in the
% current axes. Events are rendered as cursors behind the waveforms.
%
% "thisax" is the "axes" object to render to.
% "timelockdata_ft" is a Field Trip structure produced by
%   ft_timelockanalysis().
% "chans_wanted" is a cell array with channel names to plot. Pass an empty
%   cell array to plot all channels.
% "spread_fraction" specifies the Y offset spacing between channels in a
%   plot. 1.0 offsets by the full Y range; 0 overlaps channels without offset.
% "bandsigma" is a scalar indicating where to draw confidence intervals.
%   This is a multiplier for the standard deviation and for SEM.
%   Use NaN to disable these intervals.
% "plot_timerange" [ min max ] is the time range (X range) of the plot axes.
%   Pass an empty range for auto-ranging.
% "plot_yrange" [ min max ] is the Y range of the plot axes.
%   Pass an empty range for auto-ranging.
% "legendpos" is a position specifier to pass to the "legend" command, or
%   'off' to disable the plot legend. The legend lists channel labels.
% "figtitle" is the title to apply to the figure. Pass an empty character
%   array to disable the title.
```

10.2 euPlot_axesPlotFTTrials.m

```
% function euPlot_axesPlotFTTrials( thisax, ...
```

```
%   wavedata_ft, wavedata_samprate, ...
%   trialdefs, trialnames, trialdef_samprate, ...
%   chans_wanted, plot_timerange, plot_yrange, ...
%   events_codes, events_rwdA, events_rwdB, event_samprate, ...
%   legendpos, figtitle )
%
% This plots a series of stacked trial waveforms in the current axes.
% Events are rendered as cursors behind the waveforms.
%
% "thisax" is the "axes" object to render to.
% "wavedata_ft" is a Field Trip "datatype_raw" structure with the trial
%   data and metadata.
% "wavedata_samprate" is the sampling rate of "wavedata_ft".
% "trialdefs" is the Field Trip trial definition matrix or table that was
%   used to generate the trial data.
% "trialnames" is either a vector of trial numbers or a cell array of trial
%   labels, corresponding to the trials in "trialdefs". An empty vector or
%   cell array auto-generates labels.
% "trialdefs_samprate" is the sampling rate used when generating "trialdefs".
% "chans_wanted" is a cell array with channel names to plot. Pass an empty
%   cell array to plot all channels.
% "trials_wanted" is a cell array with labels of trials to plot. Pass an
%   empty cell array to plot all trials.
% "plot_timerange" [ min max ] is the time range (X range) of the plot axes.
%   Pass an empty range for auto-ranging.
% "plot_yrange" [ min max ] is the Y range of the plot axes.
%   Pass an empty range for auto-ranging.
% "events_codes" is a Field Trip event structure array or table with event
%   code events. This may be empty.
% "events_rwdA" is a Field Trip event structure array or table with reward
%   line A events. This may be empty.
% "events_rwdB" is a Field Trip event structure array or table with reward
%   line B events. This may be empty.
% "events_samprate" is the sampling rate used when reading events.
% "legendpos" is a position specifier to pass to the "legend" command, or
%   'off' to disable the plot legend. The legend lists channel labels.
% "figtitle" is the title to apply to the figure. Pass an empty character
%   array to disable the title.
```

10.3 euPlot_decimatePlotsBresenham.m

```
% function keepflags = euPlot_decimatePlotsBresenham( keepcount, origarray )
%
% This selects a subset of elements of a supplied vector to keep, and
% returns a logical vector indicating which to keep and which to discard.
%
% The idea is to be able to say "newarray = origarray(keepflags)" afterwards,
% or filter other arrays with the same geometry.
```

```
%
% "keepcount" is the desired number of elements to keep.
% "origarray" is a vector or cell array of arbitrary type (usually a label
%   or channel list) with the same dimensions as the element array to filter.
%   This should be one-dimensional (matrix results are undefined).
%
% "keepflags" is a boolean vector with the same dimensions as "origarray"
%   that's true for elements to be kept and false for elements to discard.
```

10.4 euPlot_getBoxChartGroups.m

```
% function [ boxgroupindices boxgroupvalues ] = ...
%   euPlot_getBoxChartGroups( data_xseries, data_xbins )
%
% This produces an "xgroupdata" series for use with the "boxchart"
% function. Plotted data is binned by the "x series" variable (we only need
% to know this variable, not the plotted data values themselves).
%
% "data_xseries" is a vector containing values used for binning plotted data.
% "data_xbins" is a vector containing bin edges for binning x-series values.
%
% "boxgroupindices" is a vector containing bin indices for each supplied
%   data point.
% "boxgroupvalues" is a vector containing bin midpoint X values for each
%   supplied data point (which "boxchart" uses to generate column names).
```

10.5 euPlot_getDataSeriesRange.m

```
% function [ minval maxval ] = euPlot_getDataSeriesRange( ...
%   dataseries, defaultrange, extrapoints )
%
% This accepts a vector of data values and determines its plotting extents.
%
% This will tolerate non-numeric data (such as cell arrays), falling back to
% defaults.
%
% "dataseries" is a vector containing data to be ranged.
% "defaultrange" [ min max ] is the plotting range to fall back to.
% "extrapoints" is a vector containing additional data series values. The
%   plotting range is extended to include these values.
%
% "minval" is the minimum value plotted.
% "maxval" is the maximum value plotted.
```

10.6 euPlot_getStructureSeriesRange.m

```
% function [ minval maxval ] = euPlot_getStructureSeriesRange( ...
%   structlist, fieldwanted, defaultrange, extrapoints )
%
% This accepts a structure array or a cell array of structures, aggregates
% one field as a data series, and determines the plotting extents of that
% data series.
%
% This will tolerate non-numeric data (such as cell arrays), falling back to
% defaults.
%
% "structlist" is a structure array or a cell array containing structures.
% "fieldwanted" is the name of the field to extract.
% "defaultrange" [ min max ] is the plotting range to fall back to.
% "extrapoints" is a vector containing additional data series values. The
%   plotting range is extended to include these values.
%
% "minval" is the minimum value plotted.
% "maxval" is the maximum value plotted.
```

10.7 euPlot_helperMakeTrialNames.m

```
% function newnames = euPlot_helperMakeTrialNames(oldnames, trialcount)
%
% This function converts its argument to a cell array with per-trial
% character array labels. Input can be a cell array of character labels,
% a vector of numbers, or an empty cell array or empty vector.
%
% "oldnames" is the list to convert.
% "trialcount" is the number of trials to generate labels for if passed an
%   empty list. This is ignored if passed a non-empty list (no checking).
%
% "newnames" is an Nx1 cell array with per-trial character array labels.
```

10.8 euPlot_plotAuxData.m

```
% function euPlot_plotAuxData( auxdata_ft, auxsamprate, ...
%   trialdefs, trialnames, trialsamprate, evlists, evsamprate, ...
%   chans_wanted, plots_wanted, figtitle, obase )
%
% This plots several channels of auxiliary data in strip chart form and
% saves the resulting plots. Plots may have all trials stacked or be plotted
% per trial or both.
%
```

```
% NOTE - Time ranges and decorations are hardcoded.
%
% This is a wrapper for euPlot_axesPlotFTTrials().
%
% "auxdata_ft" is a Field Trip "datatype_raw" structure with the trial data
%   and metadata.
% "auxsamprate" is the sampling rate of "auxdata_ft".
% "trialdefs" is the field trip trial definition matrix or table that was
%   used to generate the trial data.
% "trialnames" is either a vector of trial numbers or a cell array of trial
%   labels, corresponding to the trials in "trialdefs". An empty vector or
%   cell array auto-generates labels.
% "trialsamprate" is the sampling rate used when generating "trialdefs".
% "evlists" is a structure containing event lists or tables, with one event
%   list or table per field. Fields tested are 'cookedcodes', 'rwdA', and
%   'rwdB'.
% "evsamprate" is the sampling rate used when reading events.
% "chans_wanted" is a cell array with channel names to plot. This cannot be
%   empty. Each listed channel gets one strip (subplot) in the strip chart.
% "plots_wanted" is a cell array containing zero or more of 'oneplot' and
%   'pertrial', controlling which plots are produced.
% "figtitle" is the prefix used when generating figure titles.
% "obase" is the prefix used when generating output filenames.
```

10.9 euPlot_plotBasisCharts.m

```
% function euPlot_plotBasisCharts( basislist, fomlist, fompattern, ...
%   timeseries, chanlist, window_sizes_ms, size_labels, titleprefix, obase )
%
% This makes a spreadsheet and several plots describing the decomposition of a
% series of channel signals into a linear combination of basis signals.
%
% This is intended to be used with the output of euHLev_getBasisDecomposition.
%
% A "foms.csv" file contains figures of merit as a function of the number of
% basis vectors.
%
% A "coeffs.png" plot shows the contributions of each basis vector to each
% channel's signal.
%
% A "basis.png" plot shows the basis vector signals.
%
% A "background.png" plot shows the common background signal.
%
% "basislist" is a cell array containing structures that have basis vector
%   information in the following fields, per BASISVECTORS.txt:
%   "basisvecs" is a Nbasis x Ntimesamples matrix where each row is a basis
%     vector signal.
```

```
%   "coeffs" is a Nchannls x Nbasis matrix with basis vector weights for each
%     input vector. (coeffs * basisvecs) is an estimate of the original
%     Nchannels x Ntimesamples data matrix.
%   "background" is a 1 x Ntimesamples vector containing the common background
%     across channels.
% "fomlist" is a vector containing a figure of merit for each of the cases
%   in "basislist".
% "fompattern" is a sprintf pattern for formatting figures of merit.
% "timeseries" is a vector containing time axis points for plotting basis
%   vectors.
% "chanlist" is a cell array containing channel names for Nchannels signals.
% "window_sizes_ms" is a cell array. Each cell contains a plot time range
%   [ begin end ] in milliseconds, or [] for the full data extent.
% "size_labels" is a cell array containing filename-safe labels used when
%   creating filenames and annotating titles for plots of each window size.
% "titleprefix" is the prefix used when generating figure titles.
% "obase" is the prefix used when generating output filenames.
```

10.10 euPlot_plotEventAlignedWaves.m

```
% function euPlot_plotEventAlignedWaves( ...
%   evtimes, time_before_trig_ms, time_after_trig_ms, time_after_last_ms, ...
%   max_train_gap_ms, wave_sets, legendpos, max_plot_count, fnamebase )
%
% This renders multi-panels plot of several waveforms during time windows
% around a triggering event. Only the first event of a train of events is
% used as a trigger.
%
% If both "time_after_trig" and "time_after_last" are given, the longer of
% the two durations is used. If one is NaN, the other is used alone.
%
% Empty legend labels suppress rendering of that legend label.
%
% "evtimes" is a vector containing event timestamps (in seconds).
% "time_before_trig_ms" is the number of milliseconds to extend the plot
%   before the trigger event.
% "time_after_trig_ms" is the number of milliseconds to extend the plot
%   after the trigger event (first event in a train).
% "time_after_last_ms" is the number of milliseconds to extend the plot
%   after the last event in a train.
% "max_train_gap_ms" is the maximum number of milliseconds between successive
%   events for the events to still be considered part of one event train
%   (only the first event in a train is used as a plot trigger).
% "wave_sets" is a cell array containing subplot definitions. Each subplot
%   definition is itself a cell array with the following content:
%   { subplot_title, wave_list, vert_cursor_list, horiz_cursor_list }
%   "subplot_title" is a prefix to use when building human-readable titles.
%   "wave_list" is a cell array describing a wave to plot:
```

```
%     { time_series, wave_series, colour, legend_label }
%   "vert_cursor_list" is a cell array describing vertical cursor lines to
%     plot: { cursor_times, colour, legend_label }
%   "horiz_cursor_list" is a cell array describing horizontal cursor lines
%     to plot: { cursor_yvals, colour, legend_label }
% "legendpos" is the legend location ('off' to disable).
% "max_plot_count" is a scalar indicating the maximum number of plots to
%   emit. If this is smaller than the number of plot trigger events, the
%   list of plot trigger events is decimated to reduce the number of plots.
% "fnamebase" is a prefix to use when constructing filenames.
%
% No return value.
```

10.11 euPlot_plotFTTimelock.m

```
% function euPlot_plotFTTimelock( timelockdata_ft, bandsigma, ...
%   plots_wanted, window_sizes_ms, size_labels, max_count_per_size, ...
%   figtitle, obase )
%
% This plots a series of time-locked average waveforms and saves the
% resulting plots. Plots may have all channels stacked, or be per-channel,
% or a combination of the above.
%
% NOTE - Decorations are hardcoded.
%
% This is a wrapper for euPlot_axesPlotFTTimelock().
%
% "timelockdata_ft" is a Field Trip structure produced by
%   ft_timelockanalysis().
% "bandsigma" is a scalar indicating where to draw confidence intervals.
%   This is a multiplier for the standard deviation and for SEM.
% "plots_wanted" is a cell array containing zero or more of 'oneplot' and
%   'perchannel', controlling which plots are produced.
% "window_sizes_ms" is a cell array. Each cell contains a plot time range
%   [ begin end ] in milliseconds, or [] for the full data extent.
% "size_labels" is a cell array containing filename-safe labels used when
%   creating filenames and annotating titles for plots of each window size.
% "max_count_per_size" is a scalar indicating the maximum number of plots
%   to emit at a given size level. Set to inf to not decimate plots.
% "figtitle" is the prefix used when generating figure titles.
% "obase" is the prefix used when generating output filenames.
```

10.12 euPlot_plotFTTrials.m

```
% function euPlot_plotFTTrials( wavedata_ft, wavesamprate, ...
%   trialdefs, trialnames, trialsamprate, evlists, evsamprate, ...
```

```
%   plots_wanted, window_sizes, size_labels, max_count_per_size, ...
%   figtitle, obase )
%
% This plots a series of stacked trial waveforms and saves the resulting
% plots. Plots may have all trials and channels stacked, or have all
% trials stacked and have one plot per channel, or have all channels
% stacked and have one plot per trial, or a combination of the above.
%
% NOTE - Decorations are hardcoded.
%
% This is a wrapper for euPlot_axesPlotFTTrials().
%
% "wavedata_ft" is a Field Trip "datatype_raw" structure with the trial data
%   and metadata.
% "wavesamprate" is the sampling rate of "wavedata_ft".
% "trialdefs" is the field trip trial definition matrix or table that was
%   used to generate the trial data.
% "trialnames" is either a vector of trial numbers or a cell array of trial
%   labels, corresponding to the trials in "trialdefs". An empty vector or
%   cell array auto-generates labels.
% "trialsamprate" is the sampling rate used when generating "trialdefs".
% "evlists" is a structure containing event lists or tables, with one event
%   list or table per field. Fields tested for are 'cookedcodes', 'rwdA',
%   and 'rwdB'.
% "evsamprate" is the sampling rate used when reading events.
% "plots_wanted" is a cell array containing zero or more of 'oneplot',
%   'perchannel', and 'pertrial', controlling which plots are produced.
% "window_sizes" is a cell array. Each cell contains a plot time range
%   [ begin end ] in seconds, or [] for the full data extent.
% "size_labels" is a cell array containing filename-safe labels used when
%   creating filenames and annotating titles for plots of each window size.
% "max_count_per_size" is a scalar indicating the maximum number of plots
%   to emit at a given size level. Set to inf to not decimate plots.
% "figtitle" is the prefix used when generating figure titles.
% "obase" is the prefix used when generating output filenames.
```

10.13 euPlot_plotMultipleBoxCharts.m

```
% function euPlot_plotMultipleBoxCharts( ...
%   datavalues, databinvalues, datasetlabels, ...
%   xtype, ytype, xrange, yrange, xstr, ystr, wantoutliers, boxwidth, ...
%   plotlines, plothcursors, legendpos, titlestr, fname )
%
% This creates a box plot showing several data series and data sets.
%
% Data is provided as one vector, which may be several concatenated datasets.
% Each sample has a "bin value" (a scalar number) or "group label" (a
% character vector), and a "dataset label" (a character vector).
```

```
%
% Each "bin value"/"group label" is plotted at its own X position, with a
% cluster of boxes with different "dataset labels". Each "dataset label"
% gets its own box colour (so, boxes within a cluster have different colours).
%
% If there are no "bin values"/"group labels", datasets are spread out
% on the X axis instead of grouped.
%
% "datavalues" is a vector with the Y axis data values to plot.
% "databinvalues" is a vector with per-sample X axis data values for each
%   sample in "datavalues". This may alternatively be a cell array with
%   per-sample group labels. Use [] to not sort by bin/group.
% "datasetlabels" is a cell array with per-sample dataset labels for each
%   sample in "datavalues"; different datasets get different colours. Use {}
%   to not sort by dataset.
% "xtype" is 'linear' or 'log'.
% "ytype" is 'linear' or 'log'.
% "xrange" is the X axis range, or [] to auto-range.
% "yrange" is the Y axis range, or [] to auto-range.
% "xstr" is a character vector with the X axis label.
% "ystr" is a character vector with the Y axis label.
% "wantoutliers" is true to plot outlier dots and false otherwise.
% "boxwidth" is the box width ("boxchart" uses 0.5 as the default).
% "plotlines" is a cell array containing line definitions to plot. Each
%   line definition is a cell array with the form:
%   { x1 y1 x2 y2 colour label } ...If "label" is '', no legend line is shown.
% "plothcursors" is a cell array containing horizontal cursor definitions to
%   plot. Each horizontal cursor definition is a cell array with the form:
%   { yval colour label } ...If "label" is '', no legend line is shown.
% "legendpos" is the legend location showing set labels ('off' to disable).
% "titlestr" is a character vector with the plot title.
% "fname" is the filename to save the plot to.
```

10.14 euPlot_plotOverlappingHistograms.m

```
% function euPlot_plotOverlappingHistograms( ...
%   hist_list, xtype, ytype, legendpos, xstr, titlestr, fname )
%
% This generates a line plot representing multiple overlaid histograms.
%
% Empty legend labels suppress rendering of that legend label.
%
% "hist_list" is a cell array. Each cell contains a cell array describing
%   the histograms to plot: { counts, bin_edges, colour, legend_label }
% "xtype" is 'linear' or 'log'.
% "ytype" is 'linear' or 'log'.
% "xstr" is a character array containing the X axis label.
% "legendpos" is the legend location ('off' to disable).
```

```
% "titlestr" is a character array containing a human readable plot title.
% "fname" is the filename to write to.
%
% No return value.
```

10.15 euPlot_plotOverlappingRoses.m

```
% function euPlot_plotOverlappingRoses( ...
%   hist_list, cursor_list, legendpos, titlestr, fname )
%
% This generates a polar plot representing multiple overlaid histograms.
% The first and last bin of the histogram are connected (plots are closed
% curves).
%
% Empty legend labels suppress rendering of that legend label.
%
% "hist_list" is a cell array. Each cell contains a cell array describing
%   the histograms to plot: { counts, bin_edges, colour, legend_label }
% "cursor_list" is a cell array. Each cell contains a cell array describing
%   radial cursors to plot: { angles, colour, legend_label }
% "legendpos" is the legend location ('off' to disable).
% "titlestr" is a character array containing a human-readable plot title.
% "fname" is the filename to write to.
%
% No return value.
```

10.16 euPlot_plotZoomedWaves.m

```
% function euPlot_plotZoomedWaves( wave_list, flag_list, vert_cursor_list, ...
%  horiz_cursor_list, window_sizes, size_labels, max_count_per_size, ...
%   legendpos, titlebase, fnamebase )
%
% This generates a series of plots of multiple signals, stepping the plot
% window across the full time span, with multiple window sizes.
%
% Empty legend labels suppress rendering of that legend label.
%
% "wave_list" is a cell array. Each cell contains a cell array describing
%   the waves to plot: { time_series, wave_series, colour, legend_label }.
% "flag_list" is a cell array. Each cell contains a cell array describing
%   boolean data to plot: { time_series, flag_series, colour, legend_label }.
% "vert_cursor_list" is a cell array. Each cell contains a cell array
%   describing events to plot: { event_times, colour, legend_label }.
% "horiz_cursor_list" is a cell array. Each cell contains a cell array
%   describing horizontal lines to plot: { line_yvals, colour, legend_label }.
% "window_sizes" is a vector containing plot durations in seconds, stepped
```

```
%   across the time series.
% "size_labels" is a filename-safe label used when creating filenames and
%   annotating titles for plots that use a given window size.
% "max_count_per_size" is a scalar indicating the maximum number of plots
%   to emit at a given size level. Plots are spaced evenly within and
%   centered on the full time span.
% "legendpos" is the legend location ('off' to disable).
% "titlebase" is a prefix to use when constructing human-readable titles.
% "fnamebase" is a prefix to use when constructing filenames.
%
% No return value.
```

10.17 euPlot_rankMatrixData.m

```
% function [ rankreport ranktable ] = rankMatrixData( ...
%   matrixdata, dataformat, datatitle, ...
%   columncats, columnformat, columntitle, desiredcolidx, ...
%   chanlabelsraw, chanlabelscooked, sortwanted )
%
% This searches a matrix containing per-channel data, sorts it along a
% specified column, and produces a human-readable report describing the
% ranked channel values. It also produces an annotated table with the
% sorted matrix data suitable for writing to a CSV file.
%
% "matrixdata" is a Nchans x Ncols matrix of data values.
% "dataformat" is a sprintf template for formatting matrix data values.
% "datatitle" is a human-readable character array describing what the data is.
% "columncats" is a vector or cell array containing column category
%   values or labels. If this is empty, category information is omitted from
%   the report and output table. NOTE - If these are text labels, the labels
%   must be valid Matlab table column names!
% "columnformat" is a sprintf template for formatting column category
%   values, if they're numeric (vector rather than cell array).
% "columntitle" is a human-readable character array describing what type of
%   data the categories are.
% "desiredcolindex" is an index into "matrixdata" and "columncats" indicating
%   the desired column to sort on.
% "chanlabelsraw" is a cell array containing raw channel names.
% "chanlabelscooked" is a cell array containing cooked channel names.
% "sortwanted" indicates how to sort; 'min' finds the smallest values, 'max'
%   finds the largest values, 'absmin' the smallest magnitudes, and 'absmax'
%   the largest magnitudes.
%
% "rankreport" is a character array containing a human-readable report about
%   ranked channel values.
% "ranktable" is a table containing the data in "matrixdata", with an added
%   column containing channel names, and with the first row containing
%   column categories.
```

Chapter 11

"euTools" Functions

11.1 euTools_guessBadChannelsSpect.m

```
% function [ actualconfig chanlabels changoodvec ...
%   spectpower tonepower pcacoords ...
%   spectclusters toneclusters pcaclusters ] = ...
%   euTools_guessBadChannelsSpect( checkfolder, checkchans, config )
%
% This attempts to identify bad channels within ephys data by looking for
% channels with abnormal spectra compared to the group as a whole.
%
% "checkfolder" is the folder to read ephys data from (via Field Trip hooks).
% "checkchans" is a cell array containing the names of channels to read.
%   This can contain ft_channelselection patterns. If this is empty, all
%   channels are read.
% "config" is a structure with zero or more of the following fields.
%   Missing fields are filled with default values:
%   "timestart" is the starting time to read from, in seconds.
%   "duration" is the number of seconds of data to read.
%   "freqbinedges" is a vector containing bin edges for binning frequency.
%   "pcadims" is the number of principal components to extract when
%     performing dimensionality reduction for clustering.
%   "raw_clustcounts" is a vector containing cluster counts to test when
%     clustering based on individual band or tone powers.
%   "pca_clustcounts" is a vector containing cluster counts to test when
%     clustering the PCA output.
%   "raw_kmeans_repeats" is the number of times to run k-means (the
%     "Replicates" parameter) for band and tone power clustering.
%   "pca_kmeans_repeats" is the number of times to run k-means (the
%     "Replicates" parameter) for PCA output clustering.
%   "pca_reject_threshold" is the nlProc_getOutliers() threshold to use
%     for detecting bad channels. This is a multiple of the
%     median-to-quartile distance; the default is 3.0.
%
```

```
% "actualconfig" is a copy of "config" with missing values filled in.
% "chanlabels" is a nChans x 1 cell array containing the names of the
%   channels that were actually read.
% "changoodvec" is a nChans x 1 boolean vector indicating whether each
%   channel was "good".
% "spectpower" is a nChans x nBands matrix containing the total power in
%   each spectral bin for each channel.
% "tonepower" is a nChans x nBands matrix containing the ratio of the maximum
%   spectral power to the median spectral power in each spectral bin for each
%   channel. This is the height of pure tones over the noise floor.
% "pcacoords" is a nChans x nComponents matrix containing the coordinates of
%   each channel in PCA space (dimensionally reduced space).
% "spectclusters" is a nChans x nBands matrix containing cluster numbers for
%   each channel after performing k-means clustering of spectral power for
%   channels within each band.
% "toneclusters" is a nChans x nBands matrix containing cluster numbers for
%   each channel after performing k-means clustering of tone power ratio for
%   channels within each band.
% "pcaclusters" is a nChans x 1 vector containing cluster numbers for each
%   channel after performing k-means clustering in PCA space.
```

11.2 euTools_sanityCheckTree.m

```
% function [ reportshort reportlong folderdata ] = ...
%   euTools_sanityCheckTree( startdir, config )
%
% This function searches the specified directory tree for Open Ephys and
% Intan ephys data folders, opens the ephys data files, and checks for
% signal anomalies such as drop-outs and artifacts.
%
% NOTE - This tells Field Trip to use the LoopUtil file I/O hooks. So, it'll
% only work on folders that store data in a way that these support.
%
% NOTE - Channel correlation time goes up as the square of the number of
% channels!
%
% NOTE - Channel correlation may give misleading results if different probes
% are compared to each other. The workaround is to call this function with
% "chanpatterns" set to match a single probe's channels, if that becomes a
% problem.
%
% "startdir" is the root directory of the tree to search.
% "config" is a structure with some or all of the following fields. Missing
%   fields are set to default values.
%
%   "wantprogress" is true for progress reports and false otherwise.
%   "readposition" is a number between 0 and 1 indicating where to start
%     reading in the ephys data.
```

```
%   "readduration" is the number of seconds of ephys data to read.
%   "chanpatterns" is a cell array containing channel label patterns to
%     process. These are passed to ft_channelselection().
%
%   "notchfreqs" is a list of frequencies to notch-filter.
%   "notchbandwidth" is the notch bandwidth to use when filtering.
%   "lowpasscorner" is the low-pass-filter corner used for separating LFP
%     and spike data.
%   "lowpassrate" is the sampling rate to use when downsampling LFP data.
%
%   "wantquantization" is true to check for quantization and false otherwise.
%   "quantization_bits" is the minimum acceptable number of bits of dynamic
%     range in the data.
%
%   "wantartifacts" is true to check for artifacts and drop-outs and false
%     otherwise.
%   "smoothfreq" is the approximate low-pass corner frequency for smoothing
%     before artifact and dropout checking.
%   "dropout_threshold" is the threshold for detecting drop-outs. This is a
%     multiple of the median value, and should be less than 1.
%   "artifact_threshold" is the threshold for detecting artifacts. This is
%     a multiple of the median value, and should be greater than 1.
%   "frac_samples_bad" is the minimum fraction of bad samples in a channel
%     needed for that channel to be flagged as bad.
%
%   "wantcorrel" is true to check for correlated groups of channels and false
%     otherwise.
%   "correlthreshabslfp" is the absolute r-value threshold for considering
%     LFP channels to be correlated. This should be in the range 0..1.
%   "correlthreshrellfp" is the relative r-value threshold for considering
%     LFP channels to be correlated. This should be greater than 1.
%   "correlthreshabsspike" is the absolute r-value threshold for considering
%     spike channels to be correlated. This should be in the range 0..1.
%   "correlthreshrelspike" is the relative r-value threshold for considering
%     spike channels to be correlated. This should be greater than 1.
%
%   "wantspect" is true to check the LFP power spectrum and false otherwise.
%   "spectrange" [ min max ] is the range of frequencies to perform the LFP
%     power spectrum fit over.
%   "spectbinwidth" is the relative width of LFP power spectrum frequency
%     bins. A value of 0.1 would mean a bin width of 0.1 times each bin's
%     center frequency.
%
% "reportshort" is a character vector containing a human-readable summary
%   of the sanity check.
% "reportlong" is a character vector containing a much more detailed
%   human-readable summary of the sanity check.
% "folderdata" is an array of structures with the following fields:
%
%   "datadir" is the path containing the ephys data files.
```

```
%   "config" is a copy of the configuration structure with missing values
%     set to appropriate defaults.
%   "ftheader" is the Field Trip header for the ephys data.
%   "samprange" [min max] is the range of samples read for the test.
%   "chanlist" is a cell array with the names of selected channels.
%
%   "isquantized" is a boolean vector indicating whether channel data showed
%     quantization.
%   "hadartifacts" is a boolean vector indicating whether channel data had
%     large amplitude excursions (electrical artifacts).
%   "haddropouts" is a boolean vector indicating whether channel data had
%     intervals with no data (usually a throughput problem).
%
%   "lfpchangroups" is a vector indicating which correlated LFP groups
%      channels were members of. Contacts on the same probe or from probes
%      that are very close to each other will tend to have correlated LFPs.
%      Channels that don't strongly correlate have NaN as a group.
%   "lfpgroupdefs" is a cell array with one cell per LFP group containing
%      vectors indicating which channels (from "chanlist") were in the group.
%   "rvalues_lfp" is a square matrix containing correlation coefficients
%      between LFP channels.
%   "spikechangroups" is a vector indicating which correlated spike groups
%      channels were members of. Floating channels in the same probe or
%      headstage will tend to have correlated high-frequency signals.
%      Channels that don't strongly correlate have NaN as a group.
%   "spikegroupdefs" is a cell array with one cell per spike group containing
%      vectors indicating which channels (from "chanlist") were in the group.
%   "rvalues_spike" is a square matrix containing correlation coefficients
%      between LFP channels.
%
%   "lfpfitexponents" is a vector containing the exponent of power-law fits
%      to each channel's LFP power spectrum. A clean LFP should be -1 to -2.
%   "lfpfitlabels" is a cell array containing a human-readable type label
%      for each channel's LFP power spectrum, per nlProc_examineLFPSpectrum.
```

Chapter 12

“euUSE” Notes

12.1 EVCODEDEFS.txt

```
There are two types of event code definition structure: "raw" and "cooked".


The "raw" event code definition structure is the structure returned by
calling "jsondecode" on the contents of the "eventcodes_USE_05.json" file.

This is a structure with one field per event code, with the field name being
the event code's name. Each field contains a structure with the following
fields (upper-case leading):

- "Value" is an integer value in the range 0..65025.
- "Description" is a character array with human-readable text describing this
  event code.

Note that there are often pairs of codes with the name "FooMin" and "FooMax"
describing ranges of code values that are interpreted per their descriptions.


The "cooked" event code definition structure has one field per event code,
with the field name being the event code's name. Each field contains a
structure with the following fields (lower-case):

- "value" is either a scalar integer value or a [min max] pair of integer
  values (describing a ranged code).
- "description" is a cell array containing character arrays with
  human-readable text describing this event code.
- "offset" (optional) is a number to be subtracted from the code value to
  convert it into a processed data value.
- "multiplier" (optional) is a factor by which the code value is to be
  multiplied (after offset subtraction) to convert it into a processed value.
```

```
When turning an event code value into a processed value, the formula used is:
  processed = multiplier * (raw - offset)

Pairs of raw codes of the form "FooMin" and "FooMax" are turned into single
cooked code definitions for the code "Foo". The description strings from
"FooMin" and "FooMax" are usually identical, but both are stored in the
cooked definition as a precaution.



This is the end of the file.
```

12.2 FILES.txt

```
Files that we read from USE, as of 03 Feb 2022:


The "RuntimeData" folder has files of interest in the following subfolders:

- "SerialSent" has messages sent from USE to the SynchBox, with Unity
  timestamps. These are per-trial plus a startup file.
- "SerialRecv" has messages sent from the SynchBox to USE, with both
  Unity and SynchBox timestamps. These are per-trial plus a startup file.
- "GazeData" has eye-tracker data in TSV format. These are per-trial.
- "FrameData" has USE state information, including eye tracker timestamps,
  in TSV format.

NOTE - Trial files are stored as "_N.txt", "_NN.txt", etc; either use the
third-party "sort_nat" to sort them, or sort data based on timestamps.


The "ProcessedData" folder has ".mat" files with data tables:

- "SerialData.mat" has SerialSentData and SerialRecvData tables.
- "RawGazeData" has the gaze data.


The "use/USE_Analysis" scripts on GitHub are used for converting raw into
processed data.

More recent ones are in "use_analysis" on bitbucket (private repository).
```

Chapter 13

"euUSE" Functions

13.1 euUSE_aggregateTrialFiles.m

```
% function tabdata = euUSE_aggregateTrialFiles(filepattern, sortcolumn)
%
% This reads the specified set of tab-delimited text files, aggregating the
% data contained within. The resulting combined table is sorted based on the
% specified column and returned.
%
% This is intended to be used with per-trial files, sorting on the timestamp.
%
% "filepattern" is the path-plus-wildcards file specifier to pass to dir().
% "sortcolumn" is the name of the table column to sort table rows with.
%
% "tabdata" is the resulting sorted merged table.
```

13.2 euUSE_alignTwoDevices.m

```
% function [ alignedevents timecorresp ] = euUSE_alignTwoDevices( ...
%   eventtables, firsttimecol, secondtimecol, alignconfig )
%
% This function performs time alignment between two devices by examining
% correlated event sequences between the devices.
%
% This is a wrapper for "euAlign_alignTables()".
%
% NOTE - If it can't find events in common, it centers the time series on
% each other (using the maximum extents of each source). If it can't even do
% that (for example if one source has no events of any type), it lines them
% up by fiat (declaring missing extents to be 0 to 3600 seconds).
% FIXME - We should really have a return flag to indicate this. Right now
% it just shows up on console output.
```

```
%
% "eventtables" is a Nx2 cell array. Each row contains two tables with
%   events that ostensibly match each other. The first row where both tables
%   are non-empty is used for alignment.
% "firsttimecol" is the name of the timestamp column in tables stored in
%   column 1 of "eventtables".
% "secondtimecol" is the name of the timestamp column in tables stored in
%   column 2 of "eventtables".
% "alignconfig" is a structure with zero or more of the following fields:
%   "coarsewindows" is a vector with coarse alignment window half-widths.
%   "medwindows" is a vector with medium alignment window half-widths.
%   "finewindow" is the window half-width for final non-uniform alignment.
%   "outliersigma" is the threshold for rejecting spurious matches.
%   "verbosity" is 'verbose', 'normal', or 'quiet'.
%   Missing fields will be set to reasonable defaults.
%
% "alignedevents" is a copy of "eventtables" with the tables in column 1
%   augmented with "secondtimecol" timestamps and the tables in column 2
%   augmented with "firsttimecol" timestamps.
% "timecorresp" is a table with "firsttimecol" and "secondtimecol" columns,
%   containing tuples with known-good time alignment. This is the "canonical"
%   set of timestamps used to interpolate time values between tables.
```

13.3 euUSE_cleanEventsTabular.m

```
% function newtable = euUSE_cleanEventsTabular( oldtable, datacol, timecol )
%
% This processes a data table of raw code word events, removing entries that
% have a value of zero (the idle state) and merging events that are one
% sample apart (which happens when event codes change at a sampling boundary).
%
% "oldtable" is a table containing event tuples to process.
% "datacol" is the label of the column that contains data values.
% "timecol" is the label of the column that contains timestamps (in samples).
%
% "newtable" is the revised table.
```

13.4 euUSE_deglitchEvents.m

```
% function [ newvals newtimes ] = euUSE_deglitchEvents( oldvals, oldtimes )
%
% This checks a list of event timestamps for events that are one sample
% apart, and merges them. This happens when event codes change at a sampling
% boundary.
%
% This will also catch the situation where events are reported multiple
```

```
% times with the same timestamp.
%
% "oldvals" is a list of event data samples (integer data values).
% "oldtimes" is a list of event timestamps (in samples).
%
% "newvals" is a revised list of event data samples.
% "newtimes" is a revised list of event timestamps.
```

13.5 euUSE_getEventCodeDefOverrides.m

```
% function defoverrides = euUSE_getEventCodeDefOverrides()
%
% This function provides a default "hints" structure for parsing event code
% definitions. This is mostly used for changing the range of codes that have
% ranged values encoded.
%
% NOTE - This is entirely composed of magic values reflecting our current
% USE configuration, and will have to be manually edited if that changes.
%
% "defoverrides" is a structure suitable for passing to
%   euUSE_parseEventCodeDefs().
```

13.6 euUSE_lookUpEventCode.m

```
% function [ codedata codelabel ] = euUSE_loopUpEventCode( codeword, codedefs)
%
% This function looks up the event code definition for a specified code word.
% The translated event code data value, if applicable, is generated.
%
% "codeword" is the event code word to translate.
% "codedefs" is a "cooked" event code definition structure per
%   "EVCODEDEFS.txt".
%
% "codedata" is a translated data value corresponding to this event code, if
%   translation is appropriate, or a copy of "codeword" if not.
% "codelabel" is the field name of the corresponding entry in "codedefs", if
%   the code was understood. This is usually a human-readable code name. If
%   the code was not recognized, this is an empty character array ('').
```

13.7 euUSE_parseEventCodeDefs.m

```
% function cookeddefs = euUSE_parseEventCodeDefs( rawdefs, overrides )
%
```

```
% This parses raw event code definitions (as given by "jsondecode") and
% builds a structure containing processed event definitions.
%
% "rawdefs" is the decoded JSON event code definition structure.
% "overrides" is a structure containing specific event code fields that
%   overwrite automatically-generated fields in "cookeddefs".
%
% "cookeddefs" is a structure with fields indexed by event code names, each
%   of which contains a structure with the following fields (per
%   EVCODEDEFS.txt):
%   "value" is a scalar or a two-element vector, containing the event code
%     value or range of values ( [ min max ] ) for this code.
%   "description" is a cell array containing human-readable description
%     strings for the event code. There may be multiple strings if the code
%     is defined by multiple entries in "rawdefs" (as with ranged codes).
%   "offset" (optional) is a number to be subtracted from the code value
%     to convert it into a processed data value.
%   "multiplier" (optional) is a factor by which the code value is to be
%     multiplied to convert it into a processed value. This is not
%     automatically generated.
%
% When turning an event code value into a processed value, the formula used
% is:  processed = multiplier * (raw - offset)
%
% The idea is that the automatic parsing generates reasonable guesses for the
% interpretation of "FooMin" and "FooMax" definition pairs, and the
% "overrides" structure can modify these interpretations for known cases
% where automatic guesses aren't correct.
```

13.8 euUSE_parseSerialRecvData.m

```
% function [ evsynchA evsynchB evrwdA evrwdB evcodes ] = ...
%   euUSE_parseSerialRecvData( serialrecvdata, codeformat )
%
% This function parses the "SerialRecvData" table from a "serialData"
% structure, read from the "SerialData.mat" file produced by the USE
% processing scripts. It may alternatively be read by using the
% euUSE_readRawSerialData() function.
%
% This extracts event data. To extract analog data, use
% euUSE_parseSerialRecvDataAnalog().
%
% The input table contains communication received by Unity from the SynchBox,
% which includes Unity and SynchBox timestamps. Timing, reward, and event
% code messages are parsed, and are returned as separate tables. Each table
% contains Unity and SynchBox timestamps (converted to seconds); the reward
% tables also contain reward pulse duration in seconds, and event code
% tables contain the event code value.
```

```
%
% Event codes may be in any of several formats; "word" format codes are
% preserved as-is, while "byte" format codes are shortened to 8 bits. The
% byte may be taken from the most-significant or least-significant 8 bits
% of the code word. For "dupbyte", the most significant and least significant
% 8 bits are expected to contain the same values; any codes that don't are
% rejected.
%
% "serialrecvdata" is the data table containing raw inbound serial data.
% "codeformat" is 'word' (for 16-bit words), 'hibyte' for MS bytes, 'lobyte'
%   for LS bytes, and 'dupbyte' for MS and LS both replicating a byte value.
%
% "evsynchA" and "evsynchB" are tables containing timing A and B events.
%   Table columns are 'unityTime' and 'synchBoxTime'.
% "evrwdA" and "evrwdB" are tables containing reward trigger A and B events.
%   Table columns are 'unityTime', 'synchBoxTime', and 'pulseDuration'.
% "evcodes" is a table containing event code events. Table columns are
%   'unityTime', 'synchBoxTime', and 'codeValue'.
```

13.9 euUSE_parseSerialRecvDataAnalog.m

```
% function analogdata = euUSE_parseSerialRecvDataAnalog( serialrecvdata )
%
% This function parses the "SerialRecvData" table from a "serialData"
% structure, read from the "SerialData.mat" file produced by the USE
% processing scripts. It may alternatively be read by using the
% euUSE_readRawSerialData() function.
%
% This extracts analog data sent by the synchbox. To extract event
% information, use euUSE_parseSerialDecvData().
%
% "serialrecvdata" is the data table containing raw inbound serial data.
%
% "analogdata" is a tale containing five analog channels and two flags.
%   Table columns are 'unityTime', 'synchBoxTime', 'joyX', 'joyY', 'joyZ',
%   'lightL', 'lightR', 'flagL', and 'flagR'.
```

13.10 euUSE_parseSerialSentData.m

```
% function [ evrwdA evrwdB evcodes ] = ...
%   euUSE_parseSerialSentData( serialsentdata, codeformat )
%
% This function parses the "SerialSentData" table from a "serialData"
% structure, read from the "SerialData.mat" file produced by the USE
% processing scripts. It may alternatively be read by using the
% euUSE_readRawSerialData() function.
```

```
%
% This table contains communication sent from Unity to the SynchBox, which
% includes Unity timestamps (but no synchbox timestamps). Reward and event
% code messages are parsed, and are returned as separate tables. Each table
% contains Unity timestamps (converted to seconds); the reward tables also
% contain reward pulse duration in seconds, and event code tables contain
% the event code value.
%
% Event codes may be in any of several formats; "word" format codes are
% preserved as-is, while "byte" format codes are shortened to 8 bits. The
% byte may be taken from the most-significant or least-significant 8 bits
% of the code word. For "dupbyte", the most significant and least significant
% 8 bits are expected to contain the same values; any codes that don't are
% rejected.
%
% "serialsentdata" is the data table containing raw outbound serial data.
% "codeformat" is 'word' (for 16-bit words), 'hibyte' for MS bytes, 'lobyte'
%   for LS bytes, and 'dupbyte' for MS and LS both replicating a byte value.
%
% "evrwdA" and "evrwdB" are tables containing reward trigger A and B events.
%   Table columns are 'unityTime' and 'pulseDuration'.
% "evcodes" is a table containing event code events. Table columns are
%   'unityTime' and 'codeValue'.
```

13.11 euUSE_readAllEphysEvents.m

```
% function [ ttlevents cookedevents ] = ...
%   euUSE_readAllEphysEvents( ephysfolder, bitsignaldefs, codesignaldefs, ...
%   evcodedefs, codebytes, codeendian )
%
% This function reads TTL events from one ephys device and parses these into
% synch, reward, and event code events.
%
% This is a wrapper for the following functions:
%   ft_read_header()
%   ft_read_event()
%   euFT_getSingleBitEvent()
%   euFT_getCodeWordEvent()
%   euUSE_cleanEventsTabular()
%   euUSE_reassembleEventCodes()
%
% NOTE - Channel specifiers (individual or wildcard-based) have the format
% used by ft_channelselection().
%
% "ephysfolder" is the folder containing "structure.oebin", "info.rhd", or
%   "info.rhs".
% "bitsignaldefs" is a structure indexed by signal name, with each field
%   containing the TTL channel name with that signal's events.
```

```
%   If the structure is empty, no single-bit signals are recorded.
% "codesignaldefs" is a structure with the following fields:
%   "signameraw" is the output event code signal name for untranslated bytes.
%   "signamecooked" is the output event code signal name for event codes.
%   "channame" is the TTL channel name to convert. This may be a single
%     channel (for word-based TTL data) or a wildcard expression (for
%     single-bit TTL data).
%   "bitshift" is the number of bits to shift to the right, if reassembling
%     from bits. This is also used to compensate for 1-based numbering (the
%     channel names for bit lines are assumed to start at 0).
%   If there's only one matching channel, it's assumed to contain word data.
%   otherwise it's assumed to contain bit data.
%   If the structure is empty, no event codes are recorded.
% "evcodedefs" is a USE event code definition structure per EVCODEDEFS.txt.
% "codebytes" is the number of bytes used to encode each event code.
%   This defaults to 2 if unspecified.
% "codeendian" is 'big' if the most significant byte is received first or
%   'little' if the least-significant byte is received first.
%   This defaults to 'big' if unspecified.
%
% "ttlevents" is the raw TTL event list returned by ft_read_event().
% "cookedevents" is a structure with one field per signal listed in
%   "bitsignaldefs" and "codesignaldefs". Each field contains a table
%   of matching events whose columns correspond to the event list's fields.
%   These tables may be empty if no events were found.
```

13.12 euUSE_readAllUSEEvents.m

```
% function [ boxevents gameevents evcodedefs ] = ...
%   euUSE_readAllUSEEvents( runtimefolder, codeformat, codebytes, codeendian )
%
% This function reads and parses serial data from a USE "RuntimeData" folder.
% This gives events and timestamps from USE and from the SynchBox.
%
% This is a wrapper for the following functions:
%   euUSE_readRawSerialData()
%   euUSE_parseSerialRecvData()
%   euUSE_parseSerialSentData()
%   euUSE_readEventCodeDefs()
%   euUSE_reassembleEventCodes()
%
% "runtimefolder" is the path to the "RuntimeData" folder.
% "codeformat" is the event code format used by the SynchBox. This is 'word',
%   'hibyte', 'lobyte', or 'dupbyte', per euUSE_parseSerialRecvData().
%   This defaults to 'dupbyte' if unspecified.
% "codebytes" is the number of bytes used to encode each event code.
%   This defaults to 2 if unspecified.
% "codeendian" is 'big' if the most significant byte is received first or
```

```
%   'little' if the least-significant byte is received first.
%    This defaults to 'big' if unspecified.
%
% "boxevents" is a structure with fields "synchA", "synchB", "rwdA", "rwdB",
%   "rawcodes", and "cookedcodes". These each contain tables of events.
%   Common columns are 'unityTime' and 'synchBoxTime'. Additional columns are
%   'pulseDuration' (for rewards), 'codeValue' (for raw codes), and
%   'codeWord', 'codeData', and 'codeLabel' (for cooked codes).
% "gameevents" is a structure with fields "rwdA", "rwdB", "rawcodes", and
%   "cookedcodes". These each contain tables of events. All tables contain
%   a 'unityTime' column. Additional columns are 'pulseDuration' (for
%   rewards), 'codeValue' (for raw codes), and 'codeWord', 'codeData', and
%   'codeLabel' (for cooked codes).
% "evcodedefs" is a USE event code definition structure per EVCODEDEFS.txt.
```

13.13 euUSE_readEventCodeDefs.m

```
% function defscooked = euUSE_readEventCodeDefs( runtimedir )
%
% This function reads the "eventcodes_USE_05.json" file and returns a
% "cooked" event code definition structure, per EVCODEDEFS.txt.
%
% "runtimedir" is the "RuntimeData" directory location.
%
% "defscooked" is a "cooked" event code definition structure.
```

13.14 euUSE_readRawFrameData.m

```
% function framedata = euUSE_readRawFrameData( runtimedir )
%
% This function looks for "*_Trial_(number).txt" files in the FrameData folder
% in the specified directory, and converts them into an aggregated Matlab
% table with rows sorted by timestamp.
%
% New timestamp columns ("SystemTimeSeconds" and "EyetrackerTimeSeconds")
% are generated from the respective native timestamp columns.
%
% "runtimedir" is the "RuntimeData" directory location.
%
% "framedata" is an aggregated data table.
```

13.15 euUSE_readRawGazeData.m

```
% function gazedata = euUSE_readRawGazeData( runtimedir )
%
% This function looks for "*_Trial_(number).txt" files in the GazeData folder
% in the specified directory, and converts them into an aggregated Matlab
% table with rows sorted by timestamp.
%
% A new timestamp column ("time_seconds") is generated from the native gaze
% timestamp column.
%
% "runtimedir" is the "RuntimeData" directory location.
%
% "gazedata" is an aggregated data table.
```

13.16 euUSE_readRawSerialData.m

```
% function [ sentdata recvdata ] = euUSE_readRawSerialData( runtimedir )
%
% This function looks for "*_Trial_(number).txt" files in the SerialSent
% and SerialRecv folders in the specified directory, and converts them
% into aggregated Matlab tables with rows sorted by Unity timestamp.
%
% "runtimedir" is the "RuntimeData" directory location.
%
% "sentdata" is aggregated data from trial files in the "SerialSent" folder.
% "recvdata" is aggregated data from trial files in the "SerialRecv" folder.
```

13.17 euUSE_reassembleEventCodes.m

```
% function [ cookedcodes cookedindices ] = euUSE_reassembleEventCodes( ...
%   rawcodes, codedefs, codebytes, codeendian, rawcolumn )
%
% This translates a data table containing raw (byte) event codes into a
% table containing cooked (word) event codes.
%
% This recovers from dropped bytes. Anything unrecognized (due to dropped
% bytes or just not being in the definition file) is tagged with an empty
% character array as the "codeLabel".
%
% "rawcodes" is a table containing a raw code value column and optionally
%   other columns.
% "codedefs" is a structure containing "cooked" event code definitions per
%   "parseEventCodeDefs" and "EVCODEDEFS.txt".
% "codebytes" is the number of bytes per cooked code.
```

```
% "codeendian" is 'big' if the most-significant byte is received first
%   or 'little' if the least-significant byte is received first.
% "rawcolumn" is the name of the column to read raw codes from.
%
% "cookedcodes" is a table with the following columns:
%   "codeWord" is the reconstructed event code word value.
%   "codeData" is (code number - offset) * multiplier, per "EVCODEDEFS.txt".
%   "codeLabel" is the corresponding "codedefs" field name for this code.
%     This is usually a human-readable code name.
%   Other columns from "rawcodes" are copied for rows corresponding to the
%   first byte of each cooked code. These are typically timestamps.
% "cookedindices" is a vector with length equal to the number of rows in
%   "cookedcodes", containing indices (row numbers) pointing to the
%   locations of the corresponding first code bytes in "rawcodes".
```

13.18 euUSE_removeLargeTimeOffset.m

```
% function [ reftime newsignals ] = ...
%   euUSE_removeLargeTimeOffset( oldsignals, timecolumn, desiredreftime )
%
% This function subtracts a large time offset from all signal tables in
% the provided structure.
%
% This is intended to be used to modify Unity timestamps, which are relative
% to 1 Jan 1970.
%
% "oldsignals" is a structure with fields that each contain a table of
%   timestamped events.
% "timecolumn" is the name of the table column that contains timestamps.
% "desiredreftime" is a desired time offset to be subtracted from all
%   timestamps. If this argument is omitted, and arbitrary offset is chosen.
%
% "reftime" is the time value that was subtracted from all timestamps.
% "newsignals" is a copy of "oldsignals" with "reftime" subtracted from all
%   tables' timestamp columns.
```

13.19 euUSE_segmentTrialsByCodes.m

```
% function [ pertrialtabs alltrialtab ] = ...
%   euUSE_segmentTrialsByCodes( rawtab, labelfield, valuefield, discardbad )
%
% This function processes a table containing event code sequences, segmenting
% the sequence into trials and optionally discarding bad trials.
%
% Trials are demarked by 'TrlStart' and 'TrlEnd' codes. "Good" trials are
% those for which 'TrialNumber' incremented.
```

```
%
% Additional columns in "rawtab" are copied to the output tables. These are
% typically things like timestamps.
%
% "rawtab" is the table to process. It must contain columns with event code
%   labels and (if discarding bad trials) event code data values.
% "labelfield" is the name of the table column that has code label character
%   arrays.
% "valuefield" is the name of the table column that has code data values.
%   This is ignored if bad trials aren't being filtered.
% "discardbad" is true if bad trials are to be discarded and false otherwise.
%
% "pertrialtabs" is a cell array containing event code sequence tables for
%   each trial.
% "alltrialtab" is an event code sequence table containing all trials
%   (equal to the concatenated contents of "pertrialtabs").
```

13.20 euUSE_translateGazeIntoSignals.m

```
% function dataset = euUSE_translateGazeIntoSignals( ...
%   gazetable, timecol, ignorecols, samprate )
%
% This translates tabular gaze data read from USE into a Field Trip data
% structure containing continuous waveform data.
%
% NOTE - The signal names produced by this will vary depending on the
% eye-tracker used.
%
% NOTE - Do not pick a sampling rate that exactly matches the eye-tracker
% rate. That will almost certainly result in beat frequencies in the data.
%
% "gazetable" is a table containing raw USE gaze data.
% "timecol" is the label of the table column to read timestamps from.
% "ignorecols" is a cell array containing labels of columns to not copy.
% "samprate" is the sampling rate to use for the output data.
%
% "dataset" is a ft_datatype_raw structure containing gaze data signals.
```

Chapter 14

"euUtil" Notes

14.1 LOUIELOGFILES.txt

```
Louie's log files are Matlab code with the following format:


The first line may be a function declaration:

function D = (function name)

...Otherwise, it's just inline code setting the value of "D".


The log file initializes a structure array "D", adding records with some or
all of the following fields. These are all character arrays unless otherwise
indicated (even if holding numeric data):

"day" is a human-readable day name.
"week" is the week number (of the experiment, not the year).
"date" is a datestamp formatted as 'YYYY-MM-DD'.
"rec_day" is the recording day number in the experiment.
"rec_session" is the recording session number in that day.
"start_time" has the form 'HH:MM'.
"session_length" has the form 'H:MM'.
"weight" has the form 'nn.n'.
"chairing_comments" describes anything noteworthy about chairing.
"chamber_quality" describes health of the chamber site.
"CHANNELS_neuro" is a cell array of channel labels. These are typically
  the names I'd used for EIB channels.
"CHANNELS_area" is a cell array of labels indicating the brain region that
  each channel was recording from.
"CHANNELS_loc" is a cell array of vectors of the form [ X Y Z ], giving the
  estimated coordinates each channel was recording from (mm, mm, microns).
"impedance_test" is a cell array of scalars, giving the impedance of each
```

```
  channel in kOhms.
"CHANNELS_stim" is a cell array of labels. This is typically a subset of
  CHANNELS_neuro, and indicates which channels stimulation was performed on.
"stim_amp" is a cell array containing one scalar indicating stimulation
  current in uA.
"task_version" is the version label of the game task.
"blockDef" is a filename (without path) of the block definition file used.
"performance" describes anything noteworthy about behavior during the task.
"condition" describes the task conditions under which stimulation was
  delivered.
"blocks" is the number of blocks completed.
"trials" is the number of trials completed.
"correct" is the number of trials completed correctly.
"ml_reward" is the amount of fluid reward delivered, in mL.
"dataset" is the name of the folder containing ephys and game data.
"other_comments" describes anything else noteworthy about the session.


NOTE - The fields that are expected will vary depending on the experiment
and on the analysis scripts used.


The library functions that read and process log file data expect the
following:

- Each record is preceded by "iD = numel(D) + 1;".
  This is used for segmenting records during parsing.

- Either "dataset" or "date" exists (preferably "dataset").
  euUtil_getLouieFoldersAndLogs() uses this to associate folders with logs.



(This is the end of the file.)
```

Chapter 15

"euUtil" Functions

15.1 euUtil_calcAllCircularStats.m

```
% function [ cmean cvar lindev fwhmraw fwhmabove ] = ...
%   euUtil_calcAllCircularStats( angleseries )
%
% This calculates circular statistics for an angle series using
% "nlProc_calcCircularStats", and also estimates the FWHM of the series
% using "bbCalc_estimateFWHM".
%
% "angleseries" is a vector containing angles in radians.
%
% "cmean" is the circular mean of the angle series.
% "cvar" is the circular variance of the angle series. Phase locking value
%   is (1 - cvar).
% "lindev" is the linear standard deviation of the angle series. For tightly
%   clustered angles, this can be more intuitive than circular variance.
% "fwhmraw" is the estimated full-width half-maximum of the angle distribution
%   measured relative to amplitude zero.
% "fwhmabove" is the estimated full-width half-maximum of the angle
%   distribution measured relative to the distribution's estimated "floor".
```

15.2 euUtil_concatenateCellStrings.m

```
% function newstring = euUtil_concatenateCellStrings( oldstrings, suffixstr )
%
% This accepts a cell array containing character vector and concatenates them
% into a single character vector. If a suffix is supplied, that suffix is
% appended to every input character vector. If no suffix is suppled, the
% value of "sprintf('\n')" is used (platform-agnostic line break).
%
% "oldstrings" is a cell array containing character vectors to concatenate.
```

```
% "suffixstr" is a cell array containing a suffix to append to all inputs.
%   If this is omitted, the value of "sprintf('\n')" is used.
%
% "newstring" is a character vector containing the concatenated inputs.
```

15.3 euUtil_getExperimentFolders.m

```
% function [ dirs_opene dirs_intanrec dirs_intanstim dirs_use ] = ...
%   euUtil_getExperimentFolders( topdir )
%
% This function searches a directory tree, looking for subfolders containing
% Open Ephys data (structure.oebin), Intan data (info.rhs/info.rhd), and
% USE data (RuntimeData folder).
%
% "topdir" is the folder to search. This may contain wildcards.
%
% "dirs_opene" is a cell array containing paths to Open Ephys folders.
% "dirs_intanrec" is a cell array containing paths to Intan recorder folders.
% "dirs_intanstim" is a cell array with paths to Intan stimulator folders.
% "dirs_use" is a cell array with paths to USE "RuntimeData" folders.
```

15.4 euUtil_getLabelChannelMap_OEv5.m

```
% function [ maplabelsraw maplabelscooked ] = ...
%   euUtil_getLabelChannelMap_OEv5( mapdir, datadir )
%
% This function loads an Open Ephys channel map, loads a saved dataset's
% header, and translates the numeric channel map into a label-based channel
% map.
%
% This is a wrapper for "euUtil_getOpenEphysChannelMap_v5",
% "nlIO_readFolderMetadata", and "nlFT_getLabelChannelMapFromNumbers".
%
% If a channel map couldn't be built, empty cell arrays are returned.
%
%
% "mapdir" is the folder to search for channel map files (including saved
%   Open Ephys v5 configuration files).
% "datadir" is the folder to search for Open Ephys datasets.
%
% "maplabelsraw" is a cell array containing raw channel names that correspond
%   to the names in "maplabelscooked".
% "maplabelscooked" is a cell array containing cooked channel names that
%   correspond to the names in "maplabelsraw".
```

15.5 euUtil_getLouieFoldersAndLogs.m

```
% function sessionlist = ...
%   euUtil_getLouieFoldersAndLogs( sessionfolders, logfilepatterns );
%
% This fetches recording/USE folder locations and session metadata for all
% recording sessions found in a specified list of top-level directories.
%
% NOTE - Any folders that we couldn't find log data for are dropped.
%
% "sessionfolders" is a cell array containing paths to folders to
%   recursively probe for ephys data.
% "logfilepatterns" is a cell array containing filename patterns (including
%   wildcards) that may match Louie's log files.
%
% "sessionlist" is structure array, with the following fields:
%   "logdata" is a record structure from Louie's log file.
%   "folders_openephys" is a cell array of Open Ephys data folder names.
%   "folders_intanrec" is a cell array of Intan recording controller folders.
%   "folders_intanstim" is a cell array of Intan stim controller folders.
%   "folders_game" is a cell array of game data folder name.
```

15.6 euUtil_getLouieLogData.m

```
% function recdata = euUtil_getLouieLogData( infile, datewanted )
%
% This function reads one of Louie's log files and extracts the record for
% the specified date.
%
% If "datewanted" is empty, all records are returned (as a struct array).
%
% If multiple records match the date, the first one is returned.
% If the date isn't matched, an empty struct array is returned.
%
% "infile" is the name of the file to read (containing Matlab code).
% "datewanted" is a character array containing the desired "date" field
%   contents, or an empty character array to return all records.
%
% "recdata" is a struct array containing zero or more matching records.
%
%
% Louie's epxeriment logs are Matlab functions that return a structure array
% named "D" with each day recorded in a struct.
%
% NOTE - This manually parses Louie's log into record segments and tries
% to run each segment's code individually. If any given segment fails to run
% (which happens), the location of the offending segment is reported.
```

```
%
% WARNING - This blindly trusts that the code in the log file is safe!
```

15.7 euUtil_getOpenEphysChannelMap_v5.m

```
% function chanmap = euUtil_getOpenEphysChannelMap_v5( inputfolder )
%
% This searches the specified tree looking for Open Ephys configuration
% files and channel mapping files (anything with "config" or "mapping" in
% the filename). Channel maps are extracted, and the first map found is
% returned. If no maps are found, an empty structure array is returned.
%
% This is a wrapper for "euUtil_getOpenEphysConfigFiles",
% "nlOpenE_parseChannelMapJSON_v5", and "nlOpenE_parseChannelMapXML_v5".
%
% "inputfolder" is the top-level folder to search.
%
% "chanmap" is a structure with the following fields:
%   "oldchan" is a vector indexed by new channel number containing the old
%     channel number that maps to each new location, or NaN if none does.
%   "oldref" is a vector indexed by new channel number containing the old
%     channel number to be used as a reference for each new location, or
%     NaN if unspecified.
%   "isenabled" is a vector of boolean values indexed by new channel number
%     indicating which new channels are enabled.
```

15.8 euUtil_getOpenEphysConfigFiles.m

```
% function [ configfiles mapfiles ] = euUtil_getOpenEphysConfigFiles( topdir )
%
% This function looks for files with "Config" or "Mapping" in the name (not
% case-sensitive) and reports their full names (including path).
%
% This is intended to be used to find Open Ephys channel mapping and
% configuration files.
%
% "topdir" is the top-level folder to search (usually the Open Ephys folder).
%
% "configfiles" is a cell array containing full filenames that had "config".
% "mapfiles" is a cell array containing full filenames that had "mapping".
```

15.9 euUtil_makePrettyTime.m

```
% function durstring = euUtil_makePrettyTime(dursecs)
%
% This formats a duration (in seconds) in a meaningful human-readable way.
% Examples would be "5.0 ms" or "5d12h".
%
% "dursecs" is a duration in seconds to format. This may be fractional.
%
% "durstring" is a character array containing a terse human-readable
%   summary of the duration.
```

15.10 euUtil_makeSafeString.m

```
% function [ newlabel newtitle ] = euUtil_makeSafeString( oldstring )
%
% This function makes label- and title-safe versions of an input string.
% The label-safe version strips anything that's not alphanumeric.
% The title-safe version replaces stripped characters with spaces.
%
% This is more aggressive than filename- or fieldname-safe strings; in
% particular, underscores are interpreted as typesetting metacharacters
% in plot labels and titles.
%
% "oldstring" is a character vector to convert.
%
% "newlabel" is a character vector with only alphanumeric characters.
% "newtitle" is a character vector with non-alphanumeric characters replaced
%   with spaces.
```

15.11 euUtil_makeSafeStringArray.m

```
% function [ newlabels newtitles ] = euUtil_makeSafeStringArray( oldstrings )
%
% This function calls "euUtil_makeSafeString" on all elements of a
% one-dimensional cell array, making "safe" variants of all elements.
% The "label-safe" versions strip anything that's not alphanumeric.
% The "title-safe" versions replace stripped characters with spaces.
%
% This is more aggressive than filename- or fieldname-safe strings; in
% particular, underscores are interpreted as typesetting metacharacters in
% plot labels and titles.
%
% "oldstrings" is a 1D cell array containing character vectors to convert.
%
```

```
% "newlabels" is a cell array containing character vectors with only
%   alphanumeric characters.
% "newtitles" is a cell array containing character vectors with
%   non-alphanumeric characters replaced with spaces.
```

15.12 euUtil_parseDateNumber.m

```
% function [ yearnum, monthnum, monthshort monthlong, daynum ] = ...
%   euUtil_parseDateNumber( packeddatenum )
%
% This parses a date number that has the form YYYYMMDD, YYMMDD, or MMDD.
% This may be supplied as a number or as a character vector.
%
% "packeddatenum" is the date number to parse.
%
% "yearnum" is the year (YYYY or YY), or NaN if no year was provided.
% "monthnum" is the month number (expected to be 1-12; MM in the template).
% "monthshort" is a character vector with a three-letter month label.
% "monthlong" is a character vector with the full month name.
% "daynum" is the day number (DD in the template).
```

15.13 euUtil_pickBiggestEphysFolder.m

```
% function [ foldername foldersizes ] = ...
%   euUtil_pickBiggestEphysfolder( folderlist )
%
% This reads Field Trip headers for each of a list of ephys folders and
% picks the one that has the most data (nTrials * nSamples).
%
% "folderlist" is a cell array with a list of folders to test.
%
% "foldername" is the folder that had the most data.
% "foldersizes" is a vector with the same size as "folderlist" containing
%   the size (nTrials * nSamples) of each folder.
```

15.14 euUtil_readChannelClasses.m

```
% function [ channames, chanclasses ] = ...
%   euUtil_readChannelClasses( fname, namecolumn, classcolumn )
%
% This function reads a CSV file that assigns hand-annotated types to ephys
% channels. This file must have column labels in the first row.
%
```

```
% NOTE - Matlab will modify table column names; this function calls
% matlab.lang.makeValidName() to translate "namecolumn" and "classcolumn"
% using the same rules.
%
% "fname" is the name of the file to read from.
% "namecolumn" is the name of the column to read channel names from.
% "classcolumn" is the name of the column to read channel types from.
%
% "channames" is a vector or cell array containing channel numbers or names,
%   respectively.
% "chanclasses" is a cell array containing channel type labels.
```