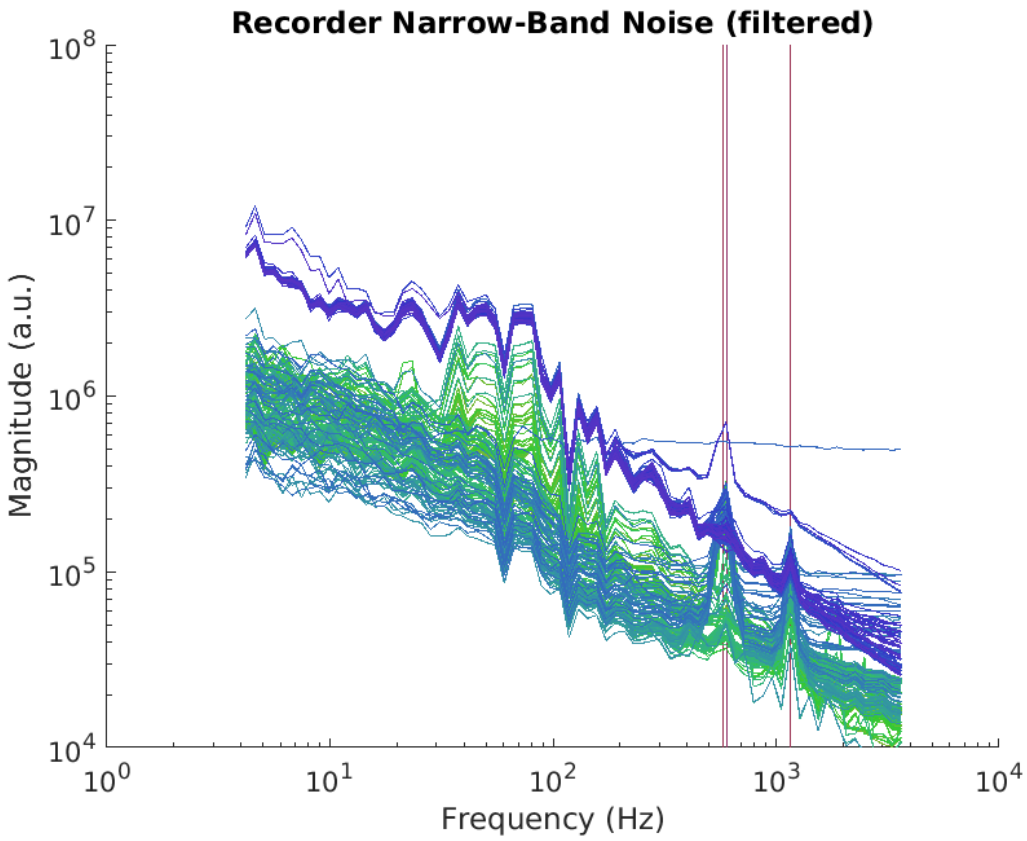
Field Trip Sample Script – Code Reference

Written by Christopher Thomas – March 4, 2022.



Contents

Chapter 1

Overview

This script is designed to show all of the steps involved in reading and pre-processing experiment data, integrating and aligning data from the Intan recorder, Intan stimulator, and from USE.

The component scripts are as follows, in the order in which they're called:

- `do_test` is the entrypoint. Edit one line in this file to select which dataset is processed (the datasets themselves are defined in a different file).
- `do_test_config` sets configuration parameters. This is the file that you edit if you want to change how the script processes data.
- `do_test_datasets` defines the datasets that might be processed. Add new datasets to represent data folders in your own environment.
- `do_test_get_metadata` reads the Field Trip headers for the ephys data and stores relevant metadata. This also fills in most of the fields of the configuration structures that will get passed to `ft_preprocessing`.
- `do_test_autoclassify_chans`, if enabled, reads a short segment of the recorder and stimulator ephys data and performs signal analysis on the resulting waveform data. This looks for several things: quantization noise, narrow-band spectral noise, signal drop-outs, and voltage excursions. This also performs a power law fit to the LFP spectrum to try to guess whether a channel contains neural data or noise. Finally, this computes correlation coefficients between channels to identify groups of channels with identical data (usually indicating floating channels). The intention of most of this is to identify "bad" channels in an automated way. The narrow-band noise information is used to identify beat frequencies that need to be added to the dataset's notch filter.
- `do_test_align` reads the Unity and SynchBox event data, reads the ephys TTL data, and builds lists of event codes and reward pulses. Time alignment tables are generated and used to align event data as well as frame and gaze data tables from Unity.
- `do_test_process_monolithic`, if enabled, tries to read the entire ephys dataset and perform signal processing on it. This may optionally be restricted to a subset of channels and to a shorter time window inside the dataset. This was mainly used during testing; keep it disabled unless you actually need it for something, since it takes up a lot of time and RAM.

- `do_test_define_trials` examines the event code sequence and defines trial boundaries and trigger points based on it. Trials are grouped into batches, so that trials can be read within a reasonable memory footprint.
- `do_test_process_trials` calls `ft_preprocessing` to read each batch of trials, performs signal processing on them, extracts aligned event, frame, and gaze data, and saves all of the associated data series for each batch of trials to disk.

Further processing can be performed by reading the trial data and trial metadata from disk, without having to touch the original data folders.

Chapter 2

ReadMe File

2.1 README.md

```
# Chris's Field Trip Examples

## Overview

This is a set of documentation and sample code intended to guide new users
through reading and processing our lab's data using Field Trip.

This is done using Field Trip's functions where possible, augmented with my
library code for experiment-specific tasks.

Type `make -C manual` to rebuild the manual for these scripts.

## 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.
* Assemble event code information, reward triggers, and stimulation triggers
from TTL data.
* Time-align signals from different machines (recorder and stimulator) 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 from the high-pass signal.
* Perform experiment-specific analysis.

Reading Data with Field Trip

* `ft_read_header` reads a dataset's configuration information. You can
pass it a custom reading function to read data types it doesn't know about.
For Intan or Open Ephys data that will be a LoopUtil function.
* `ft_read_event` reads a dataset's event lists (TTL data is often stored as
events). You can pass it a custom reading function to read data types it
doesn't know about it.

* ‘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.

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.

This is done in ‘do_test_define_trials.m‘. At some point this will get cleaned up and folded into library functions (along with much of the rest of this script).

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.

Minimum Working Example

For a very minimal example of Field Trip code, examine the ‘euTools_sanityCheckTree‘ function in ‘lib-exputils-tools‘.

* This does not do time alignment.
* This does not read USE data.
* This does not read TTL data (or any other events).
* This does not integrate data from multiple pieces of equipment.

What it *does* do is read individual saved datasets and perform signal processing on small segments of them, using Field Trip’s hooks.

A More Complex Example

The scripts in this directory perform all of the steps we’ll want to perform when pre-processing data from real experiments:

* Metadata for the recorder and stimulator datasets are read using

‘ft_read_header‘.
* Recorder and stimulator TTL data is read using ‘ft_read_event‘. This is assembled into event codes and reward/timer events.
* USE event data is read using ‘lib-exputils-use‘ functions. This includes a record of SynchBox and eye-tracker activity.
* USE, SynchBox, eye-tracker, and TTL data are time-aligned (using event codes if possible, other signals if not). Time alignment tables are built that can translate any piece of equipment’s timestamps into recorder time.
* Trials are defined based on appropraite event codes.
* ‘ft_preprocessing‘ is called to read the resulting trial ephys data. This happens in small batches of trials to avoid filling memory.
* Signal processing is performed on the trials:
    * Common-average referencing is performed on various pools of signals, if pools for this are defined.
    * Notch filtering is applied to remove power line noise and its harmonics, as well as any beat frequencies introduced by equipment sampling rates.
    * A "local field potential" signal is generated by low-pass-filtering and downsampling.
    * A "raw spikes" signal is generated by high-pass-filtering at the native sampling rate.
    * A "rectified spiking activity" series ("multi-unit activity" series) is generated by rectifying the "raw spikes" series (taking the absolute value), followed by low-pass filtering and downsampling.
* Eye-tracker data, TTL data, and event data that overlaps each trial is extracted.
* Processed analog and digital signals and events from batches of trials are saved to disk, along with trial metadata.

The intention is that after this preprocessing is done, the experiment analysis can be performed without having to touch the raw data again.

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.

Chapter 3

Top-Level and Metadata

3.1 do_test.m

```
% Field Trip sample script / test script.
% Written by Christopher Thomas.


%
% Paths.

% First step: Add the library root folders.
% These should be changed to match your system's locations, or you can set
% them as part of Matlab's global configuration.

addpath('lib-exp-utils-cjt');
addpath('lib-looputil');
addpath('lib-fieldtrip');
addpath('lib-openephys');
addpath('lib-npy-matlab');

% Second step: Call various functions to add library sub-folders.

addPathsExpUtilsCjt;
addPathsLoopUtil;

% Wrap this in "evalc" to avoid the annoying banner.
evalc('ft_defaults');



%
% Load configuration parameters.

do_test_config;
```

```
% This loads dataset information, but we still have to pick a dataset.
do_test_datasets;


% Pick the dataset we want to use.

%thisdataset = dataset_big_tungsten;
thisdataset = dataset_big_silicon_20211111;
%thisdataset = dataset_big_silicon_20211105;



%
% Initial setup.


% Set the number of channels we want in memory at any given time.
nlFT_setMemChans(memchans);


% Turn off FT notification messages. Otherwise they get spammy.
ft_notice('off');
ft_info('off');

% FIXME - Suppress warnings too.
% Among other things, when preprocessing the auto-generated configs have
% deprecated fields that generate lots of warnings.
ft_warning('off');

% FIXME - Suppress Matlab warnings. The NPy library complains about text data.
% Use "warning(warnstate)" to restore warnings to their default state.
warnstate = warning('off');



%
% Banner.

disp(sprintf('== Processing "%s".', thisdataset.title));



%
% Read the headers and select channels and timespans.

% FIXME - This reads and sets workspace variables directly.
% The Right Way to do this is to accept and return configuration structures.

do_test_get_metadata;
```

```
%
% Try autodetecting valid/invalid channels using windowed continuous data.

if want_auto_channel_types
  do_test_autoclassify_chans;
end



%
% Read and preview monolithic data (before segmenting).

% FIXME - This reads and sets workspace variables directly.
% The Right Way to do this is to accept configuration structures and to
% write large result structures to disk.

if want_process_monolithic
  do_test_process_monolithic;
end



%
% Load Unity data and TTL data, and perform alignment.

% FIXME - This reads and sets workspace variables directly.
% The Right Way to do this is to accept configuration structures and to
% write large result structures to disk.

if want_align
  do_test_align;
end



%
% Define data segments and process the segments.

% FIXME - This reads and sets workspace variables directly.
% The Right Way to do this is to accept configuration structures and to
% write large result structures to disk.

if want_define_trials
  do_test_define_trials;
end

% This will load trial definitions from disk if we didn't define them above.
```

```
if want_process_trials
  do_test_process_trials;
end



%
% Banner.

disp(sprintf('== Finished processing "%s".', thisdataset.title));


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

3.2 do_test_config.m

```
% Field Trip sample script / test script - Configuration.
% Written by Christopher Thomas.


%
% Behavior switches.
% These are what you'll usually want to edit.


% Trimming control for monolithic data processing.
% The idea is to trim big datasets to be small enough to fit in memory.
% 100 seconds is okay with 128ch, 1000 seconds is okay with 4ch.

want_crop_big = true;
crop_window_seconds = 100;
%crop_window_seconds = 1000;
want_detail_zoom = false;


% Ephys channel subset control.
% The idea is to read a small number of channels for debugging, for datasets
% that take a while to read.
% Alternatively we can force it to read all channels even when some of those
% are known to be floating.

want_chan_subset = true;
want_chan_include_unused = false;

% Set this to true to re-reference where metadata is available for that.
want_reref = true;
```

```
% Number of trials to batch-process at once.
% Memory footprint is on the order of 1 GB per trial for 128ch.
% Set this to "inf" to process all trials in a single batch.

%trials_per_batch = inf;
trials_per_batch = 10;

% Debugging switch - process only one batch (in the middle of the data).
want_one_batch = true;


% Turn on and off various processing steps.

% Try to automatically label ephys channels as good/bad/floating/etc.
% We identify dropouts and quantization in the time domain, and narrow-band
% noise in the frequency domain. We guess at good/bad by doing a power-law
% fit to the LFP in the frequency domain.
want_auto_channel_types = true;

% This debugging switch forces auto-typing to happen near the beginning of
% the data instead of in the middle.
want_auto_channel_early = false;

% Process continuous data before aligning and segmenting.
% This is mostly for debugging.
want_process_monolithic = false;

% Compare and align Unity and TTL data.
want_align = false;

% Build trial definitions.
want_define_trials = false;

% Process segmented data.
want_process_trials = false;

% Bring up the GUI data browser after processing (for debugging).
want_browser = true;

% Generate plots (for debugging).
want_plots = true;


% Optionally save data from various steps to disk.
% Optionally load previously-saved data instead of processing raw data.

want_save_data = true;

want_cache_autoclassify = false;
```

```
want_cache_monolithic = true;
want_cache_align_raw = true;
want_cache_align_done = true;
% Trial _definitions_ aren't cached; it's faster to rebuild them.
% Trial _data_ can be cached.
want_cache_epoched = false;



%
% Various magic values.
% You usually won't want to edit these.


% Output directories.

plotdir = 'plots';
datadir = 'output';


% Automatic channel classification.

% Analysis window duration in seconds for automatically testing for good/bad
% channels.
classify_window_seconds = 30;

% Anything with a range of this many bits or lower is flagged as quantized.
quantization_bits = 8;

% Anything with a smoothed rectified signal amplitude this far above or
% below the median is flagged as an artifact or drop-out, respectively, for
% classification purposes.
artifact_rect_threshold = 5;
dropout_rect_threshold = 0.3;

% Approximate duration of artifacts and dropouts, in seconds.
% This should be at least 5x longer than spike durations.
% Anything within a factor of 2-3 of this will get recognized, at minimum.
artifact_dropout_time = 0.02;

% Channels with more than this fraction of artifacts or dropouts are flagged
% as bad.
artifact_bad_frac = 0.01;
dropout_bad_frac = 0.01;

% Tuning parameters for looking for narrow-band noise.
% See nlProc_findSpectrumPeaks() for discussion.
if true
  % Relatively wide search bins, relatively insensitive.
  noisepeakwidth = 0.1;
```

```
  noisebackgroundwidth = 2.0;
  noisepeakthreshold = 3.0;
elseif true
  % Relatively narrow search bins. Somewhat more sensitive.
  noisepeakwidth = 0.03;
  noisebackgroundwidth = 1.5;
  noisepeakthreshold = 4.0;
else
  % Very narrow search bins. This is too sensitive (lots of harmonics).
  noisepeakwidth = 0.01;
  noisebackgroundwidth = 1.5;
  noisepeakthreshold = 4.0;
end

% Tuning parameters for looking at the LFP spectrum shape.
% See nlProc_examineLFPSpectrum() for discussion.
lfpspectrange = [ 4 200 ];
lfpbinwidth = 0.03;

% Tuning parameters for looking at correlations between channels.
% See nlProc_findCorrelatedChannels() for discussion.
% Actual sets of floating channels tend to have R values of 0.99 or so.
correl_abs_thresh = 0.9;
correl_rel_thresh = 4.0;


% Analog signal filtering.

% Valid filter types are 'fir', 'dft', 'cosine', 'brickwall', and 'thilo'.

% Filter type to use for long data.
% The brick-wall filter is the only one that works on long-duration signals
% with acceptable time and memory costs.
filter_type_long = 'brickwall';

% Filter type to use for trial data.
% The "dft" and "brickwall" filters are the only ones that work while using
% a reasonable amount of time and memory.
% Alarmingly, the brick-wall filter has fewer artifacts at high frequencies
% than the "dft" band-stop filter.
%filter_type_short = 'dft';
filter_type_short = 'brickwall';

% The power frequency filter filters the fundamental mode and some of the
% harmonics of the power line frequency. Mode count should be 2-3 typically.
power_freq = 60.0;
power_filter_modes = 3;

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

```
% The DC component should have been removed in an earlier step.
lfp_corner = 300;
lfp_rate = 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_corner = 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_corners = [ 1000 3000 ];
rect_lowpass = 500;
rect_rate = 2000;


% Event code processing.

evcodebytes = 2;
evcodeendian = 'big';


% Time alignment.

% For coarse windows, one candidate is picked within the window and matched
% against other candidates. The window is walked forwards by its radius.
% Constant-delay alignment is performed using the first coarse window value.
aligncoarsewindows = [ 100.0 ];

% For medium windows, each event is considered as the center of a window and
% is matched against other candidates in the window.
alignmedwindows = [ 1.0 ];

% For fine alignment, each event is considered as the center of a window,
% all events in the window are considered to match their nearest candidates,
% and a fine-tuning offset is calculated for that window position.
alignfinewindow = 0.1;

% Outlier time-deltas will substantially skew time estimates around them.
alignoutliersigma = 4.0;

% This should either be 'quiet' or 'normal'. 'verbose' is for debugging.
alignverbosity = 'normal';


% Epoch segmentation.

% We're always using 'TrlStart' and 'TrlEnd' to identify trials.
% Those aren't stored here.
```

```
% To identify the span to _save_, we use 'trialstartcode' and 'trialendcode'.
% The trial starts at the _latest_ start code seen and ends at the _earliest_
% end code seen.
trialstartcodes = { 'TrlStart' };
trialendcodes = { 'TrlEnd' };

% We want to add a halo to give filters time to stabilize. This should be
% at least 3 times the longest corner period and preferably 10 times.
% The problem is that artifacts may occur at certain points in the trial
% (e.g. when rewards are given), and those may cause large filter artifacts.
trialstartpadsecs = 3.0;
trialendpadsecs = 3.0;

% We want to examine multiple time alignment cases (e.g. cue, choice, and
% feedback alignment).
% For each case, there may be multiple codes to align to (e.g. "correct" vs
% "incorrect"). We align to the _first_ such code seen, for a given case.
% Remember to use pairs of braces so that we get one structure instead of
% an array.
trialaligncodes = struct( 'cue', {{ 'StimOn' }} );


% Table columns to convert to FT waveform data.

% Cooked gaze coordinates from FrameData.
% NOTE - There's an 'EyePositionZ', but it doesn't contain useful data.
% NOTE - In the sample dataset, absolute position is -1k..+2k, and relative
% position is -1..+2, roughly.
frame_gaze_cols = { 'EyePositionX', 'EyePositionY', ...
  'RelativeEyePositionX', 'RelativeEyePositionY' };

% Sampling rate for cooked gaze data.
gaze_rate = lfp_rate;


% File I/O.

% The number of channels to load into memory at one time, when loading.
% This takes up at least 1 GB per channel-hour.
memchans = 4;

% Patterns that various channel names match.
% See "ft_channelselection" for special names. Use "*" as a wildcard.
name_patterns_ephys = { 'Amp*', 'CH*' };
name_patterns_digital = { 'Din*', 'Dout*', 'DigBits*', 'DigWords*' };
name_patterns_stim_current = { 'Stim*' };
name_patterns_stim_flags = { 'Flags*' };

% Which types of data to read.
% We usually want all data; this lets us turn off elements for testing.
```

```
want_data_ephys = true;
want_data_ttl = true;
want_data_stim = true;
want_data_events = true;



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

3.3 do_test_datasets.m

```
% Field Trip sample script / test script - Dataset definitions.
% Written by Christopher Thomas.

% Dataset definitions are structures containing various metadata about each
% dataset we might want to process.


%
% Short datasets for format testing.


% Intan monolithic format.

srcdir = [ 'datasets-intan', filesep, 'MonolithicIntan_format' ];
dataset_intan_monolithic = struct( ...
  'title', 'Intan Monolithic', 'label', 'intanmono', ...
  'recfile', [ srcdir, filesep, 'record_211206_171502.rhd' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211206_171502.rhs' ], ...
  'use_looputil', true );


% Intan one-file-per-type format.

srcdir = [ 'datasets-intan', filesep, 'OneFilePerTypeOfChannel_format' ];
dataset_intan_pertype = struct( ...
  'title', 'Intan Per-Type', 'label', 'intanpertype', ...
  'recfile', [ srcdir, filesep, 'record_211206_172518' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211206_172519' ], ...
  'use_looputil', true );


% Intan one-file-per-channel format.

srcdir = [ 'datasets-intan', filesep, 'OneFilePerChannel_format' ];
dataset_intan_perchan = struct( ...
```

```
  'title', 'Intan Per-Channel', 'label', 'intanperchan', ...
  'recfile', [ srcdir, filesep, 'record_211206_172734' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211206_172734' ], ...
  'use_looputil', true );

% Add "zoom" case.
if want_detail_zoom
  dataset_intan_perchan.timerange = [ 4.0 6.0 ];
end


% Intan monolithic format converted to Plexon .NEX or .NEX5.

srcdir = [ 'datasets-intan', filesep, 'MonolithicIntan_Plexon' ];
dataset_intan_plexon = struct( ...
  'title', 'Intan (converted to NEX)', 'label', 'intanplexon', ...
  'recfile', [ srcdir, filesep, 'record_211206_171502.nex' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211206_171502.nex' ], ...
  'use_looputil', false );

dataset_intan_plexon_nex5 = dataset_intan_plexon;
dataset_intan_plexon_nex5.recfile = ...
  [ srcdir, filesep, 'record_211206_171502.nex5' ];


% Open Ephys monolithic format.

srcdir = [ 'datasets-openephys', filesep, ...
  'OEBinary_IntanStimOneFilePerChannel_format' ];

% NOTE - Pointing to directory, not "structure.oebin".

dataset_openephys_monolithic = struct( ...
  'title', 'Open Ephys Monolithic', 'label', 'openmono', ...
  'recfile', [ srcdir, filesep, ...
    '2021-12-17_14-47-00', filesep, 'Record Node 101', filesep, ...
    'experiment1', filesep, 'recording1' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211217_144659' ], ...
  'use_looputil', true );

% Add "zoom" case.
if want_detail_zoom
  dataset_openephys_monolithic.timerange = [ 12.0 14.0 ];
end


% Open Ephys one-file-per-channel format.

srcdir = [ 'datasets-openephys', filesep, ...
  'OEOpenEphys_IntanStimOneFilePerChannel_format' ];
```

```
dataset_openephys_perchan = struct( ...
  'title', 'Open Ephys Per-Channel', 'label', 'openperchan', ...
  'recfile', [ srcdir, filesep, ...
    '2021-12-17_14-47-00', filesep, 'Record Node 101' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211217_150043' ], ...
  'use_looputil', true );


% Open Ephys monolithic converted to Plexon .PLX format.

% FIXME - Open Ephys Plexon NYI.



%
% Large datasets.


% 2021 November 12 tungsten recording (has stimulation).

srcdir = [ 'datasets-big', filesep, '20211112-frey-tungsten' ];
dataset_big_tungsten = struct( ...
  'title', '2021 Nov 12 Frey Tungsten', 'label', 'freytungsten', ...
  'recfile', [ srcdir, filesep, 'record_211112_112922' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211112_112924' ], ...
  'unityfile', [ srcdir, filesep, 'Session4__12_11_2021__11_29_57', ...
    filesep, 'RuntimeData' ], ...
  'synchbox', struct( 'reccodebits', 'Din_*', 'recshift', 8, ...
     'stimrwdB', 'Din_002' ), ...
  'use_looputil', true );

% FIXME - Manually adding a filter for beat frequency noise.
dataset_big_tungsten.extra_notches = [ 561.6 ];

% These are the channels that were actually used.
chansrec = { 'AmpA_045', 'AmpA_047' };
chansstim = { 'AmpC_011' };

if ~want_chan_include_unused
  dataset_big_tungsten.channels_rec = chansrec;
  dataset_big_tungsten.channels_stim = chansstim;

  % We can use common-average referencing on the recording channels, but
  % we only have a single stimulation channel.
  dataset_big_tungsten.commonrefs_rec = { chansrec };
end

% Crop the dataset if desired.

if want_crop_big
```

```
  % The full trace is about 5800 seconds long (1.6h).
%  crop_start = 1000.0;
  crop_start = 2000.0;
%  crop_start = 3000.0;
  dataset_big_tungsten.timerange = ...
    [ crop_start (crop_start + crop_window_seconds) ];
end

% Add "zoom" cases.

if want_detail_zoom
% FIXME - Detail zoom for Frey tungsten NYI.
end


% 2021 November 05 silicon recording (NOTE - has dropouts).

srcdir = [ 'datasets-big', filesep, '20211105-frey-silicon' ];

% NOTE - Pointing to directory, not "structure.oebin".

dataset_big_silicon_20211105 = struct( ...
  'title', '2021 Nov 05 Frey Silicon', 'label', 'freysilicon', ...
  'recfile', [ srcdir, filesep, ...
    '2021-11-05_11-55-08', filesep, 'Record Node 101', filesep, ...
    'experiment1', filesep, 'recording1' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211105_115503' ], ...
  'unityfile', [ srcdir, filesep, 'Session17__05_11_2021__11_55_39', ...
    filesep, 'RuntimeData' ], ...
  'synchbox', struct( 'reccodes', 'DigWordsA_000', 'recshift', 8, ...
     'stimrwdB', 'Din_002' ), ...
  'use_looputil', true );

% Crop the dataset if desired.

if want_crop_big
  % The full trace is about 4300 seconds long (1.2h).
%  crop_start = 1000.0;
  crop_start = 2000.0;
%  crop_start = 3000.0;
  dataset_big_silicon_20211105.timerange = ...
    [ crop_start (crop_start + crop_window_seconds) ];
end

% Add "zoom" cases.

if want_detail_zoom
% FIXME - Detail zoom for Frey silicon NYI.
end
```

```
% Add "only a few channels" case.
% FIXME - Need to prune floating channels even without this!

if want_chan_subset
  % Filter analog channels on the recorder.
  dataset_big_silicon_20211105.channels_rec = ...
    { 'CH_001', 'CH_030', 'CH_070', 'CH_110' };
end


% 2021 November 11 silicon recording (NOTE - has dropouts).

srcdir = [ 'datasets-big', filesep, '20211111-frey-silicon' ];

% NOTE - Pointing to directory, not "structure.oebin".

dataset_big_silicon_20211111 = struct( ...
  'title', '2021 Nov 11 Frey Silicon', 'label', 'freysiliconbad', ...
  'recfile', [ srcdir, filesep, ...
    '2021-11-11_12-08-33', filesep, 'Record Node 101', filesep, ...
    'experiment2', filesep, 'recording1' ], ...
  'stimfile', [ srcdir, filesep, 'stim_211111_121220' ], ...
  'unityfile', [ srcdir, filesep, 'Session3__11_11_2021__12_12_49', ...
    filesep, 'RuntimeData' ], ...
  'synchbox', struct( 'reccodes', 'DigWordsA_000', 'recshift', 8, ...
     'stimrwdB', 'Din_002' ), ...
  'use_looputil', true );

% Crop the dataset if desired.

if want_crop_big
  % The full trace is about 4300 seconds long (1.2h).
%  crop_start = 1000.0;
  crop_start = 2000.0;
%  crop_start = 3000.0;
  dataset_big_silicon_20211111.timerange = ...
    [ crop_start (crop_start + crop_window_seconds) ];
end

% Add "zoom" cases.

if want_detail_zoom
% FIXME - Detail zoom for Frey silicon NYI.
end

% Add "only a few channels" case.
% FIXME - Need to prune floating channels even without this!

if want_chan_subset
  % Filter analog channels on the recorder.
```

```
  dataset_big_silicon_20211111.channels_rec = ...
    { 'CH_001', 'CH_030', 'CH_070', 'CH_110' };
end



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

3.4 do_test_get_metadata.m

```
% Field Trip sample script / test script - Headers and metadata.
% Written by Christopher Thomas.

% This reads headers and selects the timespans and channels we want.
% FIXME - Doing this by reading and setting workspace variables directly.
%
% Variables that get set:
%   rechdr
%   stimhdr
%   rec_channels_ephys
%   rec_channels_digital
%   stim_channels_ephys
%   stim_channels_digital
%   stim_channels_current
%   stim_channels_flags
%   preproc_config_rec
%   preproc_config_stim
%   preproc_config_rec_span_default
%   preproc_config_stim_span_default
%   preproc_config_rec_span_autotype
%   preproc_config_stim_span_autotype


%
% Read the headers.


disp('-- Reading headers.');

% NOTE - Field Trip will throw an exception if this fails. Wrap this to
% catch exceptions.

try

  % Read the headers. This gives us the channel lists.

  if thisdataset.use_looputil
```

```
    rechdr = ft_read_header( thisdataset.recfile, ...
      'headerformat', 'nlFT_readHeader' );
    stimhdr = ft_read_header( thisdataset.stimfile, ...
      'headerformat', 'nlFT_readHeader' );
  else
    rechdr = ft_read_header( thisdataset.recfile );
    stimhdr = ft_read_header( thisdataset.stimfile );
  end

catch errordetails
  disp(sprintf( ...
    '###  Exception thrown while reading "%s".', thisdataset.title));
  disp(sprintf('Message: "%s"', errordetails.message));

  % Abort the script and send the user back to the Matlab prompt.
  error('Couldn''t read headers; bailing out.');
end



%
% Select channels.


% FIXME - Not filtering digial I/O channels.
% Blithely assuming these channels fit in RAM.

% FIXME - Not filtering stimulation current or flag data for now.
% This only fits in RAM if the user saved the channels used rather than
% saving all channels.
% We really should filter this the same way we filter analog data from
% the stimulation channels.


% Analog ephys channels.

rec_channels_ephys = ...
  ft_channelselection( name_patterns_ephys, rechdr.label, {} );
if isfield( thisdataset, 'channels_rec' )
  rec_channels_ephys = ...
    ft_channelselection( thisdataset.channels_rec, rechdr.label, {} );
end

stim_channels_ephys = ...
  ft_channelselection( name_patterns_ephys, stimhdr.label, {} );
if isfield( thisdataset, 'channels_stim' )
  stim_channels_ephys = ...
    ft_channelselection( thisdataset.channels_stim, stimhdr.label, {} );
end
```

```
% Digital I/O channels.

rec_channels_digital = ...
  ft_channelselection( name_patterns_digital, rechdr.label, {} );

stim_channels_digital = ...
  ft_channelselection( name_patterns_digital, stimhdr.label, {} );


% Stimulation current and other stimulation metadata.

stim_channels_current = ...
  ft_channelselection( name_patterns_stim_current, stimhdr.label, {} );
stim_channels_flags = ...
  ft_channelselection( name_patterns_stim_flags, stimhdr.label, {} );


% Suppress data types we don't want.

if ~want_data_ephys
  rec_channels_ephys = {};
  stim_channels_ephys = {};
end

if ~want_data_ttl
  rec_channels_digital = {};
  stim_channels_digital = {};
end

if ~want_data_stim
  stim_channels_current = {};
  stim_channels_flags = {};
end



% NOTE - Passing an empty channel list to ft_preprocessing() results in all
% channels being read. Passing a bogus placeholder name to guarantee no
% channels being read causes ft_preprocessing() to throw an exception (it
% does this if it can't find any data).

% Long story short, before calling ft_preprocessing() explicitly check for
% empty channel lists.



%
% Select timespans and build preprocessing configuration structures.
```

```
% NOTE - We'll be reading several different types of signal separately.
% For each call to ft_preprocessing, we have to build a configuration
% structure specifying what we want to read.
% The only part that changes for these calls is "channel" (the channel
% name list), which is different for different signal types.


% Basic configuration.

preproc_config_rec = struct( ...
  'datafile', thisdataset.recfile, 'headerfile', thisdataset.recfile );
preproc_config_stim = struct( ...
  'datafile', thisdataset.stimfile, 'headerfile', thisdataset.stimfile );

if thisdataset.use_looputil
  % NOTE - Promoting everything to double-precision floating-point.
  % It might be better to keep TTL signals in native format.

  preproc_config_rec.headerformat = 'nlFT_readHeader';
  preproc_config_rec.dataformat = 'nlFT_readDataDouble';

  preproc_config_stim.headerformat = 'nlFT_readHeader';
  preproc_config_stim.dataformat = 'nlFT_readDataDouble';
end


% Build time ranges.
% These get stored as "preproc_config_XX.trl".
% Defining a single trial gets us windowed continuous data.

% Monolithic data time range.

% FIXME - Windowing is done before time alignment, so we'd better be sure
% that the time difference between the recorder and stimulator is much
% shorter than the window size.

preproc_config_rec_span_default = [ 1 rechdr.nSamples 0 ];
preproc_config_stim_span_default = [ 1 stimhdr.nSamples 0 ];

if isfield(thisdataset, 'timerange')

  firstsamp = round( min(thisdataset.timerange) * rechdr.Fs );
  firstsamp = min(firstsamp, rechdr.nSamples);
  firstsamp = max(firstsamp, 1);

  lastsamp = round ( max(thisdataset.timerange) * rechdr.Fs );
  lastsamp = min(lastsamp, rechdr.nSamples);
  lastsamp = max(lastsamp, 1);
```

```
  preproc_config_rec_span_default = [ firstsamp lastsamp 0 ];


  firstsamp = round( min(thisdataset.timerange) * stimhdr.Fs );
  firstsamp = min(firstsamp, stimhdr.nSamples);
  firstsamp = max(firstsamp, 1);

  lastsamp = round ( max(thisdataset.timerange) * stimhdr.Fs );
  lastsamp = min(lastsamp, stimhdr.nSamples);
  lastsamp = max(lastsamp, 1);

  preproc_config_stim_span_default = [ firstsamp lastsamp 0 ];

end

% Auto-configuration time range.
% Put this in the middle of the dataset.

autosamp_first_frac = 0.5;
if want_auto_channel_early
  % FIXME - Force this to the start, for testing before stimulation.
  autosamp_first_frac = 0.05;
end

firstsamp = round(autosamp_first_frac * rechdr.nSamples);
lastsamp = firstsamp + round( classify_window_seconds * rechdr.Fs );
lastsamp = min( lastsamp, rechdr.nSamples );

preproc_config_rec_span_autotype = [ firstsamp lastsamp 0 ];

firstsamp = round(autosamp_first_frac * stimhdr.nSamples);
lastsamp = firstsamp + round( classify_window_seconds * stimhdr.Fs );
lastsamp = min( lastsamp, stimhdr.nSamples );

preproc_config_stim_span_autotype = [ firstsamp lastsamp 0 ];



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

Chapter 4

Time Alignment

4.1 do_test_align.m

```
% Field Trip sample script / test script - Time alignment.
% Written by Christopher Thomas.

% This reads Unity event data and TTL data and time-aligns them.
% FIXME - Doing this by reading and setting workspace variables directly.
%
% Variables that get set:
%   have_unity
%   evcodedefs
%   boxsynchA
%   boxsynchB
%   boxrwdA
%   boxrwdB
%   boxcodes
%   boxcodes_raw
%   gamerwdA
%   gamerwdB
%   gamecodes
%   gamecodes_raw
%   have_recevents_dig
%   have_stimevents_dig
%   recevents_dig
%   stimevents_dig
%   have_recrwdA
%   have_recrwdB
%   have_recsynchA
%   have_recsynchB
%   have_reccodes
%   recrwdA
%   recrwdB
%   recsynchA
```

```
%   recsynchB
%   reccodes
%   reccodes_raw
%   have_stimrwdA
%   have_stimrwdB
%   have_stimsynchA
%   have_stimsynchB
%   have_stimcodes
%   stimrwdA
%   stimrwdB
%   stimsynchA
%   stimsynchB
%   stimcodes
%   gamegaze_raw
%   gameframedata_raw
%   times_recorder_synchbox
%   times_recorder_game
%   times_recorder_stimulator
%   times_game_eyetracker
%   times_recorder_eyetracker
%   unityreftime


%
% == Raw data.

% We're either loading this from ephys/unity or loading a cached version.

fname_rawttl = [ datadir filesep 'ttl_raw.mat' ];
fname_rawevents = [ datadir filesep 'events_raw.mat' ];
fname_rawgaze = [ datadir filesep 'gaze_raw.mat' ];
fname_rawframe = [ datadir filesep 'frame_raw.mat' ];


if want_cache_align_raw ...
  && isfile(fname_rawttl) && isfile(fname_rawevents) ...
  && isfile(fname_rawgaze) && isfile(fname_rawframe)

  %
  % Load raw data from disk.

  disp('-- Loading raw TTL events.');

  load(fname_rawttl);

  disp('-- Unpacking raw TTL events.');

  recevents_dig = struct([]);
  if have_recevents_dig
    recevents_dig = ...
```

```
      nlFT_uncompressFTEvents( recevents_dig_tab, rechdr.label );
  end
  stimevents_dig = struct([]);
  if have_stimevents_dig
    stimevents_dig = ...
      nlFT_uncompressFTEvents( stimevents_dig_tab, stimhdr.label );
  end

  disp('-- Loading raw Unity events.');

  load(fname_rawevents);

  disp('-- Loading raw Unity gaze data.');

  load(fname_rawgaze);

  disp('-- Loading raw Unity frame data.');

  load(fname_rawframe);

  disp('-- Finished loading.');

else

  %
  % Load raw data from ephys and unity folders.


  %
  % Read TTL and gaze data from Unity.

  % FIXME - These should use Field Trip wrappers!

  % FIXME - We have to keep the raw event codes as well.
  % The alignment routines misbehave trying to line up the SynchBox with
  % the ephys machines based on cooked codes, due to a large number of
  % dropped bytes (the synchbox-to-unity reply link is saturated).

  have_unity = false;
  if isfield( thisdataset, 'unityfile' )

    have_unity = true;

    disp('-- Reading Unity event data.');

    [ sentdata recvdata ] = euUSE_readRawSerialData(thisdataset.unityfile);
    [ boxsynchA boxsynchB boxrwdA boxrwdB boxcodes_raw ] = ...
      euUSE_parseSerialRecvData(recvdata, 'dupbyte');
    [ gamerwdA gamerwdB gamecodes_raw ] = ...
      euUSE_parseSerialSentData(sentdata, 'dupbyte');
```

```
  evcodedefs = euUSE_readEventCodeDefs(thisdataset.unityfile);

  % Translate raw code bytes into cooked codes.
  [ boxcodes origlocations ] = euUSE_reassembleEventCodes( ...
    boxcodes_raw, evcodedefs, evcodebytes, evcodeendian, 'codeValue' );
  [ gamecodes origlocations ] = euUSE_reassembleEventCodes( ...
    gamecodes_raw, evcodedefs, evcodebytes, evcodeendian, 'codeValue' );

  disp('-- Finished reading Unity event data.');


  disp('-- Reading Unity gaze data.');

  % FIXME - This should be turned into waveform data. For now, keep it
  % as a not-quite-uniformly-sampled data table.
  % NOTE - This uses its own timestamps and time quantum, not unity's.
  % The loading function gives us a 'time_seconds' column.
  gamegaze_raw = euUSE_readRawGazeData(thisdataset.unityfile);

  disp('-- Finished reading Unity gaze data.');


  disp('-- Reading Unity frame data.');

  % NOTE - This has eye-tracker and unity timestamps.
  % The loading function adds "SystemTimeSeconds" and
  % "EyetrackerTimeSeconds" columns with timestamps in seconds.
  gameframedata_raw = euUSE_readRawFrameData(thisdataset.unityfile);

  disp('-- Finished reading Unity frame data.');

end


%
% Read TTL data from ephys recorders.


% First, get the ephys TTL events themselves if we don't already have them.

% NOTE - Field Trip will throw an exception if this fails. Wrap this to
% catch exceptions.

try

  disp('-- Reading ephys digital events.');

  if exist('have_recevents_dig', 'var') && have_recevents_dig
    disp('.. Already have events from the recorder.');
```

```
  else
    disp('.. Reading from recorder.');

    recevents_dig = ft_read_event( thisdataset.recfile, ...
      'headerformat', 'nlFT_readHeader', 'eventformat', 'nlFT_readEvents' );

    if isempty(recevents_dig)
      disp('.. No recorder events found. Trying again using waveforms.');
      recevents_dig = ft_read_event( thisdataset.recfile, ...
        'headerformat', 'nlFT_readHeader', ...
        'eventformat', 'nlFT_readEventsContinuous' );
    end

    have_recevents_dig = true;
  end

  if exist('have_stimevents_dig', 'var') && have_stimevents_dig
    disp('.. Already have events from the stimulator.');
  else
    disp('.. Reading from stimulator.');

    stimevents_dig = ft_read_event( thisdataset.stimfile, ...
      'headerformat', 'nlFT_readHeader', 'eventformat', 'nlFT_readEvents' );

    if isempty(stimevents_dig)
      disp('.. No stimulator events found. Trying again using waveforms.');
      stimevents_dig = ft_read_event( thisdataset.stimfile, ...
        'headerformat', 'nlFT_readHeader', ...
        'eventformat', 'nlFT_readEventsContinuous' );
    end

    have_stimevents_dig = true;
  end

  disp('-- Finished reading ephys digital events.');

catch errordetails
  disp(sprintf( ...
    '###  Exception thrown while reading "%s".', thisdataset.title));
  disp(sprintf('Message: "%s"', errordetails.message));

  % Abort the script and send the user back to the Matlab prompt.
  error('Couldn''t read events; bailing out.');
end


% Finished loading raw data.


%
```

```
% Isolate the signals we're interested in.


disp('-- Looking for SynchBox signals in ephys data.');

synchboxsignals = struct();
if isfield(thisdataset, 'synchbox')
  synchboxsignals = thisdataset.synchbox;
end

% FIXME - This only works for LoopUtil events.
% Those events store the channel name as the event "type" field.


% Recorder single-bit signals.

[ recrwdA have_recrwdA ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'recrwdA', rechdr.label, recevents_dig );
[ recrwdB have_recrwdB ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'recrwdB', rechdr.label, recevents_dig );
[ recsynchA have_recsynchA ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'recsynchA', rechdr.label, recevents_dig );
[ recsynchB have_recsynchB ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'recsynchB', rechdr.label, recevents_dig );

% Keep only rising-edge events.
if have_recrwdA
  recrwdA = recrwdA(recrwdA.value > 0,:);
end
if have_recrwdB
  recrwdB = recrwdB(recrwdB.value > 0,:);
end
if have_recsynchA
  recsynchA = recsynchA(recsynchA.value > 0,:);
end
if have_recsynchB
  recsynchB = recsynchB(recsynchB.value > 0,:);
end


% Stimulator single-bit signals.

[ stimrwdA have_stimrwdA ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'stimrwdA', stimhdr.label, stimevents_dig );
[ stimrwdB have_stimrwdB ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'stimrwdB', stimhdr.label, stimevents_dig );
[ stimsynchA have_stimsynchA ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'stimsynchA', stimhdr.label, stimevents_dig );
[ stimsynchB have_stimsynchB ] = euFT_getSingleBitEvent( ...
  synchboxsignals, 'stimsynchB', stimhdr.label, stimevents_dig );
```

```
% Keep only rising-edge events.
if have_stimrwdA
  stimrwdA = stimrwdA(stimrwdA.value > 0,:);
end
if have_stimrwdB
  stimrwdB = stimrwdB(stimrwdB.value > 0,:);
end
if have_stimsynchA
  stimsynchA = stimsynchA(stimsynchA.value > 0,:);
end
if have_stimsynchB
  stimsynchB = stimsynchB(stimsynchB.value > 0,:);
end


% Event codes from both devices.

% FIXME - We have to keep the raw event codes as well.
% The alignment routines misbehave trying to line up the SynchBox with
% the ephys machines based on cooked codes, due to a large number of
% dropped bytes (the synchbox-to-unity reply link is saturated).

% FIXME - The only situation where we have to assemble from bits is with
% the Intan machine, and channel numbering starts at 1 in that situation.
firstbit = 1;

[ reccodes_raw have_reccodes ] = euFT_getCodeWordEvent( ...
  synchboxsignals, 'reccodes', 'reccodebits', firstbit, 'recshift', ...
  rechdr.label, recevents_dig );

[ stimcodes_raw have_stimcodes ] = euFT_getCodeWordEvent( ...
  synchboxsignals, 'stimcodes', 'stimcodebits', firstbit, 'stimshift', ...
  stimhdr.label, stimevents_dig );


% Squash event code values of zero; that's the idle state.
% Merge codes that repeat the same timestamp or that are one sample apart.
% These use the FT event column labels ("sample", "value", "type",
% "offset", "duration").
reccodes_raw = euUSE_cleanEventsTabular( reccodes_raw, 'value', 'sample' );
stimcodes_raw = euUSE_cleanEventsTabular( stimcodes_raw, 'value', 'sample' );


% Translate raw code bytes into cooked codes.
if ~have_unity
  disp( ...
    '###  Can''t reassemble event codes without USE''s code definitions!');
else
  [ reccodes origlocations ] = euUSE_reassembleEventCodes( ...
```

```
    reccodes_raw, evcodedefs, evcodebytes, evcodeendian, 'value' );
  [ stimcodes origlocations ] = euUSE_reassembleEventCodes( ...
    stimcodes_raw, evcodedefs, evcodebytes, evcodeendian, 'value' );
end

% Done.

disp('-- Finished looking for SynchBox signals in ephys data.');


%
% Save the results to disk, if requested.

if want_save_data
  if isfile(fname_rawttl)    ; delete(fname_rawttl)    ; end
  if isfile(fname_rawevents) ; delete(fname_rawevents) ; end
  if isfile(fname_rawgaze)   ; delete(fname_rawgaze)   ; end
  if isfile(fname_rawframe ) ; delete(fname_rawframe)  ; end

  % NOTE - Saving TTL events in packed tabular form, as that's far smaller
  % than structure array form.

  disp('-- Compressing raw TTL events.');

  recevents_dig_tab = table();
  if have_recevents_dig
    [ recevents_dig_tab scratchlut ] = ...
      nlFT_compressFTEvents( recevents_dig, rechdr.label );
  end
  stimevents_dig_tab = table();
  if have_stimevents_dig
    [ stimevents_dig_tab scratchlut ] = ...
      nlFT_compressFTEvents( stimevents_dig, stimhdr.label );
  end

  disp('-- Saving raw TTL event data.');

  save( fname_rawttl, ...
    'have_recevents_dig', 'recevents_dig_tab', ...
    'have_stimevents_dig', 'stimevents_dig_tab', ...
    '-v7.3' );

  disp('-- Saving raw Unity event data.');

  save( fname_rawevents, ...
    'have_unity', 'evcodedefs', ...
    'boxsynchA', 'boxsynchB', 'boxrwdA', 'boxrwdB', ...
    'boxcodes', 'boxcodes_raw', ...
    'gamerwdA', 'gamerwdB', 'gamecodes', 'gamecodes_raw', ...
    'have_recrwdA', 'recrwdA', 'have_recrwdB', 'recrwdB', ...
```

```
      'have_recsynchA', 'recsynchA', 'have_recsynchB', 'recsynchB', ...
      'have_reccodes', 'reccodes', 'reccodes_raw', ...
      'have_stimrwdA', 'stimrwdA', 'have_stimrwdB', 'stimrwdB', ...
      'have_stimsynchA', 'stimsynchA', 'have_stimsynchB', 'stimsynchB', ...
      'have_stimcodes', 'stimcodes', 'stimcodes_raw', ...
      '-v7.3' );

    disp('-- Saving raw Unity gaze data.');

    save( fname_rawgaze, 'gamegaze_raw', '-v7.3' );

    disp('-- Saving raw Unity frame data.');

    save( fname_rawframe, 'gameframedata_raw', '-v7.3' );

    disp('-- Finished saving.');
  end

end



% FIXME - Diagnostics.

disp(sprintf( ...
'.. From SynchBox:  %d rwdA  %d rwdB  %d synchA  %d synchB  %d codes', ...
  height(boxrwdA), height(boxrwdB), ...
  height(boxsynchA), height(boxsynchB), height(boxcodes) ));
disp(sprintf( ...
  '.. From USE:  %d rwdA  %d rwdB  %d codes', ...
  height(gamerwdA), height(gamerwdB), height(gamecodes) ));

disp(sprintf( ...
  '.. From recorder:  %d rwdA  %d rwdB  %d synchA  %d synchB  %d codes', ...
  height(recrwdA), height(recrwdB), ...
  height(recsynchA), height(recsynchB), height(reccodes) ));
disp(sprintf( ...
  '.. From stimulator:  %d rwdA  %d rwdB  %d synchA  %d synchB  %d codes', ...
  height(stimrwdA), height(stimrwdB), ...
  height(stimsynchA), height(stimsynchB), height(stimcodes) ));



%
% == Time alignment.

% We're propagating recorder timestamps to all other devices and using these
% as our official set of timestamps.
```

```
%
% Check for cached data and return immediately if we find it.

fname_cookedevents = [ datadir filesep 'events_aligned.mat' ];
fname_cookedgaze = [ datadir filesep 'gaze_aligned.mat' ];
fname_cookedframe = [ datadir filesep 'frame_aligned.mat' ];

if want_cache_align_done && isfile(fname_cookedevents) ...
  && isfile(fname_cookedgaze) && isfile(fname_cookedframe)

  %
  % Load aligned data from disk.

  disp('-- Loading time-aligned Unity events and alignment tables.');

  load(fname_cookedevents);

  disp('-- Loading time-aligned gaze data.');

  load(fname_cookedgaze);

  disp('-- Loading time-aligned Unity frame data.');

  load(fname_cookedframe);

  disp('-- Finished loading.');

  % We've loaded cached results. Bail out of this portion of the script.
  return;

end


%
% If we don't have the information we need, bail out.

% We need USE and the recorder. Stimulator is optional.
% We need at least one data track in common between USE/recorder and (if
% present) USE/stimulator.

isok = false;

if ~have_unity
  disp('-- Can''t do time alignment without Unity data.');
elseif ~have_recevents_dig
  disp('-- Can''t do time alignment without recorder data.');
else
  % Make sure we have a common SynchBox/recorder pair.
  % FIXME - Not aligning on synchA/synchB. Misalignment is larger than the
  % synch pulse interval, so we'd get an ambiguous result.
```

```
  if ( (~isempty(gamecodes)) && (~isempty(reccodes)) ) ...
    || ( (~isempty(gamerwdA)) && (~isempty(recrwdA)) ) ...
    || ( (~isempty(gamerwdB)) && (~isempty(recrwdB)) )

    % We have enough information to align the recorder.
    % Check the stimulator if requested.

    if ~have_stimevents_dig
      % No stimulator; we're fine as-is.
      isok = true;
    elseif ( (~isempty(reccodes)) && (~isempty(stimcodes)) ) ...
      || ( (~isempty(recrwdA)) && (~isempty(stimrwdA)) ) ...
      || ( (~isempty(recrwdB)) && (~isempty(stimrwdB)) )
      % We have enough information to align the stimulator with the recorder.
      isok = true;
    elseif ( (~isempty(gamecodes)) && (~isempty(stimcodes)) ) ...
      || ( (~isempty(gamerwdA)) && (~isempty(stimrwdA)) ) ...
      || ( (~isempty(gamerwdB)) && (~isempty(stimrwdB)) )
      % We have enough information to align the stimulator with Unity.
      isok = true;
    else
      disp('-- Not enough information to align the stimulator.');
    end

  else
    disp('-- Not enough information to align the recorder with Unity.');
  end
end

if ~isok
  % End the script here if there was a problem.
  error('Couldn''t perform time alignment; bailing out.');
end



%
% Remove enormous offsets from the various time series.

% In practice, this is the Unity timestamps, which are relative to 1 Jan 1970.
% FIXME - Leaving synchbox, recorder, and stimulator, and gaze timestamps
% as-is. The offsets in these should be modest (hours at most).

% FIXME - Leaving "system timestamps" in frame and gaze data alone.
% We'll subtract the offset when resaving as "unityTime".


% Pick an arbitrary time reference. Negative offsets relative to it are fine.

unityreftime = 0;
```

```
if ~isempty(gamecodes)
  unityreftime = min(gamecodes.unityTime);
elseif ~isempty(gamerwdA)
  unityreftime = min(gamerwdA.unityTime);
elseif ~isempty(gamerwdB)
  unityreftime = min(gamerwdB.unityTime);
end

% Subtract the time offset.
% We have a "unityTime" column in the "gameXX" and "boxXX" tables.

if ~isempty(gamecodes)
  gamecodes.unityTime = gamecodes.unityTime - unityreftime;
end

if ~isempty(gamerwdA)
  gamerwdA.unityTime = gamerwdA.unityTime - unityreftime;
end
if ~isempty(gamerwdA)
  gamerwdB.unityTime = gamerwdB.unityTime - unityreftime;
end

if ~isempty(boxcodes)
  boxcodes.unityTime = boxcodes.unityTime - unityreftime;
end

if ~isempty(boxrwdA)
  boxrwdA.unityTime = boxrwdA.unityTime - unityreftime;
end
if ~isempty(boxrwdA)
  boxrwdB.unityTime = boxrwdB.unityTime - unityreftime;
end

if ~isempty(boxsynchA)
  boxsynchA.unityTime = boxsynchA.unityTime - unityreftime;
end
if ~isempty(boxsynchA)
  boxsynchB.unityTime = boxsynchB.unityTime - unityreftime;
end



%
% Augment everything that doesn't have a time in seconds with time in seconds.

% Recorder tables get "recTime", stimulator tables get "stimTime".


if ~isempty(recrwdA)
```

```
  recrwdA.recTime = recrwdA.sample / rechdr.Fs;
end
if ~isempty(recrwdB)
  recrwdB.recTime = recrwdB.sample / rechdr.Fs;
end

if ~isempty(recsynchA)
  recsynchA.recTime = recsynchA.sample / rechdr.Fs;
end
if ~isempty(recsynchB)
  recsynchB.recTime = recsynchB.sample / rechdr.Fs;
end

if ~isempty(reccodes)
  reccodes.recTime = reccodes.sample / rechdr.Fs;
end
if ~isempty(reccodes_raw)
  reccodes_raw.recTime = reccodes_raw.sample / rechdr.Fs;
end


if ~isempty(stimrwdA)
  stimrwdA.stimTime = stimrwdA.sample / stimhdr.Fs;
end
if ~isempty(stimrwdB)
  stimrwdB.stimTime = stimrwdB.sample / stimhdr.Fs;
end

if ~isempty(stimsynchA)
  stimsynchA.stimTime = stimsynchA.sample / stimhdr.Fs;
end
if ~isempty(stimsynchB)
  stimsynchB.stimTime = stimsynchB.sample / stimhdr.Fs;
end

if ~isempty(stimcodes)
  stimcodes.stimTime = stimcodes.sample / stimhdr.Fs;
end
if ~isempty(stimcodes_raw)
  stimcodes_raw.stimTime = stimcodes_raw.sample / stimhdr.Fs;
end



%
% Propagate recorder timestamps to the SynchBox.

% Recorder and synchbox timestamps do drift but can be aligned to about 0.1ms
% precision locally (using raw, not cooked, event codes).
```

```
% Do alignment using event codes if possible. Failing that, using reward
% lines. We can't usefully align based on periodic synch signals.
% FIXME - Reward line alignment will take much longer due to not being able
% to filter based on data values.


% NOTE - We can fall back to reward alignment but not synch pulse alignment.
% The synch pulses are at regular intervals, so alignment is ambiguous.

times_recorder_synchbox = table();

if (~isempty(reccodes_raw)) && (~isempty(boxcodes_raw))
  disp('.. Aligning SynchBox and recorder using event codes.');

  % NOTE - Event code alignment with the SynchBox has to use raw codes.
  % The alignment routines misbehave trying to line up the SynchBox with
  % the ephys machines based on cooked codes, due to a large number of
  % dropped bytes (the synchbox-to-unity reply link is saturated).

  [ boxcodes_raw, reccodes_raw, boxmatchmask, recmatchmask, ...
    times_recorder_synchbox ] = ...
    euAlign_alignTables( boxcodes_raw, reccodes_raw, ...
      'synchBoxTime', 'recTime', 'codeValue', 'value', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(recrwdA)) && (~isempty(boxrwdA))
  disp('.. Aligning SynchBox and recorder using Reward A.');

  [ boxrwdA, recrwdA, boxmatchmask, recmatchmask, ...
    times_recorder_synchbox ] = ...
    euAlign_alignTables( boxrwdA, recrwdA, ...
      'synchBoxTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(recrwdB)) && (~isempty(boxrwdB))
  disp('.. Aligning SynchBox and recorder using Reward B.');

  [ boxrwdB, recrwdB, boxmatchmask, recmatchmask, ...
    times_recorder_synchbox ] = ...
    euAlign_alignTables( boxrwdB, recrwdB, ...
      'synchBoxTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
else
```

```
  disp('###  Not enough information to align recorder and SynchBox!');
end


% If we've aligned the recorder and synchbox, augment all synchbox data
% that doesn't already have recorder timestamps with recorder timestamps.

if ~isempty(times_recorder_synchbox)

  % This checks for cases where translation can't be done or where the
  % new timestamps are already present.

  boxcodes_raw = euAlign_addTimesToTable( boxcodes_raw, ...
    'synchBoxTime', 'recTime', times_recorder_synchbox );

  boxcodes = euAlign_addTimesToTable( boxcodes, ...
    'synchBoxTime', 'recTime', times_recorder_synchbox );

  boxrwdA = euAlign_addTimesToTable( boxrwdA, ...
    'synchBoxTime', 'recTime', times_recorder_synchbox );

  boxrwdB = euAlign_addTimesToTable( boxrwdB, ...
    'synchBoxTime', 'recTime', times_recorder_synchbox );

  boxsynchA = euAlign_addTimesToTable( boxsynchA, ...
    'synchBoxTime', 'recTime', times_recorder_synchbox );

  boxsynchB = euAlign_addTimesToTable( boxsynchB, ...
    'synchBoxTime', 'recTime', times_recorder_synchbox );

end



%
% Propagate recorder timestamps to USE.

% Unity timestamps have a lot more jitter (about 1.0 to 1.5 ms total).

% Do alignment using event codes if possible. Failing that, using reward
% lines.
% FIXME - Reward line alignment will take much longer due to not being able
% to filter based on data values.

% NOTE - USE's record of event codes is complete, so we can align on cooked
% codes without problems.

times_recorder_game = table();

if (~isempty(reccodes)) && (~isempty(gamecodes))
```

```
  disp('.. Aligning USE and recorder using event codes.');

  [ gamecodes, reccodes, gamematchmask, recmatchmask, ...
    times_recorder_game ] = ...
    euAlign_alignTables( gamecodes, reccodes, ...
      'unityTime', 'recTime', 'codeWord', 'codeWord', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(recrwdA)) && (~isempty(gamerwdA))
  disp('.. Aligning USE and recorder using Reward A.');

  [ gamerwdA, recrwdA, gamematchmask, recmatchmask, ...
    times_recorder_game ] = ...
    euAlign_alignTables( gamerwdA, recrwdA, ...
      'unityTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(recrwdB)) && (~isempty(gamerwdB))
  disp('.. Aligning USE and recorder using Reward B.');

  [ gamerwdB, recrwdB, gamematchmask, recmatchmask, ...
    times_recorder_game ] = ...
    euAlign_alignTables( gamerwdB, recrwdB, ...
      'unityTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
else
  disp('###  Not enough information to align recorder and USE!');
end


% If we've aligned the recorder and USE, augment all game data that doesn't
% already have recorder timestamps with recorder timestamps.

if ~isempty(times_recorder_game)

  % This checks for cases where translation can't be done or where the
  % new timestamps are already present.

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

  gamecodes = euAlign_addTimesToTable( gamecodes, ...
    'unityTime', 'recTime', times_recorder_game );
```

```
  gamerwdA = euAlign_addTimesToTable( gamerwdA, ...
    'unityTime', 'recTime', times_recorder_game );

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

end



%
% Propagate recorder timestamps to the stimulator.

% If we can do this directly, that's ideal. Otherwise go through the SynchBox.

% Do alignment using event codes if possible. Failing that, using reward
% lines. We can't usefully align based on periodic synch signals.
% FIXME - Reward line alignment will take much longer due to not being able
% to filter based on data values.

times_recorder_stimulator = table();

if (~isempty(reccodes)) && (~isempty(stimcodes))
  disp('.. Aligning stimulator and recorder using event codes.');

  [ stimcodes, reccodes, stimmatchmask, recmatchmask, ...
    times_recorder_stimulator ] = ...
    euAlign_alignTables( stimcodes, reccodes, ...
      'stimTime', 'recTime', 'codeWord', 'codeWord', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(boxcodes_raw)) && (~isempty(stimcodes_raw))
  disp('.. Aligning stimulator and recorder via SynchBox using event codes.');

  % NOTE - Event code alignment with the SynchBox has to use raw codes.
  % The alignment routines misbehave trying to line up the SynchBox with
  % the ephys machines based on cooked codes, due to a large number of
  % dropped bytes (the synchbox-to-unity reply link is saturated).

  % The "boxcodes_raw" table has already been augmented with "recTime".
  [ stimcodes_raw, boxcodes_raw, stimmatchmask, boxcmatchmask, ...
    times_recorder_stimulator ] = ...
    euAlign_alignTables( stimcodes_raw, boxcodes_raw, ...
      'stimTime', 'recTime', 'value', 'codeValue', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );
```

```
  disp('.. Finished aligning.');
elseif (~isempty(recrwdA)) && (~isempty(stimrwdA))
  disp('.. Aligning stimulator and recorder using Reward A.');

  [ stimrwdA, recrwdA, stimmatchmask, recmatchmask, ...
    times_recorder_stimulator ] = ...
    euAlign_alignTables( stimrwdA, recrwdA, ...
      'stimTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(recrwdB)) && (~isempty(stimrwdB))
  disp('.. Aligning stimulator and recorder using Reward B.');

  [ stimrwdB, recrwdB, stimmatchmask, recmatchmask, ...
    times_recorder_stimulator ] = ...
    euAlign_alignTables( stimrwdB, recrwdB, ...
      'stimTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(boxrwdA)) && (~isempty(stimrwdA))
  disp('.. Aligning stimulator and recorder via SynchBox using Reward A.');

  % The "boxrwdA" table has already been augmented with "recTime".
  [ stimrwdA, boxrwdA, stimmatchmask, boxmatchmask, ...
    times_recorder_stimulator ] = ...
    euAlign_alignTables( stimrwdA, boxrwdA, ...
      'stimTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
elseif (~isempty(boxrwdB)) && (~isempty(stimrwdB))
  disp('.. Aligning stimulator and recorder via SynchBox using Reward B.');

  % The "boxrwdB" table has already been augmented with "recTime".
  [ stimrwdB, boxrwdB, stimmatchmask, boxmatchmask, ...
    times_recorder_stimulator ] = ...
    euAlign_alignTables( stimrwdB, boxrwdB, ...
      'stimTime', 'recTime', '', '', ...
      aligncoarsewindows, alignmedwindows, alignfinewindow, ...
      alignoutliersigma, alignverbosity );

  disp('.. Finished aligning.');
else
  disp('###  Not enough information to align recorder and stimulator!');
end
```

```
% If we've aligned the recorder and stimulator, augment all stimulator data
% that doesn't already have recorder timestamps with recorder timestamps.

if ~isempty(times_recorder_stimulator)

  % This checks for cases where translation can't be done or where the
  % new timestamps are already present.

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

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

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

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

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

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

end



%
% Time-align gaze data and frame data if possible.


% Align USE timestamps with eye-tracker timestamps.
% The "frame data" table has this information already; we just have to pick
% the subset of points that actually correspond.

times_game_eyetracker = table();

if ~isempty(gameframedata_raw)
  disp('.. Aligning USE and eye-tracker using FrameData table.');

  % We have two columns: "SystemTimeSeconds" and "EyetrackerTimeSeconds".
  % System timestamps (Unity) are unique; ET timestamps aren't.
  % Pick the smallest system timestamp for each ET timestamp.

  systimes_raw = gameframedata_raw.SystemTimeSeconds;
```

```
  eyetimes_raw = gameframedata_raw.EyetrackerTimeSeconds;

  eyetimes = unique(eyetimes_raw);

  systimes = [];
  for eidx = 1:length(eyetimes)
    thistime = eyetimes(eidx);
    thissys = systimes_raw(eyetimes_raw == thistime);
    systimes(eidx) = min(thissys);
  end

  if ~iscolumn(systimes)
    systimes = transpose(systimes);
  end
  if ~iscolumn(eyetimes)
    eyetimes = transpose(eyetimes);
  end

  times_game_eyetracker.unityTime = systimes;
  times_game_eyetracker.eyeTime = eyetimes;
end

if isempty(times_game_eyetracker)
  disp('###  Not enough information to align eye-tracker!');
else
  disp('.. Finished aligning.');
end


% Propagate relevant time fields to the GazeData and FrameData tables.

if ~isempty(gameframedata_raw)

  disp( ...
'.. Augmenting FrameData with recorder and interpolated gaze timestamps.' );

  % Save copies of timestamp columns with our standard names.
  % Interpolate timestamps for the ET data saved with successive Unity times.

  % NOTE - Remember to subtract the enormous offset from the Unity timestamp!

  gameframedata_raw.unityTime = ...
    gameframedata_raw.SystemTimeSeconds - unityreftime;

  % Interpolate gaze timestamps.
  % We should always have this alignment table if we have gameframedata_raw.
  if ~isempty(times_game_eyetracker)
    gameframedata_raw = euAlign_addTimesToTable( gameframedata_raw, ...
      'unityTime', 'eyeTime', times_game_eyetracker );
  end
```

```
  % Augment with the recorder timestamp.
  if ~isempty(times_recorder_game)
    gameframedata_raw = euAlign_addTimesToTable( gameframedata_raw, ...
      'unityTime', 'recTime', times_recorder_game );
  end

  disp('.. Finished augmenting.');

end

if ~isempty(gamegaze_raw)

  disp('.. Augmenting GazeData with recorder and USE timestamps.');

  % Save a renamed copy of the ET timestamp.
  gamegaze_raw.eyeTime = gamegaze_raw.time_seconds;

  % Augment with USE time if we have a translation table for that.
  % If we can get USE timestamps, augment with recorder timestamps if we
  % have a table for _that_.

  if ~isempty(times_game_eyetracker)
    gamegaze_raw = euAlign_addTimesToTable( gamegaze_raw, ...
      'eyeTime', 'unityTime', times_game_eyetracker );

    if ~isempty(times_recorder_game)
      gamegaze_raw = euAlign_addTimesToTable( gamegaze_raw, ...
        'unityTime', 'recTime', times_recorder_game );
    end
  end

  disp('.. Finished augmenting.');

end


% Build a direct mapping table from ET timestamps to recorder timestamps.

times_recorder_eyetracker = table();

if (~isempty(times_game_eyetracker)) && (~isempty(times_recorder_game))
  times_scratch = times_game_eyetracker;
  times_scratch = euAlign_addTimesToTable( times_scratch, ...
    'unityTime', 'recTime', times_recorder_game );

  times_recorder_eyetracker.recTime = times_scratch.recTime;
  times_recorder_eyetracker.eyeTime = times_scratch.eyeTime;
end
```

```
%
% Save the results to disk, if requested.

if want_save_data
  if isfile(fname_cookedevents) ; delete(fname_cookedevents) ; end
  if isfile(fname_cookedgaze)   ; delete(fname_cookedgaze)   ; end
  if isfile(fname_cookedframe)  ; delete(fname_cookedframe)  ; end

  disp('-- Saving time-aligned Unity events and alignment tables.');

  % NOTE - Only save the tables we annotated, and selected metadata.
  % In particular recevents_dig and stimevents_dig are huge and raw TTL.
  % There's no further need for them and they're alreadys saved as raw data.

  save( fname_cookedevents, ...
    'have_unity', 'evcodedefs', ...
    'boxsynchA', 'boxsynchB', 'boxrwdA', 'boxrwdB', ...
    'boxcodes', 'boxcodes_raw', ...
    'gamerwdA', 'gamerwdB', 'gamecodes', 'gamecodes_raw', ...
    'have_recrwdA', 'recrwdA', 'have_recrwdB', 'recrwdB', ...
    'have_recsynchA', 'recsynchA', 'have_recsynchB', 'recsynchB', ...
    'have_reccodes', 'reccodes', 'reccodes_raw', ...
    'have_stimrwdA', 'stimrwdA', 'have_stimrwdB', 'stimrwdB', ...
    'have_stimsynchA', 'stimsynchA', 'have_stimsynchB', 'stimsynchB', ...
    'have_stimcodes', 'stimcodes', 'stimcodes_raw', ...
    'times_recorder_synchbox', 'times_recorder_game', ...
    'times_recorder_stimulator', 'times_game_eyetracker', ...
    'times_recorder_eyetracker', 'unityreftime', ...
    '-v7.3' );

  disp('-- Saving time-aligned gaze data.');

  save( fname_cookedgaze, 'gamegaze_raw', '-v7.3' );

  disp('-- Saving time-aligned Unity frame data.');

  save( fname_cookedframe, 'gameframedata_raw', '-v7.3' );

  disp('-- Finished saving.');
end



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

Chapter 5

Trial Processing

5.1 do_test_define_trials.m

```
% Field Trip sample script / test script - Trial definitions.
% Written by Christopher Thomas.

% This processes event code data and produces Field Trip trial definition
% structures, along with metadata tables.
% FIXME - Doing this by reading and setting workspace variables directly.
%
% Variables that get set:
%   trialcodes_raw
%   trialcodes_valid
%   trialdefs
%   trialdefcolumns
%   trialdeftables



%
% Load cached time-aligned event data if we don't already have it.

if ~exist('times_recorder_game', 'var')
  fname_aligned = [ datadir filesep 'events_aligned.mat' ];

  if ~isfile(fname_aligned)
    % No time-aligned data. Abort the script and send the user back to
    % the Matlab prompt.
    error('Can''t define trials without time alignment information.');
  else
    disp('-- Loading time-aligned Unity events and alignment tables.');
    load(fname_aligned);
    disp('-- Finished loading.');
  end
end
```

```
%
% Build trial definitions.


% First pass: Segment the event code sequence into trials.
% Use Unity's list of event codes, since it's guaranteed to exist.

allcodes = gamecodes;
codelabels = allcodes.codeLabel;

trialcodes_raw = {};
segcount = 0;
prevstart = NaN;

for cidx = 1:length(codelabels);
  thislabel = codelabels{cidx};

  if strcmp('TrlStart', thislabel)
    % Check for malformed trials.
    if ~isnan(prevstart)
      disp(sprintf( '###  "TrlStart" inside a trial at line %d.', cidx ));
    end

    % Start or restart the trial.
    prevstart = cidx;
  elseif strcmp('TrlEnd', thislabel)
    % Check for malformed trials.
    if isnan(prevstart)
      disp(sprintf( '###  "TrlEnd" without trial start at line %d.', cidx ));
    else
      % Only create a trial record if we ended a correctly-formed trial.
      % Save the in-trial portion of the code table.
      segcount = segcount + 1;
      trialcodes_raw{segcount} = allcodes(prevstart:cidx,:);
    end

    % End the trial no matter what.
    prevstart = NaN;
  end
end


% Filter the list of trials to only contain "valid" trials.

% FIXME - I'm not clear on how to do this. Marcus said that TrialNumber only
% increments after valid trials, and that there's an abort code to look for
% for other trials, but that filtering based on these wasn't a great idea.
```

```
% So, stick a user-defined function here to do it the way you want.

trialcodes_valid = doSelectGoodTrials(trialcodes_raw);



% Second pass: Iterate through the list of alignment cases, building
% trial definitions for each case.

samprate = rechdr.Fs;
trialdefs = struct();
trialdeftables = struct();
aligncases = fieldnames(trialaligncodes);

% FIXME - Magic values. These are the labels of the trial matrix columns.
% The first three are FT's required columns. Additional columns get copied
% to a "trialinfo" matrix. This can only contain numeric data.
% To get full event data for a trial, load trialcodes_valid{validindex}.

trialdefcolumns = { 'sampstart', 'sampend', 'sampoffset', ...
  'validindex', 'trialindex', 'trialnum', ...
  'rectimestart', 'rectimeend', 'rectimeevent' };

for cidx = 1:length(aligncases)

  thisalignlabel = aligncases{cidx};
  thisaligncodelist = trialaligncodes.(thisalignlabel);

  thistrialdef = [];
  thistrialcount = 0;

  for tidx = 1:length(trialcodes_valid)

    thistable = trialcodes_valid{tidx};
    thislabellist = thistable.codeLabel;

    % Find the span to save.

    timestart = NaN;
    timestop = NaN;

    thisridx = helper_findRow(thislabellist, trialstartcodes, 'last');
    if ~isnan(thisridx)
      timestart = thistable.recTime(thisridx);
    end

    thisridx = helper_findRow(thislabellist, trialendcodes, 'first');
    if ~isnan(thisridx)
      timestop = thistable.recTime(thisridx);
    end
```

```
% Find the event to align to.

timealign = NaN;

thisridx = helper_findRow(thislabellist, thisaligncodelist, 'first');
if ~isnan(thisridx)
  timealign = thistable.recTime(thisridx);
end


% If we found everything we're looking for, create a trial definition.
% Save metadata as extra columns.
if (~isnan(timestart)) && (~isnan(timestop)) && (~isnan(timealign))

  thistrialcount = thistrialcount + 1;

  % FIXME - Per above, our desired columns are:
  % sampstart, sampend, sampoffset (FT's required columns)
  % validindex, trialindex, trialnum
  % rectimestart, rectimeend, rectimevent

  timestart = timestart - trialstartpadsecs;
  timestop = timestop + trialendpadsecs;

  sampstart = round(timestart * samprate);
  sampstop = round(timestop * samprate);
  sampalign = round(timealign * samprate);

  validindex = tidx;
  trialindex = NaN;
  trialnum = NaN;

  % These should always exist but bulletproof it anyways.

  thisridx = helper_findRow(thislabellist, {'TrialIndex'}, 'last');
  if ~isnan(thisridx)
    trialindex = thistable.codeData(thisridx);
  end

  thisridx = helper_findRow(thislabellist, {'TrialNumber'}, 'last');
  if ~isnan(thisridx)
    trialnum = thistable.codeData(thisridx);
  end

  % Offset is negative if the trial starts before the trigger.
  thistrialdef(thistrialcount,:) = ...
    [ sampstart, sampstop, (sampstart - sampalign), ...
      validindex, trialindex, trialnum, timestart, timestop, timealign ];
```

```
    end

  end


  % Save this trial definition matrix. This is empty if we found no trials.
  trialdefs.(thisalignlabel) = thistrialdef;


  % Make a copy of the augmented trial definition as a proper table with
  % readable labels. Save this as a CSV file as well.

  trialtab = table();

  for lidx = 1:length(trialdefcolumns)
    thiscolumn = [];
    if ~isempty(thistrialdef)
      thiscolumn = thistrialdef(:,lidx);
    end
    trialtab.(trialdefcolumns{lidx}) = thiscolumn;
  end

  trialdeftables.(thisalignlabel) = trialtab;

  fname = [ datadir filesep 'trialdefs-' thisalignlabel '.csv' ];
  writetable(trialtab, fname);

end



%
% Save variables to disk, if requested.

% NOTE - There isn't much point, since these are fast to generate, but we
% might want it for auditing purposes.

if want_save_data
  fname = [ datadir filesep 'trialmetadata.mat' ];

  if isfile(fname)
    delete(fname);
  end

  disp('-- Saving trial definition metadata.');

  save( fname, ...
    'trialcodes_raw', 'trialcodes_valid', ...
    'trialdefs', 'trialdefcolumns', 'trialdeftables', ...
    '-v7.3' );
```

```
  disp('-- Finished saving.');
end



%
% Helper functions.


% This finds an entry in the first list that matches one of the labels in
% the second list.
%
% "rowlabels" is a cell array containing labels to search.
% "desiredlabels" is a cell array containing valid matches for the search.
% "order" is 'first' to return the index of the earliest match or 'last' to
%   return the index of the latest match.
%
% "rowindex" is the index of the matching entry, or NaN if no match was found.

function rowindex = helper_findRow(rowlabels, desiredlabels, order)

  matchlist = [];

  for didx = 1:length(desiredlabels)
    thismatch = strcmp(rowlabels, desiredlabels{didx});
    if ~isrow(thismatch)
      thismatch = transpose(thismatch);
    end
    matchlist = [ matchlist find(thismatch) ];
  end

  if strcmp('last', order)
    rowindex = max(matchlist);
  else
    rowindex = min(matchlist);
  end

  if isempty(rowindex)
    rowindex = NaN;
  end

% Done.

end



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

5.2 do_test_process_trials.m

```
% Field Trip sample script / test script - Epoched data processing.
% Written by Christopher Thomas.

% This reads data according to predefined trial definitions, processes and
% saves it trial by trial, and optionally displays it using FT's browser.
% FIXME - Doing this by reading and setting workspace variables directly.
%
% NOTE - Most data is computed per-batch and saved to per-batch files,
% rather than saved in workspace variables.
%
%
% Data that persists in workspace variables:
%
% trialbatchmeta
%
%
% Variables saved per-batch:
%
% thisbatchlabel
% thisbatchtrials_rec
% thisbatchtrials_stim  (only the three mandatory columns)
% thisbatchtrials_gaze  (only the three mandatory columns)
% thisbatchtable_rec  (same as trials_rec but with column headings)
% trialdefcolumns
%
% have_batchdata_rec
% batchdata_rec_wb
% batchdata_rec_lfp
% batchdata_rec_spike
% batchdata_rec_rect
%
% have_batchdata_stim
% batchdata_stim_wb
% batchdata_stim_lfp
% batchdata_stim_spike
% batchdata_stim_rect
%
% batchevents_codes
% batchevents_rwdA
% batchevents_rwdB
%
% batchrows_gaze
% batchrows_frame
%
% batchdata_gaze
```

```
%
% Load cached data if we don't already have it.

% Trial definitions.

if ~exist('trialdefs', 'var')
  fname_trialdefs = [ datadir filesep 'trialmetadata.mat' ];
  if ~isfile(fname_trialdefs)
    error('Can''t process trials without trial definitions.');
  else
    disp('-- Loading trial definitions.');
    load(fname_trialdefs);
    disp('-- Finished loading.');
  end
end

% "Game" event lists and recorder/stimulator time translation table.

if (~exist('gamecodes', 'var')) ...
  || (~exist('times_recorder_stimulator', 'var'))

  fname_events = [ datadir filesep 'events_aligned.mat' ];

  if ~isfile(fname_events)
    % We can live without events, but we need the alignment tables.
    error('Can''t process trials without recorder/stimulator alignment.');
  else
    disp('-- Loading time-aligned Unity events and alignment tables.');
    load(fname_events);
    disp('-- Finished loading.');
  end

end

% Get frame and gaze data tables.

if ~exist('gamegaze_raw', 'var')
  fname_gaze = [ datadir filesep 'gaze_aligned.mat' ];
  if ~isfile(fname_gaze)
    error('Can''t find gaze data for creating trials.');
  else
    disp('-- Loading time-aligned gaze data.');
    load(fname_gaze);
    disp('-- Finished loading.');
  end
end

if ~exist('gameframedata_raw', 'var')
  fname_frame = [ datadir filesep 'frame_aligned.mat' ];
  if ~isfile(fname_frame)
```

```
    error('Can''t find frame data for creating trials.');
  else
    disp('-- Loading time-aligned frame data.');
    load(fname_frame);
    disp('-- Finished loading.');
  end
end


% Extract various fields we'd otherwise have to keep looking up.

times_recstim_rec = times_recorder_stimulator.recTime;
times_recstim_stim = times_recorder_stimulator.stimTime;

gamecoderectime = [];
gamerwdArectime = [];
gamerwdBrectime = [];
if ~isempty(gamecodes) ; gamecoderectime = gamecodes.recTime ; end
if ~isempty(gamerwdA)  ; gamerwdArectime = gamerwdA.recTime  ; end
if ~isempty(gamerwdB)  ; gamerwdBrectime = gamerwdB.recTime  ; end

gamegazerectime = [];
gameframerectime = [];
if ~isempty(gamegaze_raw) ; gamegazerectime = gamegaze_raw.recTime ; end
if ~isempty(gameframedata_raw)
  gameframerectime = gameframedata_raw.recTime;
end


% Set up the table reader to read from FrameData.

% FIXME - Windows get clamped to the time ranges specified here, so the time
% range given here matters. We should calculate the maximum recTime timestamp
% we'd have in the ephys data, but instead just pad the last gaze recTime by
% 10 seconds.

frametimes = gameframedata_raw.recTime;
nlFT_initReadTable( gameframedata_raw, frame_gaze_cols, 'recTime', ...
  0.0, 10.0 + max(frametimes), gaze_rate, gaze_rate );


%
% Process trials.

% Banner.
disp('== Processing epoched trial data.');

trialcases = fieldnames(trialdefs);
trialbatchmeta = struct();
```

```
for caseidx = 1:length(trialcases)

  % Get alignment case metadata.

  thiscaselabel = trialcases{caseidx};
  thistrialdefs = trialdefs.(thiscaselabel);
  thistrialdeftable = trialdeftables.(thiscaselabel);


  % Split this case's trials into batches small enough to process.

  % Default to monolithic.

  trialcount = size(thistrialdefs);
  trialcount = trialcount(1);

  batchlabels = { thiscaselabel };
  batchtrialdefs = { thistrialdefs };
  batchtrialdeftables = { thistrialdeftable };

  % If we have too many trials, break it into batches.

  if trialcount > trials_per_batch
    batchlabels = {};
    batchtrialdefs = {};
    trialsfirst = 1:trials_per_batch:trialcount;
    trialslast = min(( trialsfirst + trials_per_batch - 1), trialcount );

    for bidx = 1:length(trialsfirst)
      thistrialfirst = trialsfirst(bidx);
      thistriallast = trialslast(bidx);

      batchlabels{bidx} = sprintf('%s-batch%04d', thiscaselabel, bidx);
      batchtrialdefs{bidx} = ...
        thistrialdefs(thistrialfirst:thistriallast,:);
      batchtrialdeftables{bidx} = ...
        thistrialdeftable(thistrialfirst:thistriallast,:);
    end
  end


  % Identify certain special batches, for debugging.

  earlybatch = round(1 + 0.2 * length(batchlabels));
  earlybatch = min(earlybatch, length(batchlabels));
  middlebatch = round(1 + 0.5 * length(batchlabels));
  middlebatch = min(middlebatch, length(batchlabels));
  latebatch = round(1 + 0.8 * length(batchlabels));
  latebatch = min(latebatch, length(batchlabels));
```

```
%
% Process this case's trial batches.

% NOTE - There's a debug switch to process only a single batch, for testing.

batchspan = 1:length(batchlabels);
if want_one_batch
  batchspan = middlebatch:middlebatch;
end

plotbatches = [ earlybatch middlebatch latebatch ];

for bidx = batchspan

  thisbatchlabel = batchlabels{bidx};
  thisbatchtrials_rec = batchtrialdefs{bidx};
  % This has the same information as "trials", but has column labels.
  thisbatchtable_rec = batchtrialdeftables{bidx};

  fname_batch = [ datadir filesep 'trials-' thisbatchlabel '.mat' ];
  need_save = false;

  if want_cache_epoched && isfile(fname_batch)

    %
    % Load the data we previously processed.

    disp([ '.. Loading batch "' thisbatchlabel '".' ]);
    load(fname_batch);
    disp([ '.. Finished loading.' ]);

  else

    %
    % Rebuild data for this set of trials.


    disp([ '.. Reading recorder data for batch "' thisbatchlabel '".' ]);

    % Read and process recorder trials.
    % Sample counts are fine as-is.

    preproc_config_rec.trl = thisbatchtrials_rec;

    % Turn off the progress bar.
    preproc_config_rec.feedback = 'no';

    have_batchdata_rec = false;
```

```
batchdata_rec_wb = struct([]);
batchdata_rec_lfp = struct([]);
batchdata_rec_spike = struct([]);
batchdata_rec_rect = struct([]);

if ~isempty(rec_channels_ephys)
  preproc_config_rec.channel = rec_channels_ephys;
  batchdata_rec_wb = ft_preprocessing(preproc_config_rec);
  have_batchdata_rec = true;

  % De-trend and remove power line noise.
  extra_notches = [];
  if isfield( thisdataset, 'extra_notches' )
    extra_notches = thisdataset.extra_notches;
  end
  batchdata_rec_wb = doSignalConditioning( batchdata_rec_wb, ...
    power_freq, power_filter_modes, filter_type_short, extra_notches );

  % Extract processed signals of interest.
  [ batchdata_rec_lfp batchdata_rec_spike batchdata_rec_rect ] = ...
    doFeatureFiltering( batchdata_rec_wb, ...
      lfp_corner, lfp_rate, spike_corner, ...
      rect_corners, rect_lowpass, rect_rate );

  if want_reref && isfield( thisdataset, 'commonrefs_rec' )
    batchdata_rec_lfp = doCommonAverageReference( ...
      batchdata_rec_lfp, thisdataset.commonrefs_rec );
  end
end


disp([ '.. Reading stimulator data for batch "' thisbatchlabel '".' ]);

% Convert recorder trial definition samples to stimulator samples.
% Remember that sample indices are 1-based.

thisstart = thisbatchtrials_rec(:,1);
thisend = thisbatchtrials_rec(:,2);
thisoffset = thisbatchtrials_rec(:,3);

thisstart = (thisstart - 1) / rechdr.Fs;
thisstart = nlProc_interpolateSeries( ...
  times_recstim_rec, times_recstim_stim, thisstart );
thisstart = 1 + round(thisstart * stimhdr.Fs);

thisend = (thisend - 1) / rechdr.Fs;
thisend = nlProc_interpolateSeries( ...
  times_recstim_rec, times_recstim_stim, thisend );
thisend = 1 + round(thisend * stimhdr.Fs);
```

```
thisoffset = 1 + round((thisoffset - 1) * stimhdr.Fs / rechdr.Fs);

thisbatchtrials_stim = [];
thisbatchtrials_stim(:,1) = thisstart;
thisbatchtrials_stim(:,2) = thisend;
thisbatchtrials_stim(:,3) = thisoffset;

% Read and process stimulator trials.

preproc_config_stim.trl = thisbatchtrials_stim;

% Turn off the progress bar.
preproc_config_stim.feedback = 'no';

have_batchdata_stim = false;

batchdata_stim_wb = struct([]);
batchdata_stim_lfp = struct([]);
batchdata_stim_spike = struct([]);
batchdata_stim_rect = struct([]);

if ~isempty(stim_channels_ephys)
  preproc_config_stim.channel = stim_channels_ephys;
  batchdata_stim_wb = ft_preprocessing(preproc_config_stim);
  have_batchdata_stim = true;

  % De-trend and remove power line noise.
  extra_notches = [];
  if isfield( thisdataset, 'extra_notches' )
    extra_notches = thisdataset.extra_notches;
  end
  batchdata_stim_wb = doSignalConditioning( batchdata_stim_wb, ...
    power_freq, power_filter_modes, filter_type_short, extra_notches );

  % Extract processed signals of interest.
  [ batchdata_stim_lfp batchdata_stim_spike batchdata_stim_rect ] = ...
    doFeatureFiltering( batchdata_stim_wb, ...
      lfp_corner, lfp_rate, spike_corner, ...
      rect_corners, rect_lowpass, rect_rate );

  if want_reref && isfield( thisdataset, 'commonrefs_stim' )
    batchdata_stim_lfp = doCommonAverageReference( ...
      batchdata_stim_lfp, thisdataset.commonrefs_stim );
  end
end


disp([ '.. Copying Unity event data for batch "' thisbatchlabel '".' ]);

% NOTE - We've loaded "events_aligned.mat" when building trial
```

```
    % definitions. This gives us "gamecodes", "gamerwdA", and "gamerwdB",
    % among other things. Those are the events that we care about.

    % We always "have" event data, but a batch or a trial may have 0 events.

    batchevents_codes = {};
    batchevents_rwdA = {};
    batchevents_rwdB = {};

    for tidx = 1:height(thisbatchtable_rec)
      thisrectimestart = thisbatchtable_rec.rectimestart(tidx);
      thisrectimeend = thisbatchtable_rec.rectimeend(tidx);

      thistrial_codes = table();
      thistrial_rwdA = table();
      thistrial_rwdB = table();

      if ~isempty(gamecodes)
        thismask = (gamecoderectime >= thisrectimestart) ...
          & (gamecoderectime <= thisrectimeend);
        thistrial_codes = gamecodes( thismask, : );
      end

      if ~isempty(gamerwdA)
        thismask = (gamerwdArectime >= thisrectimestart) ...
          & (gamerwdArectime <= thisrectimeend);
        thistrial_rwdA = gamerwdA( thismask, : );
      end

      if ~isempty(gamerwdB)
        thismask = (gamerwdBrectime >= thisrectimestart) ...
          & (gamerwdBrectime <= thisrectimeend);
        thistrial_rwdB = gamerwdB( thismask, : );
      end

      batchevents_codes{tidx} = thistrial_codes;
      batchevents_rwdA{tidx} = thistrial_rwdA;
      batchevents_rwdB{tidx} = thistrial_rwdB;
    end


    disp([ ...
'.. Copying Unity "FrameData" and "GazeData" rows for batch "' ...
      thisbatchlabel '".' ]);

    batchrows_gaze = {};
    batchrows_frame = {};

    for tidx = 1:height(thisbatchtable_rec)
      thisrectimestart = thisbatchtable_rec.rectimestart(tidx);
```

```
  thisrectimeend = thisbatchtable_rec.rectimeend(tidx);

  thistrial_gaze = table();
  thistrial_frame = table();

  if ~isempty(gamegaze_raw)
    thismask = (gamegazerectime >= thisrectimestart) ...
      & (gamegazerectime <= thisrectimeend);
    thistrial_gaze = gamegaze_raw( thismask, : );
  end

  if ~isempty(gameframedata_raw)
    thismask = (gameframerectime >= thisrectimestart) ...
      & (gameframerectime <= thisrectimeend);
    thistrial_frame = gameframedata_raw( thismask, : );
  end

  batchrows_gaze{tidx} = thistrial_gaze;
  batchrows_frame{tidx} = thistrial_frame;
end


disp( '.. Converting cooked gaze coordinates to waveforms.' );

% NOTE - The raw gaze information in "gamegaze_raw" is in three
% different eye-tracker-specific coordinate systems. We're not going
% to deal with that headache here. There's cooked gaze information in
% "framedata_raw", which should be adequate for this script.

% If you need to process the original high-sampling-rate raw data,
% the same function calls used here should work for converting it to
% Field Trip format.


% We're calling ft_read_header() and ft_read_data() with LoopUtil
% hooks that make it read from tabular data in memory.


% Read the header. Among other things this gives use the sampling rate
% (that we set earlier).
% NOTE - Field Trip tests that the file exists, so give it the
% Unity folder.
gazehdr = ft_read_header( thisdataset.unityfile, ...
  'headerformat', 'nlFT_readTableHeader' );


% Convert recorder trial definition samples to gaze samples.
% These are already aligned to recTime 0 (sample 1); just rescale.

thisstart = thisbatchtrials_rec(:,1);
```

```
      thisend = thisbatchtrials_rec(:,2);
      thisoffset = thisbatchtrials_rec(:,3);

      thisstart = (thisstart - 1) / rechdr.Fs;
      thisstart = 1 + round(thisstart * gazehdr.Fs);

      thisend = (thisend - 1) / rechdr.Fs;
      thisend = 1 + round(thisend * gazehdr.Fs);

      thisoffset = 1 + round((thisoffset - 1) * gazehdr.Fs / rechdr.Fs);

      thisbatchtrials_gaze = [];
      thisbatchtrials_gaze(:,1) = thisstart;
      thisbatchtrials_gaze(:,2) = thisend;
      thisbatchtrials_gaze(:,3) = thisoffset;


      % Read and process gaze trials.

      % NOTE - Field Trip tests that the file exists, so give it the
      % Unity folder.
      preproc_config_gaze = struct( ...
        'headerfile', thisdataset.unityfile, ...
        'datafile', thisdataset.unityfile, ...
        'headerformat', 'nlFT_readTableHeader', ...
        'dataformat', 'nlFT_readTableData', ...
        'trl', thisbatchtrials_gaze );

      % Turn off the progress bar.
      preproc_config_gaze.feedback = 'no';

      % Select channels.
      preproc_config_gaze.channel = ...
        ft_channelselection( frame_gaze_cols, gazehdr.label, {} );

      % Read the gaze data out of the table.
      batchdata_gaze = ft_preprocessing(preproc_config_gaze);


      %
      % Save this batch of trial data.

      disp([ '.. Saving trial batch "' thisbatchlabel '".' ]);


% NOTE - Variables being saved per batch.
%
% thisbatchlabel
% thisbatchtrials_rec
% thisbatchtrials_stim  (only the three mandatory columns)
```

```
% thisbatchtrials_gaze  (only the three mandatory columns)
% thisbatchtable_rec  (same as trials_rec but with column headings)
% trialdefcolumns
%
% have_batchdata_rec
% batchdata_rec_wb
% batchdata_rec_lfp
% batchdata_rec_spike
% batchdata_rec_rect
%
% have_batchdata_stim
% batchdata_stim_wb
% batchdata_stim_lfp
% batchdata_stim_spike
% batchdata_stim_rect
%
% batchevents_codes
% batchevents_rwdA
% batchevents_rwdB
%
% batchrows_gaze
% batchrows_frame
%
% batchdata_gaze

      save( fname_batch, ...
        'thisbatchlabel', 'thisbatchtrials_rec', 'thisbatchtable_rec', ...
        'trialdefcolumns', 'thisbatchtrials_stim', 'thisbatchtrials_gaze', ...
        'have_batchdata_rec', 'batchdata_rec_wb', 'batchdata_rec_lfp', ...
        'batchdata_rec_spike', 'batchdata_rec_rect', ...
        'have_batchdata_stim', 'batchdata_stim_wb', 'batchdata_stim_lfp', ...
        'batchdata_stim_spike', 'batchdata_stim_rect', ...
        'batchevents_codes', 'batchevents_rwdA', 'batchevents_rwdB', ...
        'batchrows_gaze', 'batchrows_frame', 'batchdata_gaze', ...
        '-v7.3' );

      disp([ '.. Finished saving.' ]);
    end


    % Generate plots for this batch, if appropriate.

    if want_plots && ismember(bidx, plotbatches)

      disp([ '.. Plotting trial batch "' thisbatchlabel '".' ]);

      fbase_plot = [ plotdir filesep 'trials' ];

      doPlotBatchTrials( fbase_plot, thisbatchlabel, thisbatchtable_rec, ...
        batchdata_rec_wb, batchdata_rec_lfp, batchdata_rec_spike, ...
```

```
        batchdata_rec_rect, batchdata_stim_wb, batchdata_stim_lfp, ...
        batchdata_stim_spike, batchdata_stim_rect, batchdata_gaze, ...
        batchevents_codes, batchevents_rwdA, batchevents_rwdB );

      close all;

      disp([ '.. Finished plotting.' ]);

    end


    % If this is the batch we want to display, display it.

    if want_browser && (bidx == middlebatch)

      disp([ '-- Rendering waveforms for batch "' thisbatchlabel '".' ]);

      doBrowseFiltered( 'Rec', batchdata_rec_wb, batchdata_rec_lfp, ...
        batchdata_rec_spike, batchdata_rec_rect );
      doBrowseFiltered( 'Stim', batchdata_stim_wb, batchdata_stim_lfp, ...
        batchdata_stim_spike, batchdata_stim_rect );

      % FIXME - Not browsing event data or frame table data or gaze table data.

      % Do browse gaze _waveform_ data.
      doBrowseWave( batchdata_gaze, 'Gaze' );

      disp('-- Press any key to continue.');
      pause;

      close all;

    end

  end


  % Record batch metadata.

  % Remember to wrap cell arrays in {}.
  trialbatchmeta.(thiscaselabel) = struct( ...
    'batchlabels', { batchlabels }, 'batchtrialdefs', { batchtrialdefs }, ...
    'batchtrialdeftables', { batchtrialdeftables } );


  % Finished with this alignment case.

end
```

```
% Release memory taken up by the table reader.
nlFT_initReadTable( table(), {}, 'recTime', 0.0, 1.0, gaze_rate, gaze_rate );


% Save batch metadata.

fname = [ datadir filesep 'batchmetadata.mat' ];
save( fname, 'trialbatchmeta', '-v7.3' );


% Banner.
disp('== Finished processing epoched trial data.');


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

Chapter 6

Helper Functions

6.1 doCommonAverageReference.m

```
function newdata = doCommonAverageReference( olddata, commongroups )

% function newdata = doCommonAverageReference( olddata, commongroups )
%
% This re-references selected groups of channels in a Field Trip dataset.
%
% "olddata" is the Field Trip dataset to process.
% "commongroups" is a cell array containing signal groups. Each signal group
%   is a cell array containing Field Trip channel labels for the group's
%   signals.
%
% "newdata" is a copy of "olddata" with the group average subtracted from
%   all signals that are members of a group.

newdata = olddata;

groupcount = length(commongroups);
chancount = length(newdata.label);

if groupcount > 0
  for tidx = 1:length(newdata.trial)

    thistrial = newdata.trial{tidx};

    sampcount = size(thistrial);
    sampcount = sampcount(2);

    for gidx = 1:groupcount
      thisgroup = commongroups{gidx};

      thiscommon = zeros(1, sampcount);
```

```
      thiscount = 0;

      for cidx = 1:chancount
        thislabel = newdata.label{cidx};
        if ismember(thislabel, thisgroup)
          thiscount = thiscount + 1;
          thiscommon = thiscommon + thistrial(cidx,:);
        end
      end

      if thiscount > 0
        thiscommon = thiscommon / thiscount;
        for cidx = 1:chancount
          thislabel = newdata.label{cidx};
          if ismember(thislabel, thisgroup)
            thistrial(cidx,:) = thistrial(cidx,:) - thiscommon;
          end
        end
      end
    end

    newdata.trial{tidx} = thistrial;

  end
end


% Done.

end


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

6.2 doFeatureFiltering.m

```
function [ datalfp dataspike datarect ] = doFeatureFiltering( ...
  datawide, lfp_corner, lfp_rate, spike_corner, ...
  rect_band, rect_lowpass, rect_rate )

% function [ datalfp dataspike datarect ] = doFeatureFiltering( ...
%   datawide, lfp_corner, lfp_rate, spike_corner, ...
%   rect_band, rect_lowpass, rect_rate )
%
% 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).
%
% "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.
% "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.
%
% "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.


% Produce LFP signals.

filtconfig = ...
  struct( 'lpfilter', 'yes', 'lpfilttype', 'but', 'lpfreq', lfp_corner );
resampleconfig = struct( 'resamplefs', lfp_rate, 'detrend', 'no' );

filtconfig.feedback = 'no';
resampleconfig.feedback = 'no';

datalfp = ft_preprocessing(filtconfig, datawide);
datalfp = ft_resampledata(resampleconfig, datalfp);


% Produce spike signals.

filtconfig = struct( ...
  'hpfilter', 'yes', 'hpfilttype', 'but', 'hpfreq', spike_corner );

filtconfig.feedback = 'no';

dataspike = ft_preprocessing(filtconfig, datawide);


% Produce rectified activity signal.

filtconfigband = struct( 'bpfilter', 'yes', 'bpfilttype', 'but', ...
  'bpfreq', [ min(rect_band), max(rect_band) ] );
rectconfig = struct('rectify', 'yes');
filtconfiglow = struct( ...
  'lpfilter', 'yes', 'lpfilttype', 'but', 'lpfreq', rect_lowpass );
resampleconfig = struct( 'resamplefs', rect_rate, 'detrend', 'no' );

filtconfigband.feedback = 'no';
rectconfig.feedback = 'no';
filtconfiglow.feedback = 'no';
```

```
resampleconfig.feedback = 'no';


% FIXME - We can group some of these calls, but that requires detailed
% knowledge of the order in which FT applies preprocessing operations.
% Do them individually for safety's sake.

datarect = ft_preprocessing(filtconfigband, datawide);
datarect = ft_preprocessing(rectconfig, datarect);
datarect = ft_preprocessing(filtconfiglow, datarect);
datarect = ft_resampledata(resampleconfig, datarect);


% Done.

end


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

6.3 doPowerFiltering.m

```
function newsignal = ...
  doPowerFiltering( oldsignal, power_freq, power_modes, filter_type )

% function newsignal = ...
%   doPowerFiltering( oldsignal, power_freq, power_modes, filter_type )
%
% This calls Field Trip's processing functions to perform power line noise
% rejection on a wideband signal.
%
% "oldsignal" is the Field Trip data structure to process.
% "power_freq" is the nominal power frequency in Hz (50 or 60 Hz).
% "power_modes" is the number of frequency modes to remove (1 = fundamental,
%   2 = fundamental and first harmonic, etc).
% "filter_type" is 'fir' to use the FIR filter (FIXME - takes forever),
%   'dft' to use a DFT band-stop filter (only usable with short signals;
%   time and memory go way up for longer), 'cosine' to use a DFT
%   filter that does cosine fitting (fast but does a poor job), 'brickwall'
%   to use a hard-stop frequency domain filter (produces ringing), and
%   and 'thilo' to use Thilo's old filter setup (a comb of cosine-fit
%   filters; performance comparable to 'cosine' in my tests).
%
% "newsignal" is a Field Trip data structure containing the filtered signal.


% Switch for using LoopUtil implementations rather than FT.
```

```
want_looputil_brick = true;

% Special-case filters that bypass FT.
if want_looputil_brick && strcmp('brickwall', filter_type)
  % Brick-wall frequency-domain filter.
  notch_bw = 2.0;
  newsignal = ...
    euFT_doBrickBandStop( oldsignal, power_freq, power_modes, notch_bw );
else
  % Use Field Trip to do the filtering.

  % Check other cases.
  if strcmp('fir', filter_type)
    % FIXME - This takes a really impractical amount of time.
    filt_power = ...
      euFT_getFiltPowerFIR(power_freq, power_modes, oldsignal.fsample);
  elseif strcmp('dft', filter_type)
    % NOTE - This takes a long time.
    filt_power = euFT_getFiltPowerDFT(power_freq, power_modes);
  elseif strcmp('cosine', filter_type)
    % NOTE - This works poorly.
    filt_power = euFT_getFiltPowerCosineFit(power_freq, power_modes);
  elseif strcmp('brickwall', filter_type)
    % FIXME - This doesn't work properly! Use the LoopUtil version instead.
    filt_power = euFT_getFiltPowerBrick(power_freq, power_modes);
  elseif strcmp('thilo', filter_type)
    % NOTE - This works poorly (it's a comb of 'cosine' filters).
    filt_power = euFT_getFiltPowerTW(power_freq, power_modes);
  else
    error([ 'Unknown filter type "' filter_type '".' ]);
  end

  % Suppress progress reports.
  filt_power.feedback = 'no';

  % Call Field Trip's filtering routines.
  newsignal = ft_preprocessing(filt_power, oldsignal);
end


% Done.

end


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

6.4 doSelectGoodTrials.m

```
function newlist = doSelectGoodTrials(oldlist)

% function newlist = doSelectGoodTrials(oldlist)
%
% This accepts a list of per-trial event code sequences, identifies and
% discards "bad" trials, and returns a list containing event code sequences
% for the "good" trials.
%
% FIXME - This is a placeholder function that looks for TrialNumber
% incrementing. Use your own filtering routines for real data.
% FIXME - This always discards the final trial, since we aren't sure if
% the index incremented or not after it.
%
% "oldlist" is a cell array containing per-trial event code tables.
%
% "newlist" is a copy of "oldlist" containing only "good" trials.


% FIXME - Select trials where TrialNumber incremented.
% FIXME - This always discards the final trial, since we aren't sure if
% the index incremented or not after it.

newlist = {};
newcount = 0;
for tidx = 2:length(oldlist)
  prevtable = oldlist{tidx-1};
  thistable = oldlist{tidx};

  prevline = prevtable(strcmp(prevtable.codeLabel, 'TrialNumber'),:);
  thisline = thistable(strcmp(thistable.codeLabel, 'TrialNumber'),:);

  if (~isempty(prevline)) && (~isempty(thisline))
    if thisline.codeData ~= prevline.codeData
      % We just incremented TrialNumber.
      % This means the "previous" trial was valid.
      newcount = newcount + 1;
      newlist{newcount} = prevtable;
    end
  end
end


% Done.

end
```

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

6.5 doSignalConditioning.m

```
function newdata = doSignalConditioning( olddata, ...
  power_freq, power_modes, filter_type, extra_notches )

% function newdata = doSignalConditioning( olddata, ...
%   power_freq, power_modes, filter_type, extra_notches )
%
% This de-trends the signal, applies a power line rejection filter, and
% optionally applies the same type of notch filter to suppress additional
% narrow-band noise spikes.
%
% "olddata" is the FT dataset to process.
% "power_freq" is the power line fundamental frequency.
% "power_modes" is the number of power line modes to filter ( 1 = fundamental,
%   2 = fundamental + first harmonic, etc.).
% "filter_type" is 'fir', 'dft', 'cosine, 'brickwall', or 'thilo', per
%   doPowerfiltering().
% "extra_notches" is a vector containing any additional frequencies to
%   filter. This may be empty.


% Copy the old dataset.
newdata = olddata;

% De-trend.
newdata = ft_preprocessing( ...
  struct( 'detrend', 'yes', 'feedback', 'no' ), ...
  newdata );

% Apply the power line filter.
newdata = doPowerFiltering( newdata, power_freq, power_modes, filter_type);

% Apply the power line filter again at various extra notch frequencies.
% Only do fundamental-mode filtering for this.
for fidx = 1:length(extra_notches)
  newdata = doPowerFiltering( newdata, extra_notches(fidx), 1, filter_type );
end


% Done.

end
```

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

Chapter 7

Plotting

7.1 doBrowseFiltered.m

```
function doBrowseFiltered( titleprefix, ...
  data_wideband, data_lfp, data_spike, data_rect )

% function doBrowseFiltered( titleprefix, ...
%   data_wideband, data_lfp, data_spike, data_rect )
%
% This pops up ft_databrowser() windows for filtered waveform data.
% Windows are given titles that begin with the specified prefix.
%
% "titleprefix" is a string prepended to the window title.
% "data_wideband" is FT-processed wideband data.
% "data_lfp" is FT-processed local field potential data.
% "data_spike" is FT-processed spike-band data.
% "data_rect" is FT-processed rectified spike activity data.


browserconfig = struct( 'allowoverlap', 'yes');

ft_databrowser(browserconfig, data_wideband);
set( gcf(), 'Name', [ titleprefix ' Wideband' ], 'NumberTitle', 'off' );

ft_databrowser(browserconfig, data_lfp);
set( gcf(), 'Name', [ titleprefix ' LFP' ], 'NumberTitle', 'off' );

ft_databrowser(browserconfig, data_spike);
set( gcf(), 'Name', [ titleprefix ' Spikes' ], 'NumberTitle', 'off' );

ft_databrowser(browserconfig, data_rect);
set( gcf(), 'Name', [ titleprefix ' Rectified' ], 'NumberTitle', 'off' );
```

```
% Done.

end


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

7.2 doBrowseWave.m

```
function doBrowseWave( thisdata, thistitle )

% function doBrowseWave( thisdata, thistitle )
%
% This pops up ft_databrowser() window for a specified list of waveform.
% The window is given the specified title.
%
% "thisdata" is a FT-processed data structure.
% "thistitle" is a character array containing the title to use.


browserconfig = struct( 'allowoverlap', 'yes' );
ft_databrowser(browserconfig, thisdata);
set( gcf(), 'Name', thistitle, 'NumberTitle', 'off' );


% Done.

end


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

7.3 doPlotBatchTrials.m

```
function doPlotBatchTrials( obase, batchlabel, batchtrialtable, ...
  ft_rec_wb, ft_rec_lfp, ft_rec_spike, ft_rec_rect, ...
  ft_stim_wb, ft_stim_lfp, ft_stim_spike, ft_stim_rect, ...
  ft_gaze, events_codes, events_rwdA, events_rwdB )

% function doPlotBatchTrials( obase, batchlabel, batchtrialtable, ...
%   ft_rec_wb, ft_rec_lfp, ft_rec_spike, ft_rec_rect, ...
%   ft_stim_wb, ft_stim_lfp, ft_stim_spike, ft_stim_rect, ...
%   events_codes, events_rwdA, events_rwdB )
%
```

```
% This generates stacked plots for a batch of trials.
%
% NOTE - "Relative" gaze signals are boosted 1000x to show up when plotted
% with absolute gaze signals.
%
% "obase" is the prefix to use when building filenames.
% "batchlabel" is a filename-safe string identifying this batch of trials.
% "batchtrialtable" is a table providing trial definitions for the trials in
%   this batch. Relevant columns are "trialindex", "trialnum",
%   "rectimestart", "rectimeend", and "rectimeevent".
% "ft_rec_wb" is a Field Trip raw data structure with wideband waveforms from
%   the ephys recorder.
% "ft_rec_lfp" is a Field Trip raw data structure with LFP waveforms from
%   the ephys recorder.
% "ft_rec_spike" is a Field Trip raw data structure with high-pass waveforms
%   from the ephys recorder.
% "ft_rec_rect" is a Field Trip raw data structure with rectified activity
%   waveforms from the ephys recorder.
% "ft_stim_wb" is a Field Trip raw data structure with wideband waveforms from
%   the ephys stimulator.
% "ft_stim_lfp" is a Field Trip raw data structure with LFP waveforms from
%   the ephys stimulator.
% "ft_stim_spike" is a Field Trip raw data structure with high-pass waveforms
%   from the ephys stimulator.
% "ft_stim_rect" is a Field Trip raw data structure with rectified activity
%   waveforms from the ephys stimulator.
% "ft_gaze" is a Field Trip raw data structure with gaze waveforms.
% "events_codes" is a table containing event codes for each trial's span.
% "events_rwdA" is a table containing "reward A" events for each trial's span.
% "events_rwdB" is a table containing "reward B" events for each trial's span.


% FIXME - Hardcoding Y range to fit the datasets we have.
ymax_wb = 600;
ymax_lfp = 200;
ymax_hp = 300;
ymax_rect = 50;

ymax_gaze = 1500;
% We need to amplify "relative" gaze signals to get a compatible plot scale.
gaze_scale_relative = 1000;


% FIXME - Hardcoding time ranges to get readable plots.

rangeclose = [ -0.2 0.4 ];
rangedetail = [ -0.1 0.1 ];
rangefine = [ -0.01 0.01 ];
rangegaze = [ -0.5 1.0 ];
```

```
% For wideband, we just want "wide" and "detail".
% Most of what we're seeing is noise, and the fact that spikes exist.
timeranges = struct( 'wide', 'auto', 'detail', rangedetail );

helper_plotStack( [ obase '-rec-wb' ], ...
  sprintf( 'Trials - %s - Rec Wideband', batchlabel ), ...
  batchtrialtable, ymax_wb, timeranges, ...
  ft_rec_wb, events_codes, events_rwdA, events_rwdB );

helper_plotStack( [ obase '-stim-wb' ], ...
  sprintf( 'Trials - %s - Stim Wideband', batchlabel ), ...
  batchtrialtable, ymax_wb, timeranges, ...
  ft_stim_wb, events_codes, events_rwdA, events_rwdB );


% For LFP and rectified activity, add "close".
% This is a wide enough range that we can see response to cues.

timeranges.('close') = rangeclose;

helper_plotStack( [ obase '-rec-lfp' ], ...
  sprintf( 'Trials - %s - Rec LFP', batchlabel ), ...
  batchtrialtable, ymax_lfp, timeranges, ...
  ft_rec_lfp, events_codes, events_rwdA, events_rwdB );

helper_plotStack( [ obase '-stim-lfp' ], ...
  sprintf( 'Trials - %s - Stim LFP', batchlabel ), ...
  batchtrialtable, ymax_lfp, timeranges, ...
  ft_stim_lfp, events_codes, events_rwdA, events_rwdB );

helper_plotStack( [ obase '-rec-rect' ], ...
  sprintf( 'Trials - %s - Rec Activity', batchlabel ), ...
  batchtrialtable, ymax_rect, timeranges, ...
  ft_rec_rect, events_codes, events_rwdA, events_rwdB );

helper_plotStack( [ obase '-stim-rect' ], ...
  sprintf( 'Trials - %s - Stim Activity', batchlabel ), ...
  batchtrialtable, ymax_rect, timeranges, ...
  ft_stim_rect, events_codes, events_rwdA, events_rwdB );


% For spikes, add "fine", so that we can see spike waveforms.

timeranges.('fine') = rangefine;

helper_plotStack( [ obase '-rec-hp' ], ...
  sprintf( 'Trials - %s - Rec High-Pass', batchlabel ), ...
  batchtrialtable, ymax_hp, timeranges, ...
  ft_rec_spike, events_codes, events_rwdA, events_rwdB );
```

```
helper_plotStack( [ obase '-stim-hp' ], ...
  sprintf( 'Trials - %s - Stim High-Pass', batchlabel ), ...
  batchtrialtable, ymax_hp, timeranges, ...
  ft_stim_spike, events_codes, events_rwdA, events_rwdB );


% For gaze, use a wider version of "glose".
% Even "wide" isn't very readable.

timeranges = struct( 'wide', 'auto', 'close', rangegaze );

helper_plotGaze( [ obase '-gaze' ], ...
  sprintf( 'Trials - %s - Gaze', batchlabel ), ...
  batchtrialtable, ymax_gaze, gaze_scale_relative, timeranges, ...
  ft_gaze, events_codes, events_rwdA, events_rwdB );


% Done.

end


%
% Helper functions.


% This plots sets of stacked trial waveforms for a given filter case.
% FIXME - This hard-codes a lot of fragile appearance information.

function helper_plotStack( ...
  fbase, figtitle, batchdefs, maxyval, timeranges, ...
  ft_wavedata, events_codes, events_rwdA, events_rwdB )

  %
  % Extract selected metadata.

  trialcount = height(batchdefs);
  firsttimes = [];
  lasttimes = [];
  reftimes = [];
  if ~isempty(batchdefs)
    firsttimes = batchdefs.rectimestart;
    lasttimes = batchdefs.rectimeend;
    reftimes = batchdefs.rectimeevent;
  end

  waverate = 1000;
  wavechans = 0;
```

```
  if ~isempty(ft_wavedata)
    waverate = ft_wavedata.fsample;
    wavechans = length(ft_wavedata.label);
  end

  timelabels = fieldnames(timeranges);

  plotyrange = [ -maxyval maxyval ];


  %
  % Render the figures and save them.

  thisfig = figure();

  % One output set of figures per channel.
  for cidx = 1:wavechans

    % We only need to plot the figure once; we'll call "xlim" for each
    % zoom level.

    figure(thisfig);
    clf('reset');

    thislabelraw = ft_wavedata.label{cidx};
    thislabel = strrep(thislabelraw, '_', ' ');
    title(sprintf( '%s - %s', figtitle, thislabel ));

    ylim( plotyrange );
    xlabel('Time (s)');
    ylabel('Amplitude (a.u.)');

    hold on;

    helper_renderStack( ...
      trialcount, cidx, firsttimes, reftimes, waverate, plotyrange, ...
      ft_wavedata, events_codes, events_rwdA, events_rwdB )

    hold off;

    % Iterate zoom ranges, saving one figure per zoom level.
    for zidx = 1:length(timelabels)
      thiszoom = timelabels{zidx};

      xlim( timeranges.(thiszoom) );

      % NOTE - Use the channel label, not the index, for readability.
%      saveas(thisfig, sprintf('%s-%s-ch%04d.png', fbase, thiszoom, cidx));
      saveas(thisfig, sprintf('%s-%s-%s.png', fbase, thiszoom, thislabelraw));
    end
```

```
  end

  % Reset before closing, just in case of memory leaks.
  figure(thisfig);
  clf('reset');

  close(thisfig);

end


% This plots a series of stacked trial waveforms in the current axes.

function helper_renderStack( ...
  trialcount, chanidx, firsttimes, reftimes, waverate, cursorrange, ...
  ft_wavedata, events_codes, events_rwdA, events_rwdB )

  % Build a decent colour palette.
  % This isn't expensive, just do it here to avoid duplication.

  % We need to be able to distinguish each _type_ of information as well as
  % trials within each type. We probably won't be able to do the latter,
  % but try.

  cols = nlPlot_getColorPalette();

  palette_waves = nlPlot_getColorSpread(cols.grn, trialcount, 180);

  % These are cursors, so making them visually distinct from each other is
  % more important than making them visually distinct from the waveforms.
  % There are a lot of event codes per frame, so make them a fainter colour.
  % FIXME - Getting this to look not-ugly involves a lot of hand-tweaking.
  palette_codes = nlPlot_getColorSpread(cols.cyn, trialcount, 20);
  palette_rwdA = nlPlot_getColorSpread(cols.brn, trialcount, 30);
  palette_rwdB = nlPlot_getColorSpread(cols.mag, trialcount, 60);


  % Render event cursors first, so that they're behind the waves.

  for tidx = 1:trialcount
    thistable = events_codes{tidx};
    for eidx = 1:height(thistable)
      thistime = thistable.recTime(eidx) - reftimes(tidx);
      plot( [ thistime thistime ], cursorrange, ...
        'Color', palette_codes{tidx}, ...
        'DisplayName', sprintf('trial %d codes', tidx) );
    end
  end

  for tidx = 1:trialcount
```

```
    thistable = events_rwdA{tidx};
    for eidx = 1:height(thistable)
      thistime = thistable.recTime(eidx) - reftimes(tidx);
      plot( [ thistime thistime ], cursorrange, ...
        'Color', palette_rwdA{tidx}, ...
        'DisplayName', sprintf('trial %d rwdA', tidx) );
    end
  end

  for tidx = 1:trialcount
    thistable = events_rwdB{tidx};
    for eidx = 1:height(thistable)
      thistime = thistable.recTime(eidx) - reftimes(tidx);
      plot( [ thistime thistime ], cursorrange, ...
        'Color', palette_rwdB{tidx}, ...
        'DisplayName', sprintf('trial %d rwdB', tidx) );
    end
  end

  % Now render the waves.

  for tidx = 1:trialcount
    thiswave = ft_wavedata.trial{tidx}(chanidx,:);
    sampcount = length(thiswave);
    thistimes = 0:(sampcount-1);
    thistimes = (thistimes / waverate) + firsttimes(tidx);
    thistimes = thistimes - reftimes(tidx);

    plot( thistimes, thiswave, 'Color', palette_waves{tidx}, ...
      'DisplayName', sprintf('trial %d', tidx) );
  end

  % Finished rendering this stack.
end


% This plots sets of stacked gaze waveforms.
% FIXME - This hard-codes a lot of fragile appearance information.

function helper_plotGaze( ...
  fbase, figtitle, batchdefs, maxyval, relativescale, timeranges, ...
  ft_wavedata, events_codes, events_rwdA, events_rwdB )

  %
  % Extract selected metadata.

  trialcount = height(batchdefs);
  firsttimes = [];
  lasttimes = [];
  reftimes = [];
```

```
if ~isempty(batchdefs)
  firsttimes = batchdefs.rectimestart;
  lasttimes = batchdefs.rectimeend;
  reftimes = batchdefs.rectimeevent;
end

waverate = 1000;
wavechans = 0;

if ~isempty(ft_wavedata)
  waverate = ft_wavedata.fsample;
  wavechans = length(ft_wavedata.label);
end

timelabels = fieldnames(timeranges);

% NOTE - Make per-channel y ranges.
% Anything with "relative" in the name gets a smaller scale.

plotyrange = {};

for cidx = 1:wavechans
  thisyrange = [ -maxyval maxyval ];

  thislabel = ft_wavedata.label{cidx};
  if contains(thislabel, 'relative', 'IgnoreCase', true)
    thisyrange = thisyrange / relativescale;
  end

  plotyrange{cidx} = thisyrange;
end



%
% Render the figures and save them.

thisfig = figure();

%
% We're stacking channels in subplots, since there are only four of them.

% We only need to plot the figure once; we'll call "xlim" for each
% zoom level.

figure(thisfig);
clf('reset');

for cidx = 1:wavechans
  subplot(wavechans, 1, cidx);
```

```
    thislabelraw = ft_wavedata.label{cidx};
    thislabel = strrep(thislabelraw, '_', ' ');
    title(sprintf( '%s - %s', figtitle, thislabel ));

    ylim( plotyrange{cidx} );
    xlabel('Time (s)');
    ylabel('Amplitude (a.u.)');

    hold on;

    helper_renderStack( ...
      trialcount, cidx, firsttimes, reftimes, waverate, plotyrange{cidx}, ...
      ft_wavedata, events_codes, events_rwdA, events_rwdB )

    hold off;
  end

  % Iterate zoom ranges, saving one figure per zoom level.
  % We can walk through the subplots nondestructively.

  for zidx = 1:length(timelabels)
    thiszoom = timelabels{zidx};

    for cidx = 1:wavechans
      subplot(wavechans, 1, cidx);
      xlim( timeranges.(thiszoom) );
    end

    % We're stacking channels, so omit channel from the filename.
    saveas(thisfig, sprintf('%s-%s.png', fbase, thiszoom));
  end

  % Reset before closing, just in case of memory leaks.
  figure(thisfig);
  clf('reset');

  close(thisfig);

end


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

7.4 doPlotStackedSpectra.m

```
function doPlotStackedSpectra( fname, spectmags, spectfreqs, ...
  scaletype, cursorfreqs, fitmags, fitfreqs, chanlabels, figtitle )
```

```
% function doPlotStackedSpectra( fname, spectmags, spectfreqs, ...
%   scaletype, cursorfreqs, fitmags, fitfreqs, chanlabels, figtitle )
%
% This plots a series of per-channel stacked spectra. These are magnitude
% spectra, not power spectra. The spectra are optionally annotated with
% cursors and with fitted curves.
%
% "fname" is the name of the file to save the plot to.
% "spectmags" is a cell array with per-channel spectrum magnitude vectors.
% "spectfreqs" is a cell array with per-channel spectrum frequency vectors.
% "scaletype" is 'magnitude' or 'power'.
% "cursorfreqs" is a cell array with per-channel cursor frequency vectors.
% "fitmags" is a cell array with per-channel fitted curve magnitude vectors.
% "fitfreqs" is a cell array with per-channel fitted curve frequency vectors.
% "chanlabels" is a cell array containing channel labels.
% "figtitle" is a character array to use as the figure title.
%
% Empty cell arrays can be passed as "cursorfreqs", "fitmags", or "fitfreqs"
% to suppress cursor or curve-fit output.


% Get metadata.
chancount = length(chanlabels);


% FIXME - Strip special characters from the channel labels.
for cidx = 1:chancount
  chanlabels{cidx} = strrep(chanlabels{cidx}, '_', ' ');
end


% Make a readable palette.

cols = nlPlot_getColorPalette();
palette_spectra = nlPlot_getColorSpread(cols.grn, chancount, 180);
palette_cursors = nlPlot_getColorSpread(cols.mag, chancount, 60);
palette_fits = nlPlot_getColorSpread(cols.brn, chancount, 60);


% FIXME - Pick a sane Y scale.

ymagmax = 0;
ymagmin = 0;
for cidx = 1:chancount
  thismax = max(spectmags{cidx});
  ymagmax = max(ymagmax, thismax);
end
ymagmax = 10^ceil(log10(ymagmax));
```

```
% FIXME - A factor of 1000 is fine for the tungsten test data but not the
% silicon one.
%yrangescale = 1e+3;
yrangescale = 1e+4;
if strcmp('power', scaletype)
  yrangescale = yrangescale * yrangescale;
end

ymagmin = ymagmax / yrangescale;


% Initialize the figure.

thisfig = figure();
clf('reset');

title(figtitle);
xlabel('Frequency (Hz)');

if strcmp('power', scaletype)
  ylabel('Power (a.u.)');
else
  ylabel('Magnitude (a.u.)');
end

set(gca, 'Xscale', 'log');
set(gca, 'Yscale', 'log');

ylim( [ymagmin ymagmax] );

hold on;


% Render cursors, followed by spectra, followed by curve fits.

if ~isempty(cursorfreqs)
  for cidx = 1:chancount
    thiscursorlist = cursorfreqs{cidx};

    for lidx = 1:length(thiscursorlist)
      thisfreq = thiscursorlist(lidx);

      if lidx == 1
        plot( [thisfreq thisfreq], [ymagmin ymagmax], ...
          'Color', palette_cursors{cidx}, 'DisplayName', chanlabels{cidx} );
      else
        plot( [thisfreq thisfreq], [ymagmin ymagmax], ...
          'Color', palette_cursors{cidx}, 'HandleVisibility', 'off' );
      end
    end
```

```
  end
end

for cidx = 1:chancount
  plot( spectfreqs{cidx}, spectmags{cidx}, ...
    'Color', palette_spectra{cidx}, 'DisplayName', chanlabels{cidx} );
end

if (~isempty(fitmags)) && (~isempty(fitfreqs))
  for cidx = 1:chancount
    thismaglist = fitmags{cidx};
    thisfreqlist = fitfreqs{cidx};

    if (~isempty(thismaglist)) && (~isempty(thisfreqlist))
      plot( thisfreqlist, thismaglist, ...
        'Color', palette_fits{cidx}, 'DisplayName', chanlabels{cidx} );
    end
  end
end


hold off;

saveas(thisfig, fname);

% Reset before closing, just in case of memory leaks.
figure(thisfig);
clf('reset');

close(thisfig);


% Done.

end


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

Chapter 8

Signal Quality Checking

8.1 do_test_autoclassify_chans.m

```
% Field Trip sample script / test script - Automatic channel classification.
% Written by Christopher Thomas.

% This reads a small section of the analog data, performs signal processing,
% and attempts to determine which channels contain valid data.
% FIXME - Doing this by reading and setting workspace variables directly.
%
% Variables that get set:
%
%   rec_bits
%   stim_bits
%   rec_quantized
%   stim_quantized
%
%   rec_has_dropouts
%   stim_has_dropouts
%   rec_dropout_frac
%   stim_dropout_frac
%
%   rec_has_artifacts
%   stim_has_artifacts
%   rec_artifact_frac
%   stim_artifact_frac
%
%   rec_has_peaks_raw
%   rec_peakfreqs_raw
%   rec_peakheights_raw
%   rec_peakwidths_raw
%   rec_has_peaks_filt
%   rec_peakfreqs_filt
%   rec_peakheights_filt
```

```
%   rec_peakwidths_filt
%
%   stim_has_peaks_raw
%   stim_peakfreqs_raw
%   stim_peakheights_raw
%   stim_peakwidths_raw
%   stim_has_peaks_filt
%   stim_peakfreqs_filt
%   stim_peakheights_filt
%   stim_peakwidths_filt
%
%   rec_peakspectmags_raw
%   rec_peakspectfreqs_raw
%   rec_peakspectmags_filt
%   rec_peakspectfreqs_filt
%
%   stim_peakspectmags_raw
%   stim_peakspectfreqs_raw
%   stim_peakspectmags_filt
%   stim_peakspectfreqs_filt
%
%   rec_lfpgood
%   rec_lfptype
%   rec_lfpexponent
%   rec_lfpspectpowers
%   rec_lfpspectfreqs
%   rec_lfpfitpowers
%
%   stim_lfpgood
%   stim_lfptype
%   stim_lfpexponent
%   stim_lfpspectpowers
%   stim_lfpspectfreqs
%   stim_lfpfitpowers
%
%   rec_correl
%   stim_correl



%
% Load cached results from disk, if requested.
% If we successfully load data, bail out without further processing.

if want_cache_autoclassify
  fname = [ datadir filesep 'autoclassify.mat' ];

  if isfile(fname)

    % Load the data we previously saved.
```

```
    disp('-- Loading channel auto-classification results.');
    load(fname);

    % Generate reports.

    thismsg = helper_reportQuantized( ...
      [ plotdir filesep 'autodetect-quantization.txt' ], ...
      rec_quantized, stim_quantized, ...
      rec_channels_ephys, stim_channels_ephys );
    disp(thismsg);

    thismsg = helper_reportDropoutArtifact( ...
      [ plotdir filesep 'autodetect-dropouts-artifacts.txt' ], ...
      rec_has_artifacts, stim_has_artifacts, ...
      rec_artifact_frac, stim_artifact_frac, ...
      rec_has_dropouts, stim_has_dropouts, ...
      rec_dropout_frac, stim_dropout_frac, ...
      rec_channels_ephys, stim_channels_ephys );
    disp(thismsg);

    % Banner.
    disp('-- Finished loading.');

    % We've loaded cached results. Bail out of this portion of the script.
    return;

  end
end



%
% Banner.

disp('-- Attempting to auto-classify channels.');



%
% Read the analog signals using ft_preprocessing().


% Select the auto-classification time window (short).
% Also read in native format, not double; this lets us catch quantization.

have_native = false;

preproc_config_rec_auto = preproc_config_rec;
preproc_config_stim_auto = preproc_config_stim;
preproc_config_rec_auto.trl = preproc_config_rec_span_autotype;
```

```
preproc_config_stim_auto.trl = preproc_config_stim_span_autotype;
if thisdataset.use_looputil
  have_native = true;
  preproc_config_rec_auto.dataformat = 'nlFT_readDataNative';
  preproc_config_stim_auto.dataformat = 'nlFT_readDataNative';
end

preproc_config_rec_auto.feedback = 'no';
preproc_config_stim_auto.feedback = 'no';


% Read the data.

% NOTE - Field Trip will throw an exception if this fails. Wrap this to
% catch exceptions.

have_recdata_auto = false;
have_stimdata_auto = false;

try

  disp('-- Reading windowed ephys amplifier data.');
  tic();

  % Report the window span.
  disp(sprintf( ...
    '.. Read window is:   %.1f - %.1f s (rec)   %.1f - %.1f s (stim).', ...
    preproc_config_rec_span_autotype(1) / rechdr.Fs, ...
    preproc_config_rec_span_autotype(2) / rechdr.Fs, ...
    preproc_config_stim_span_autotype(1) / stimhdr.Fs, ...
    preproc_config_stim_span_autotype(2) / stimhdr.Fs ));

  if isempty(rec_channels_ephys)
    disp('.. Skipping recorder (no channels selected).');
  else
    preproc_config_rec_auto.channel = rec_channels_ephys;
    recdata_auto = ft_preprocessing(preproc_config_rec_auto);
    have_recdata_auto = true;
  end

  if isempty(stim_channels_ephys)
    disp('.. Skipping stimulator (no channels selected).');
  else
    preproc_config_stim_auto.channel = stim_channels_ephys;
    stimdata_auto = ft_preprocessing(preproc_config_stim_auto);
    have_stimdata_auto = true;
  end

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. Read in %s.', thisduration ));
```

```
  % Done.
  disp('-- Finished reading data.');


catch errordetails
  disp(sprintf( ...
    '###  Exception thrown while reading "%s".', thisdataset.title));
  disp(sprintf('Message: "%s"', errordetails.message));

  % Abort the script and send the user back to the Matlab prompt.
  error('Couldn''t read ephys waveform data; bailing out.');
end



%
% Check for quantization.

% We have to do this before filtering.

nchans_rec = length(rec_channels_ephys);
nchans_stim = length(stim_channels_ephys);

rec_bits = zeros(nchans_rec, 1);
stim_bits = zeros(nchans_stim, 1);
rec_quantized = zeros(nchans_rec, 1, 'logical');
stim_quantized = zeros(nchans_stim, 1, 'logical');

if ~have_native
  disp('-- Don''t have native data; skipping quantization check.');
else

  disp('-- Checking for quantization.');

  if nchans_rec > 0
    thisbits = nlCheck_getFTSignalBits(recdata_auto);
    rec_bits = thisbits{1};
    rec_quantized = (rec_bits <= quantization_bits);
  end

  if nchans_stim > 0
    thisbits = nlCheck_getFTSignalBits(stimdata_auto);
    stim_bits = thisbits{1};
    stim_quantized = (stim_bits <= quantization_bits);
  end


  % Quantization report.
```

```
  thismsg = helper_reportQuantized( ...
    [ plotdir filesep 'autodetect-quantization.txt' ], ...
    rec_quantized, stim_quantized, ...
    rec_channels_ephys, stim_channels_ephys );

  disp(thismsg);


  % Done.
  disp('-- Finished checking for quantization.');

end



%
% Filter the continuous ephys data.
% This will have edge effects, but that should be tolerable.

% NOTE - Handling recorder and stimulator data separately, for simplicity.

% NOTE - We're assuming that the window is short enough for de-trending to
% be appropriate (i.e. that it's ramped, not randomly wandering).


% NOTE - We're keeping the unfiltered signal as "auto" and the power filtered
% signal as "wideband", to measure how well the power filter is working.


% Banner.
disp('-- Filtering windowed ephys data.');


if have_recdata_auto

  %
  % De-trending and power-line filtering.

  disp('.. [Rec] De-trending and removing power-line noise.');
  tic();

  % Save the filtered signal as "wideband". Keep the unfiltered around too.

  extra_notches = [];
  if isfield( thisdataset, 'extra_notches' )
    extra_notches = thisdataset.extra_notches;
  end

  recdata_wideband = doSignalConditioning( recdata_auto, ...
    power_freq, power_filter_modes, filter_type_long, ...
```

```
    extra_notches );

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. [Rec] Power line noise removed in %s.', thisduration ));


  %
  % NOTE - Not removing artifacts; we're _looking_ for artifacts.


  %
  % Get spike and LFP and rectified waveforms.

  % We already have the wideband signal.

  disp('.. [Rec] Generating LFP, spike, and rectified activity data series.');
  tic();

  [ recdata_lfp recdata_spike recdata_rect ] = ...
    doFeatureFiltering( recdata_wideband, ...
      lfp_corner, lfp_rate, spike_corner, ...
      rect_corners, rect_lowpass, rect_rate );

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. [Rec] Filtered series generated in %s.', thisduration ));


  %
  % NOTE - Not rereferencing. We don't know which channels are good yet.
  % We also don't know which channels should and shouldn't be averaged, here.


  % Done.

end


if have_stimdata_auto

  %
  % De-trending and power-line filtering.

  disp('.. [Stim] De-trending and removing power-line noise.');
  tic();

  % Save the filtered signal as "wideband". Keep the unfiltered around too.

  extra_notches = [];
  if isfield( thisdataset, 'extra_notches' )
    extra_notches = thisdataset.extra_notches;
```

```
  end

  stimdata_wideband = doSignalConditioning( stimdata_auto, ...
    power_freq, power_filter_modes, filter_type_long, ...
    extra_notches );

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. [Stim] Power line noise removed in %s.', thisduration ));


  %
  % NOTE - Not removing artifacts; we're _looking_ for artifacts.


  %
  % Get spike and LFP and rectified waveforms.

  % We already have the wideband signal.

  disp('.. [Stim] Generating LFP, spike, and rectified activity data series.');
  tic();

  [ stimdata_lfp stimdata_spike stimdata_rect ] = ...
    doFeatureFiltering( stimdata_wideband, ...
      lfp_corner, lfp_rate, spike_corner, ...
      rect_corners, rect_lowpass, rect_rate );

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. [Stim] Filtered series generated in %s.', thisduration ));


  %
  % NOTE - Not rereferencing. We don't know which channels are good yet.
  % We also don't know which channels should and shouldn't be averaged, here.


  % Done.

end


% Done.
disp('-- Finished filtering windowed ephys data.');



%
% Check for artifacts and dropouts.

rec_has_artifacts = zeros(nchans_rec, 1, 'logical');
```

```
rec_has_dropouts = zeros(nchans_rec, 1, 'logical');
stim_has_artifacts = zeros(nchans_stim, 1, 'logical');
stim_has_dropouts = zeros(nchans_stim, 1, 'logical');

rec_artifact_frac = zeros(nchans_rec, 1, 'double');
rec_dropout_frac = zeros(nchans_rec, 1, 'double');
stim_artifact_frac = zeros(nchans_stim, 1, 'double');
stim_dropout_frac = zeros(nchans_stim, 1, 'double');


disp('-- Checking for dropouts and artifacts.');


smoothfreq = 1.0 / artifact_dropout_time;
filtconfig_smooth = ...
  struct( 'lpfilter', 'yes', 'lpfilttype', 'but', 'lpfreq', smoothfreq );
filtconfig_smooth.feedback = 'no';

if have_recdata_auto
  % NOTE - This gets perturbed by low-frequency noise.
  [ this_dropout, this_artifact ] = ...
    nlCheck_testFTDropoutsArtifacts( recdata_wideband, smoothfreq, ...
      dropout_rect_threshold, artifact_rect_threshold );

  rec_dropout_frac = this_dropout{1};
  rec_artifact_frac = this_artifact{1};

  rec_has_artifacts = (rec_artifact_frac >= artifact_bad_frac);
  rec_has_dropouts = (rec_dropout_frac >= dropout_bad_frac);
end

if have_stimdata_auto
  % NOTE - This gets perturbed by low-frequency noise.
  [ this_dropout, this_artifact ] = ...
    nlCheck_testFTDropoutsArtifacts( stimdata_wideband, smoothfreq, ...
      dropout_rect_threshold, artifact_rect_threshold );

  stim_dropout_frac = this_dropout{1};
  stim_artifact_frac = this_artifact{1};

  stim_has_artifacts = (stim_artifact_frac >= artifact_bad_frac);
  stim_has_dropouts = (stim_dropout_frac >= dropout_bad_frac);
end


% Dropout and artifact report.

thismsg = helper_reportDropoutArtifact( ...
  [ plotdir filesep 'autodetect-dropouts-artifacts.txt' ], ...
  rec_has_artifacts, stim_has_artifacts, ...
```

```
  rec_artifact_frac, stim_artifact_frac, ...
  rec_has_dropouts, stim_has_dropouts, ...
  rec_dropout_frac, stim_dropout_frac, ...
  rec_channels_ephys, stim_channels_ephys );

disp(thismsg);


disp('-- Finished checking for dropouts and artifacts.');



%
% Check power spectra.


% Power spectrum peaks.

disp('-- Checking for narrow-band noise.');


rec_peakfreqs_raw = {};
rec_peakheights_raw = {};
rec_peakwidths_raw = {};

rec_peakfreqs_filt = {};
rec_peakheights_filt = {};
rec_peakwidths_filt = {};

rec_has_peaks_raw = logical([]);
rec_has_peaks_filt = logical([]);

rec_peakspectmags_raw = {};
rec_peakspectfreqs_raw = {};
rec_peakspectmags_filt = {};
rec_peakspectfreqs_filt = {};


if have_recdata_auto

  for cidx = 1:nchans_rec
    thisdata_raw = recdata_auto.trial{1}(cidx,:);
    rawrate = recdata_auto.fsample;

    thisdata_filt = recdata_wideband.trial{1}(cidx,:);
    filtrate = recdata_wideband.fsample;


    [ peakfreqs peakheights peakwidths spectmags spectfreqs ] = ...
      nlProc_findSpectrumPeaks( thisdata_raw, rawrate, ...
```

```
        noisepeakwidth, noisebackgroundwidth, noisepeakthreshold );

    rec_peakfreqs_raw{cidx} = peakfreqs;
    rec_peakheights_raw{cidx} = peakheights;
    rec_peakwidths_raw{cidx} = peakwidths;

    rec_peakspectmags_raw{cidx} = spectmags;
    rec_peakspectfreqs_raw{cidx} = spectfreqs;

    rec_has_peaks_raw(cidx) = ~isempty(peakfreqs);


    [ peakfreqs peakheights peakwidths spectmags spectfreqs ] = ...
      nlProc_findSpectrumPeaks( thisdata_filt, filtrate, ...
        noisepeakwidth, noisebackgroundwidth, noisepeakthreshold );

    rec_peakfreqs_filt{cidx} = peakfreqs;
    rec_peakheights_filt{cidx} = peakheights;
    rec_peakwidths_filt{cidx} = peakwidths;

    rec_peakspectmags_filt{cidx} = spectmags;
    rec_peakspectfreqs_filt{cidx} = spectfreqs;

    rec_has_peaks_filt(cidx) = ~isempty(peakfreqs);
  end

end


stim_peakfreqs_raw = {};
stim_peakheights_raw = {};
stim_peakwidths_raw = {};

stim_peakfreqs_filt = {};
stim_peakheights_filt = {};
stim_peakwidths_filt = {};

stim_has_peaks_raw = logical([]);
stim_has_peaks_filt = logical([]);

stim_peakspectmags_raw = {};
stim_peakspectfreqs_raw = {};
stim_peakspectmags_filt = {};
stim_peakspectfreqs_filt = {};


if have_stimdata_auto

  for cidx = 1:nchans_stim
    thisdata_raw = stimdata_auto.trial{1}(cidx,:);
```

```
    rawrate = stimdata_auto.fsample;

    thisdata_filt = stimdata_wideband.trial{1}(cidx,:);
    filtrate = stimdata_wideband.fsample;


    [ peakfreqs peakheights peakwidths spectmags spectfreqs ] = ...
      nlProc_findSpectrumPeaks( thisdata_raw, rawrate, ...
        noisepeakwidth, noisebackgroundwidth, noisepeakthreshold );

    stim_peakfreqs_raw{cidx} = peakfreqs;
    stim_peakheights_raw{cidx} = peakheights;
    stim_peakwidths_raw{cidx} = peakwidths;

    stim_peakspectmags_raw{cidx} = spectmags;
    stim_peakspectfreqs_raw{cidx} = spectfreqs;

    stim_has_peaks_raw(cidx) = ~isempty(peakfreqs);


    [ peakfreqs peakheights peakwidths spectmags spectfreqs ] = ...
      nlProc_findSpectrumPeaks( thisdata_filt, filtrate, ...
        noisepeakwidth, noisebackgroundwidth, noisepeakthreshold );

    stim_peakfreqs_filt{cidx} = peakfreqs;
    stim_peakheights_filt{cidx} = peakheights;
    stim_peakwidths_filt{cidx} = peakwidths;

    stim_peakspectmags_filt{cidx} = spectmags;
    stim_peakspectfreqs_filt{cidx} = spectfreqs;

    stim_has_peaks_filt(cidx) = ~isempty(peakfreqs);
  end

end


% Narrow-band noise report.

thismsg = helper_reportNarrowBandNoise( ...
  [ plotdir filesep 'autodetect-narrownoise-raw.txt' ], ...
  rec_has_peaks_raw, stim_has_peaks_raw, ...
  rec_peakfreqs_raw, stim_peakfreqs_raw, ...
  rec_peakheights_raw, stim_peakheights_raw, ...
  rec_peakwidths_raw, stim_peakwidths_raw, ...
  rec_channels_ephys, stim_channels_ephys );

% FIXME - Diagnostics.
if true
  disp('.. Before notch filtering:');
```

```
  disp(thismsg);
end

disp('.. After notch filtering:');
thismsg = helper_reportNarrowBandNoise( ...
  [ plotdir filesep 'autodetect-narrownoise.txt' ], ...
  rec_has_peaks_filt, stim_has_peaks_filt, ...
  rec_peakfreqs_filt, stim_peakfreqs_filt, ...
  rec_peakheights_filt, stim_peakheights_filt, ...
  rec_peakwidths_filt, stim_peakwidths_filt, ...
  rec_channels_ephys, stim_channels_ephys );

disp(thismsg);


if want_plots
  doPlotStackedSpectra( [ plotdir filesep 'noise-narrow-rec-raw.png' ], ...
    rec_peakspectmags_raw, rec_peakspectfreqs_raw, ...
    'magnitude', rec_peakfreqs_raw, {}, {}, ...
    rec_channels_ephys, 'Recorder Narrow-Band Noise (raw)' );

  doPlotStackedSpectra( [ plotdir filesep 'noise-narrow-rec-filt.png' ], ...
    rec_peakspectmags_filt, rec_peakspectfreqs_filt, ...
    'magnitude', rec_peakfreqs_filt, {}, {}, ...
    rec_channels_ephys, 'Recorder Narrow-Band Noise (filtered)' );

  doPlotStackedSpectra( [ plotdir filesep 'noise-narrow-stim-raw.png' ], ...
    stim_peakspectmags_raw, stim_peakspectfreqs_raw, ...
    'mangitude', stim_peakfreqs_raw, {}, {}, ...
    stim_channels_ephys, 'Stimulator Narrow-Band Noise (raw)' );

  doPlotStackedSpectra( [ plotdir filesep 'noise-narrow-stim-filt.png' ], ...
    stim_peakspectmags_filt, stim_peakspectfreqs_filt, ...
    'magnitude', stim_peakfreqs_filt, {}, {}, ...
    stim_channels_ephys, 'Stimulator Narrow-Band Noise (filtered)' );
end


disp('-- Finished checking for narrow-band noise.');


% Power spectrum low-frequency shape.

disp('-- Checking LFP noise spectrum shape.');


rec_lfpgood = logical([]);
rec_lfptype = {};
rec_lfpexponent = [];
rec_lfpspectpowers = {};
```

```
rec_lfpspectfreqs = {};
rec_lfpfitpowers = {};

if have_recdata_auto
  for cidx = 1:nchans_rec
    thisdata_lfp = recdata_lfp.trial{1}(cidx,:);
    samprate = recdata_lfp.fsample;

    [ is_lfp typelabel fitexponent spectpowers spectfreqs fitpowers] = ...
      nlProc_examineLFPSpectrum( thisdata_lfp, samprate, ...
        lfpspectrange, lfpbinwidth );

    rec_lfpgood(cidx) = is_lfp;
    rec_lfptype{cidx} = typelabel;
    rec_lfpexponent(cidx) = fitexponent;

    rec_lfpspectpowers{cidx} = spectpowers;
    rec_lfpspectfreqs{cidx} = spectfreqs;
    rec_lfpfitpowers{cidx} = fitpowers;
  end
end

stim_lfpgood = logical([]);
stim_lfptype = {};
stim_lfpexponent = [];
stim_lfpspectpowers = {};
stim_lfpspectfreqs = {};
stim_lfpfitpowers = {};

if have_stimdata_auto
  for cidx = 1:nchans_stim
    thisdata_lfp = stimdata_lfp.trial{1}(cidx,:);
    samprate = stimdata_lfp.fsample;

    [ is_lfp typelabel fitexponent spectpowers spectfreqs fitpowers] = ...
      nlProc_examineLFPSpectrum( thisdata_lfp, samprate, ...
        lfpspectrange, lfpbinwidth );

    stim_lfpgood(cidx) = is_lfp;
    stim_lfptype{cidx} = typelabel;
    stim_lfpexponent(cidx) = fitexponent;

    stim_lfpspectpowers{cidx} = spectpowers;
    stim_lfpspectfreqs{cidx} = spectfreqs;
    stim_lfpfitpowers{cidx} = fitpowers;
  end
end


% LFP spectrum report.
```

```
thismsg = helper_reportLFPShape( ...
  [ plotdir filesep 'autodetect-lfp.txt' ], ...
  rec_lfpgood, rec_lfptype, rec_lfpexponent, ...
  stim_lfpgood, stim_lfptype, stim_lfpexponent, ...
  rec_channels_ephys, stim_channels_ephys );

disp(thismsg);


if want_plots
  doPlotStackedSpectra( [ plotdir filesep 'lfp-background-rec.png' ], ...
    rec_lfpspectpowers, rec_lfpspectfreqs, ...
    'power', {}, rec_lfpfitpowers, rec_lfpspectfreqs, ...
    rec_channels_ephys, 'Recorder LFP Power Spectrum' );

  doPlotStackedSpectra( [ plotdir filesep 'lfp-background-stim.png' ], ...
    stim_lfpspectpowers, stim_lfpspectfreqs, ...
    'power', {}, stim_lfpfitpowers, stim_lfpspectfreqs, ...
    stim_channels_ephys, 'Stimulator LFP Power Spectrum' );
end


disp('-- Finished checking LFP noise spectrum shape.');



%
% Channel correlation tests.

disp('-- Checking for correlated channels.');

rec_correl = struct();

if have_recdata_auto

  rec_correl = ...
    nlFT_parseChannelsIntoBanks(recdata_lfp.label, recdata_lfp.trial{1});

  banklist = fieldnames(rec_correl);
  for bidx = 1:length(banklist)
    thisbank = banklist{bidx};

    [ thisgood thisrvals badgrouplist ] = nlProc_findCorrelatedChannels( ...
      rec_correl.(thisbank).wavedata, correl_abs_thresh, correl_rel_thresh );

    rec_correl.(thisbank).isgood = thisgood;
    rec_correl.(thisbank).rvalues = thisrvals;
    rec_correl.(thisbank).badgroups = badgrouplist;
  end
```

```
  thismsg = helper_reportCorrelChans( ...
    [ plotdir filesep 'autodetect-correl-rec.txt' ], ...
    'Recorder', rec_correl );
  disp(thismsg);

end


stim_correl = struct();

if have_stimdata_auto

  stim_correl = ...
    nlFT_parseChannelsIntoBanks(stimdata_lfp.label, stimdata_lfp.trial{1});

  banklist = fieldnames(stim_correl);
  for bidx = 1:length(banklist)
    thisbank = banklist{bidx};

    [ thisgood thisrvals badgrouplist] = nlProc_findCorrelatedChannels( ...
      stim_correl.(thisbank).wavedata, correl_abs_thresh, correl_rel_thresh );

    stim_correl.(thisbank).isgood = thisgood;
    stim_correl.(thisbank).rvalues = thisrvals;
    stim_correl.(thisbank).badgroups = badgrouplist;
  end

  thismsg = helper_reportCorrelChans( ...
    [ plotdir filesep 'autodetect-correl-stim.txt' ], ...
    'Stimulator', stim_correl );
  disp(thismsg);

end


disp('-- Finished checking for channel correlation.');



%
% Save results to disk, if requested.

if want_save_data
  fname = [ datadir filesep 'autoclassify.mat' ];

  if isfile(fname)
    delete(fname);
  end
```

```
  disp('-- Saving channel auto-classification results.');

  save( fname, ...
    'rec_channels_ephys', 'stim_channels_ephys', ...
    'rec_bits', 'stim_bits', 'rec_quantized', 'stim_quantized', ...
    'rec_has_dropouts', 'stim_has_dropouts', ...
    'rec_dropout_frac', 'stim_dropout_frac', ...
    'rec_has_artifacts', 'stim_has_artifacts', ...
    'rec_artifact_frac', 'stim_artifact_frac', ...
    'rec_has_peaks_raw', 'rec_has_peaks_filt', ...
    'rec_peakfreqs_raw', 'rec_peakfreqs_filt', ...
    'rec_peakheights_raw', 'rec_peakheights_filt', ...
    'rec_peakwidths_raw', 'rec_peakwidths_filt', ...
    'stim_has_peaks_raw', 'stim_has_peaks_filt', ...
    'stim_peakfreqs_raw', 'stim_peakfreqs_filt', ...
    'stim_peakheights_raw', 'stim_peakheights_filt', ...
    'stim_peakwidths_raw', 'stim_peakwidths_filt', ...
    'rec_peakspectmags_raw', 'rec_peakspectfreqs_raw', ...
    'rec_peakspectmags_filt', 'rec_peakspectfreqs_filt', ...
    'stim_peakspectmags_raw', 'stim_peakspectfreqs_raw', ...
    'stim_peakspectmags_filt', 'stim_peakspectfreqs_filt', ...
    'rec_lfpgood', 'rec_lfptype', 'rec_lfpexponent', ...
    'rec_lfpspectpowers', 'rec_lfpspectfreqs', 'rec_lfpfitpowers', ...
    'stim_lfpgood', 'stim_lfptype', 'stim_lfpexponent', ...
    'stim_lfpspectpowers', 'stim_lfpspectfreqs', 'stim_lfpfitpowers', ...
    'rec_correl', 'stim_correl', ...
    '-v7.3' );

  disp('-- Finished saving.');
end



%
% Inspect the waveform data, if requested.

if want_browser

  disp('-- Rendering waveforms.');

  if have_recdata_auto
    doBrowseFiltered( 'Rec', ...
      recdata_wideband, recdata_lfp, recdata_spike, recdata_rect );
  end

  if have_stimdata_auto
    doBrowseFiltered( 'Stim', ...
      stimdata_wideband, stimdata_lfp, stimdata_spike, stimdata_rect );
  end
```

```
  disp('-- Press any key to continue.');
  pause;

  % Clean up.
  close all;

end



%
% Clean up intermediate data.

if have_recdata_auto
  clear recdata_auto;
  clear recdata_wideband recdata_lfp recdata_spike recdata_rect;
end

if have_stimdata_auto
  clear stimdata_auto;
  clear stimdata_wideband stimdata_lfp stimdata_spike stimdata_rect;
end



%
% Banner.

disp('-- Finished auto-classifying channels.');



%
% Helper functions.


% Quantization report.
% If fname is non-empty, the report is also written to a file.

function reporttext = helper_reportQuantized( ...
  fname, ...
  rec_quantized, stim_quantized, ...
  rec_channels_ephys, stim_channels_ephys )

  nchans_rec = length(rec_channels_ephys);
  nchans_stim = length(stim_channels_ephys);

  reporttext = sprintf( ...
    '.. %d of %d recording channels were quantized.\n', ...
    sum(rec_quantized), nchans_rec );
```

```
  for cidx = 1:nchans_rec
    if rec_quantized(cidx)
      reporttext = [ reporttext ...
        '  ' rec_channels_ephys{cidx} '\n' ];
    end
  end

  reporttext = [ reporttext ...
    sprintf( '.. %d of %d stimulation channels were quantized.\n', ...
      sum(stim_quantized), nchans_stim ) ];

  for cidx = 1:nchans_stim
    if stim_quantized(cidx)
      reporttext = [ reporttext ...
        '  ' stim_channels_ephys{cidx} '\n' ];
    end
  end

  if ~isempty(fname)
    thisfid = fopen(fname, 'w');
    fwrite(thisfid, reporttext);
    fclose(thisfid);
  end

end



% Dropout and artifact report.
% If fname is non-empty, the report is also written to a file.

function reporttext = helper_reportDropoutArtifact( ...
  fname, ...
  rec_has_artifacts, stim_has_artifacts, ...
  rec_artifact_frac, stim_artifact_frac, ...
  rec_has_dropouts, stim_has_dropouts, ...
  rec_dropout_frac, stim_dropout_frac, ...
  rec_channels_ephys, stim_channels_ephys )

  nchans_rec = length(rec_channels_ephys);
  nchans_stim = length(stim_channels_ephys);

  reporttext = sprintf( ...
    '.. %d of %d recording channels had artifacts.\n', ...
    sum(rec_has_artifacts), nchans_rec );

  for cidx = 1:nchans_rec
    if rec_has_artifacts(cidx)
      reporttext = [ reporttext ...
```

```
      sprintf( '  %s  (%.1f %%)\n', ...
        rec_channels_ephys{cidx}, 100 * rec_artifact_frac(cidx) ) ];
    end
  end

  reporttext = [ reporttext ...
    sprintf( '.. %d of %d stimulation channels had artifacts.\n', ...
      sum(stim_has_artifacts), nchans_stim ) ];

  for cidx = 1:nchans_stim
    if stim_has_artifacts(cidx)
      reporttext = [ reporttext ...
        sprintf( '  %s  (%.1f %%)\n', ...
          stim_channels_ephys{cidx}, 100 * stim_artifact_frac(cidx) ) ];
    end
  end

  reporttext = [ reporttext ...
    sprintf( '.. %d of %d recording channels had drop-outs.\n', ...
      sum(rec_has_dropouts), nchans_rec ) ];

  for cidx = 1:nchans_rec
    if rec_has_dropouts(cidx)
      reporttext = [ reporttext ...
        sprintf( '  %s  (%.1f %%)\n', ...
          rec_channels_ephys{cidx}, 100 * rec_dropout_frac(cidx) ) ];
    end
  end

  reporttext = [ reporttext ...
    sprintf( '.. %d of %d stimulation channels had drop-outs.\n', ...
      sum(stim_has_dropouts), nchans_stim ) ];

  for cidx = 1:nchans_stim
    if stim_has_dropouts(cidx)
      reporttext = [ reporttext ...
        sprintf( '  %s  (%.1f %%)\n', ...
          stim_channels_ephys{cidx}, 100 * stim_dropout_frac(cidx) ) ];
    end
  end

  if ~isempty(fname)
    thisfid = fopen(fname, 'w');
    fwrite(thisfid, reporttext);
    fclose(thisfid);
  end

end
```

```
% Narrow-band noise peaks report.
% If fname is non-empty, the report is also written to a file.

function reporttext = helper_reportNarrowBandNoise( ...
  fname, ...
  rec_has_peaks, stim_has_peaks, ...
  rec_peakfreqs, stim_peakfreqs, ...
  rec_peakheights, stim_peakheights, ...
  rec_peakwidths, stim_peakwidths, ...
  rec_channels_ephys, stim_channels_ephys )

  nchans_rec = length(rec_channels_ephys);
  nchans_stim = length(stim_channels_ephys);


  reporttext = sprintf( ...
    '.. %d of %d recording channels had narrow-band noise.\n', ...
    sum(rec_has_peaks), nchans_rec );

  if ~isempty(rec_has_peaks)
    for cidx = 1:nchans_rec
      if rec_has_peaks(cidx)
        freqlist = rec_peakfreqs{cidx};
        heightlist = rec_peakheights{cidx};
        widthlist = rec_peakwidths{cidx};

        reporttext = [ reporttext ...
          '  ' rec_channels_ephys{cidx} ':' ...
          helper_makePrettyPeakList( freqlist, heightlist, widthlist ) ...
          sprintf('\n') ];
      end
    end
  end


  reporttext = [ reporttext sprintf( ...
    '.. %d of %d stimulation channels had narrow-band noise.\n', ...
    sum(stim_has_peaks), nchans_stim ) ];

  if ~isempty(stim_has_peaks)
    for cidx = 1:nchans_stim
      if stim_has_peaks(cidx)
        freqlist = stim_peakfreqs{cidx};
        heightlist = stim_peakheights{cidx};
        widthlist = stim_peakwidths{cidx};

        reporttext = [ reporttext ...
          '  ' stim_channels_ephys{cidx} ':' ...
          helper_makePrettyPeakList( freqlist, heightlist, widthlist ) ...
```

```
          sprintf('\n') ];
      end
    end
  end


  if ~isempty(fname)
    thisfid = fopen(fname, 'w');
    fwrite(thisfid, reporttext);
    fclose(thisfid);
  end

end



% Format a pretty string containing a list of spectrum peaks.

function prettytext = ...
  helper_makePrettyPeakList( freqlist, heightlist, widthlist )

  prettytext = '';

  for pidx = 1:length(freqlist)
    thisfreq = freqlist(pidx);
    thisheight = heightlist(pidx);
    thiswidth = widthlist(pidx);

    prettytext = [ prettytext ...
      sprintf( '  %.1f  (%.1fx avg %.2f Hz)', ...
        thisfreq, thisheight, thiswidth * thisfreq ) ];
  end

end



% LFP spectrum shape report.
% If fname is non-empty, the report is also written to a file.

function reporttext = helper_reportLFPShape( ...
  fname, ...
  rec_lfpgood, rec_lfptype, rec_lfpexponent, ...
  stim_lfpgood, stim_lfptype, stim_lfpexponent, ...
  rec_channels_ephys, stim_channels_ephys )


  nchans_rec = length(rec_channels_ephys);
  nchans_stim = length(stim_channels_ephys);
```

```
reporttext = sprintf('.. Recording channels by type:\n');

if ~isempty(rec_lfptype)
  typetally = struct();

  for cidx = 1:nchans_rec
    thistype = rec_lfptype{cidx};

    if isfield(typetally, thistype)
      typetally.(thistype) = typetally.(thistype) + 1;
    else
      typetally.(thistype) = 1;
    end
  end

  typelist = sort(fieldnames(typetally));
  for tidx = 1:length(typelist)
    thistype = typelist{tidx};
    reporttext = [ reporttext sprintf( '  %4d - "%s"\n', ...
      typetally.(thistype), thistype ) ];
  end
end

if ~isempty(rec_lfpgood)
  reporttext = [ reporttext sprintf( ...
    '.. %d of %d recorder LFPs good.\n', ...
    sum(rec_lfpgood), nchans_rec ) ];
end

% FIXME - Not reporting the exponent list.


reporttext = [ reporttext sprintf('.. Stimulation channels by type:\n') ];

if ~isempty(stim_lfptype)
  typetally = struct();

  for cidx = 1:nchans_stim
    thistype = stim_lfptype{cidx};

    if isfield(typetally, thistype)
      typetally.(thistype) = typetally.(thistype) + 1;
    else
      typetally.(thistype) = 1;
    end
  end

  typelist = sort(fieldnames(typetally));
  for tidx = 1:length(typelist)
```

```
      thistype = typelist{tidx};
      reporttext = [ reporttext sprintf( '  %4d - "%s"\n', ...
        typetally.(thistype), thistype ) ];
    end
  end

  if ~isempty(stim_lfpgood)
    reporttext = [ reporttext sprintf( ...
      '.. %d of %d stimulator LFPs good.\n', ...
      sum(stim_lfpgood), nchans_stim ) ];
  end

  % FIXME - Not reporting the exponent list.


  if ~isempty(fname)
    thisfid = fopen(fname, 'w');
    fwrite(thisfid, reporttext);
    fclose(thisfid);
  end

end



% Correlated channels report.
% If fname is non-empty, the report is also written to a file.

function reporttext = helper_reportCorrelChans( ...
  fname, devname, correl_struct )

  reporttext = sprintf('.. Correlated channels for "%s":\n', devname);

  banklist = fieldnames(correl_struct);
  for bidx = 1:length(banklist)

    thisbank = banklist{bidx};
    thiscorrel = correl_struct.(thisbank);

    reporttext = [ reporttext sprintf( '.. Bank "%s":  %d of %d good.\n', ...
      thisbank, sum(thiscorrel.isgood), length(thiscorrel.isgood) ) ];

    goodlist = thiscorrel.label(thiscorrel.isgood);
    reporttext = [ reporttext sprintf( '  %s\n', ...
      helper_formatLabelList(goodlist) ) ];

    badgroups = thiscorrel.badgroups;
    for gidx = 1:length(badgroups)
      thislabellist = thiscorrel.label( badgroups{gidx} );
      reporttext = [ reporttext sprintf( '.. Bad group %d:\n  %s\n', ...
```

```
        gidx, helper_formatLabelList(thislabellist) ) ];
    end

  end

  reporttext = [ reporttext sprintf( ...
    '.. End of correlated channels for "%s".\n', devname ) ];


  if ~isempty(fname)
    thisfid = fopen(fname, 'w');
    fwrite(thisfid, reporttext);
    fclose(thisfid);
  end

end



% This returns a comma-separated list of the listed labels.
% I'm sure there's a Matlab function that does this too.

function listtext = helper_formatLabelList(labels)

  listtext = [];
  for lidx = 1:length(labels)
    if lidx > 1
      listtext = [ listtext ', ' ];
    end
    listtext = [ listtext labels{lidx} ];
  end

end



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

Chapter 9

Monolithic Processing

9.1 do_test_process_monolithic.m

```
% Field Trip sample script / test script - Monolithic data processing.
% Written by Christopher Thomas.

% This reads data without segmenting it, performs signal processing, and
% optionally displays it using FT's browser.
% FIXME - Doing this by reading and setting workspace variables directly.
%
% Variables that get set:
%   have_recdata_an
%   have_recdata_dig
%   recdata_an
%   recdata_dig
%   have_stimdata_an
%   have_stimdata_dig
%   have_stimdata_current
%   have_stimdata_flags
%   stimdata_an
%   stimdata_dig
%   stimdata_current
%   stimdata_flags
%   have_recevents_dig
%   have_stimevents_dig
%   recevents_dig
%   stimevents_dig
%   recdata_wideband
%   recdata_lfp
%   recdata_spike
%   recdata_rect
%   stimdata_wideband
%   stimdata_lfp
%   stimdata_spike
```

```
%   stimdata_rect


%
% Load cached results from disk, if requested.
% If we successfully load data, bail out without further processing.

fname_raw = [ datadir filesep 'monolithic_raw.mat' ];
fname_cooked = [ datadir filesep 'monolithic_filtered.mat' ];
fname_ttlevents = [ datadir filesep 'monolithic_ttl_events.mat' ];

if want_cache_monolithic ...
  && isfile(fname_raw) && isfile(fname_cooked) && isfile(fname_ttlevents)

  % Load the data we previously saved.

  disp('-- Loading raw monolithic data.');

  load(fname_raw);

  disp('-- Loading processed monolithic data.');

  load(fname_cooked);

  disp('-- Loading TTL event lists from monolithic data.');

  load(fname_ttlevents);

  recevents_dig = struct([]);
  if have_recevents_dig
    recevents_dig = ...
      nlFT_uncompressFTEvents( recevents_dig_tab, rechdr.label );
  end
  stimevents_dig = struct([]);
  if have_stimevents_dig
    stimevents_dig = ...
      nlFT_uncompressFTEvents( stimevents_dig_tab, stimhdr.label );
  end

  disp('-- Finished loading.');


  % Generate reports and plots.
% FIXME - Monolithic reports and plots NYI.


  % Pull up the data browser windows, if requested.
  if want_browser
    disp('-- Rendering waveforms.');
```

```
    % Analog data.
    if have_recdata_an
      doBrowseFiltered( 'Rec', ...
        recdata_wideband, recdata_lfp, recdata_spike, recdata_rect );
    end
    if have_stimdata_an
      doBrowseFiltered( 'Stim', ...
        stimdata_wideband, stimdata_lfp, stimdata_spike, stimdata_rect );
    end

    % Continuous digital data.
    if have_recdata_dig
      doBrowseWave( recdata_dig, 'Recorder TTL' );
    end
    if have_stimdata_dig
      doBrowseWave( stimdata_dig, 'Stimulator TTL' );
    end

    disp('-- Press any key to continue.');
    pause;

    % Clean up.
    close all;
  end


  % We've loaded cached results. Bail out of this portion of the script.
  return;
end



%
% Read the dataset using ft_preprocessing().


% Select the default (large) time window.

preproc_config_rec.trl = preproc_config_rec_span_default;
preproc_config_stim.trl = preproc_config_stim_span_default;


% Turn off the progress bar.
preproc_config_rec.feedback = 'no';
preproc_config_stim.feedback = 'no';


% Read the data.

% NOTE - Field Trip will throw an exception if this fails. Wrap this to
```

```
% catch exceptions.

have_recdata_an = false;
have_stimdata_an = false;

recdata_an = struct([]);
stimdata_an = struct([]);

have_recdata_dig = false;
have_stimdata_dig = false;

recdata_dig = struct([]);
stimdata_dig = struct([]);

have_stimdata_current = false;
have_stimdata_flags = false;

stimdata_current = struct([]);
stimdata_flags = struct([]);

have_recevents_dig = false;
have_stimevents_dig = false;

recevents_dig = struct([]);
stimevents_dig = struct([]);

try

  disp('-- Reading ephys amplifier data.');
  tic();

  % Report the window span.
  disp(sprintf( ...
    '.. Read window is:   %.1f - %.1f s (rec)   %.1f - %.1f s (stim).', ...
    preproc_config_rec_span_default(1) / rechdr.Fs, ...
    preproc_config_rec_span_default(2) / rechdr.Fs, ...
    preproc_config_stim_span_default(1) / stimhdr.Fs, ...
    preproc_config_stim_span_default(2) / stimhdr.Fs ));


  % NOTE - Reading as double. This will be big!

  if isempty(rec_channels_ephys)
    disp('.. Skipping recorder (no channels selected).');
  else
    preproc_config_rec.channel = rec_channels_ephys;
    recdata_an = ft_preprocessing(preproc_config_rec);
    have_recdata_an = true;
  end
```

```
if isempty(stim_channels_ephys)
  disp('.. Skipping stimulator (no channels selected).');
else
  preproc_config_stim.channel = stim_channels_ephys;
  stimdata_an = ft_preprocessing(preproc_config_stim);
  have_stimdata_an = true;
end

thisduration = euUtil_makePrettyTime(toc());
disp(sprintf( '.. Read in %s.', thisduration ));


disp('-- Reading digital waveforms.');
tic();

if isempty(rec_channels_digital)
  disp('.. Skipping recorder (no channels selected).');
else
  preproc_config_rec.channel = rec_channels_digital;
  recdata_dig = ft_preprocessing(preproc_config_rec);
  have_recdata_dig = true;
end

if isempty(stim_channels_digital)
  disp('.. Skipping stimulator (no channels selected).');
else
  preproc_config_stim.channel = stim_channels_digital;
  stimdata_dig = ft_preprocessing(preproc_config_stim);
  have_stimdata_dig = true;
end

thisduration = euUtil_makePrettyTime(toc());
disp(sprintf( '.. Read in %s.', thisduration ));


disp('-- Reading digital events.');
tic();

if ~want_data_events
  disp('.. Skipping events.');
else
  disp('.. Reading from recorder.');

  recevents_dig = ft_read_event( thisdataset.recfile, ...
    'headerformat', 'nlFT_readHeader', 'eventformat', 'nlFT_readEvents' );

  % FIXME - Kludge for drivers that don't report events.
  % We actually don't need this - our Intan wrapper does this internally.
  if isempty(recevents_dig)
    disp('.. No recorder events found. Trying again using waveforms.');
```

```
    recevents_dig = ft_read_event( thisdataset.recfile, ...
      'headerformat', 'nlFT_readHeader', ...
      'eventformat', 'nlFT_readEventsContinuous' );
  end

  disp('.. Reading from stimulator.');

  stimevents_dig = ft_read_event( thisdataset.stimfile, ...
    'headerformat', 'nlFT_readHeader', 'eventformat', 'nlFT_readEvents' );

  % FIXME - Kludge for drivers that don't report events.
  % We actually don't need this - our Intan wrapper does this internally.
  if isempty(stimevents_dig)
    disp('.. No stimulator events found. Trying again using waveforms.');
    stimevents_dig = ft_read_event( thisdataset.stimfile, ...
      'headerformat', 'nlFT_readHeader', ...
      'eventformat', 'nlFT_readEventsContinuous' );
  end

  % NOTE - We have event lists, but those lists might be empty.
  have_recevents_dig = true;
  have_stimevents_dig = true;
end

thisduration = euUtil_makePrettyTime(toc());
disp(sprintf( '.. Read in %s.', thisduration ));


disp('-- Reading stimulation data.');
tic();

if isempty(stim_channels_current)
  disp('.. Skipping stimulation current (no channels selected).');
else
  preproc_config_stim.channel = stim_channels_current;
  stimdata_current = ft_preprocessing(preproc_config_stim);
  have_stimdata_current = true;
end

% NOTE - Reading flags as double. We can still perform bitwise operations
% on them.
if isempty(stim_channels_flags)
  disp('.. Skipping stimulation flags (no channels selected).');
else
  preproc_config_stim.channel = stim_channels_flags;
  stimdata_flags = ft_preprocessing(preproc_config_stim);
  have_stimdata_flags = true;
end

thisduration = euUtil_makePrettyTime(toc());
```

```
  disp(sprintf( '.. Read in %s.', thisduration ));


  % Done.
  disp('-- Finished reading data.');


catch errordetails
  disp(sprintf( ...
    '###  Exception thrown while reading "%s".', thisdataset.title));
  disp(sprintf('Message: "%s"', errordetails.message));

  % Abort the script and send the user back to the Matlab prompt.
  error('Couldn''t read signals/events; bailing out.');
end



%
% Filter the continuous ephys data.


% FIXME - We need to aggregate these, after time alignment.
% FIXME - We need to re-reference these in individual batches, not globally.
% After alignment and re-referencing, we can use ft_appenddata().
% In practice aggregating monolithic isn't necessarily useful; we can do it
% for trials once alignment is known.

recdata_wideband = struct([]);
recdata_lfp = struct([]);
recdata_spike = struct([]);
recdata_rect = struct([]);

stimdata_wideband = struct([]);
stimdata_lfp = struct([]);
stimdata_spike = struct([]);
stimdata_rect = struct([]);


if have_recdata_an

  % De-trending and power-line filtering.

  disp('.. [Rec] De-trending and removing power-line noise.');
  tic();

  extra_notches = [];
  if isfield( thisdataset, 'extra_notches' )
    extra_notches = thisdataset.extra_notches;
  end
```

```
  recdata_an = doSignalConditioning( recdata_an, ...
    power_freq, power_filter_modes, filter_type_long, ...
    extra_notches );

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. [Rec] Power line noise removed in %s.', thisduration ));


  % Artifact removal.

  % FIXME - NYI.
  disp('###  Artifact removal NYI!');


  % Re-referencing.

  % FIXME - NYI.
  % This needs to be done in batches of channels, representing different
  % probes.
  disp('###  Rereferencing NYI!');


  %
  % Get spike and LFP and rectified waveforms.

  % Copy the wideband signals.
  recdata_wideband = recdata_an;

  disp('.. [Rec] Generating LFP, spike, and rectified activity data series.');
  tic();

  [ recdata_lfp recdata_spike recdata_rect ] = ...
    doFeatureFiltering( recdata_wideband, ...
      lfp_corner, lfp_rate, spike_corner, ...
      rect_corners, rect_lowpass, rect_rate );

  thisduration = euUtil_makePrettyTime(toc());
  disp(sprintf( '.. [Rec] Filtered series generated in %s.', thisduration ));


  % Done.

end


if have_stimdata_an

  % De-trending and power-line filtering.
```

```
disp('.. [Stim] De-trending and removing power-line noise.');
tic();

extra_notches = [];
if isfield( thisdataset, 'extra_notches' )
  extra_notches = thisdataset.extra_notches;
end

stimdata_an = doSignalConditioning( stimdata_an, ...
  power_freq, power_filter_modes, filter_type_long, ...
  extra_notches );

thisduration = euUtil_makePrettyTime(toc());
disp(sprintf( '.. [Stim] Power line noise removed in %s.', thisduration ));


% Artifact removal.

% FIXME - NYI.
disp('###  Artifact removal NYI!');


% Re-referencing.

% FIXME - NYI.
% This needs to be done in batches of channels, representing different
% probes.
disp('###  Rereferencing NYI!');


%
% Get spike and LFP and rectified waveforms.

% Copy the wideband signals.
stimdata_wideband = stimdata_an;

disp('.. [Stim] Generating LFP, spike, and rectified activity data series.');
tic();

[ stimdata_lfp stimdata_spike stimdata_rect ] = ...
  doFeatureFiltering( stimdata_wideband, ...
    lfp_corner, lfp_rate, spike_corner, ...
    rect_corners, rect_lowpass, rect_rate );

thisduration = euUtil_makePrettyTime(toc());
disp(sprintf( '.. [Stim] Filtered series generated in %s.', thisduration ));


% Done.
```

```
end



%
% Save the results to disk, if requested.

if want_save_data

  if isfile(fname_raw)       ; delete(fname_raw)       ; end
  if isfile(fname_cooked)    ; delete(fname_cooked)    ; end
  if isfile(fname_ttlevents) ; delete(fname_ttlevents) ; end

  disp('-- Saving raw monolithic data.');

  save( fname_raw, ...
    'have_recdata_an', 'recdata_an', ...
    'have_recdata_dig', 'recdata_dig', ...
    'have_stimdata_an', 'stimdata_an', ...
    'have_stimdata_dig', 'stimdata_dig', ...
    'have_stimdata_current', 'stimdata_current', ...
    'have_stimdata_flags', 'stimdata_flags', ...
    '-v7.3' );

  disp('-- Saving processed monolithic data.');

  save( fname_cooked, ...
    'recdata_wideband', 'recdata_lfp', 'recdata_spike', 'recdata_rect', ...
    'stimdata_wideband', 'stimdata_lfp', 'stimdata_spike', 'stimdata_rect', ...
    '-v7.3' );

  disp('-- Saving compressed TTL event lists from monolithic data.');

  % NOTE - Saving TTL events in packed tabular form, as that's far smaller
  % than structure array form.

  recevents_dig_tab = table();
  if have_recevents_dig
    [ recevents_dig_tab scratchlut ] = ...
      nlFT_compressFTEvents( recevents_dig, rechdr.label );
  end
  stimevents_dig_tab = table();
  if have_stimevents_dig
    [ stimevents_dig_tab scratchlut ] = ...
      nlFT_compressFTEvents( stimevents_dig, stimhdr.label );
  end

  save( fname_ttlevents, ...
    'have_recevents_dig', 'recevents_dig_tab', ...
    'have_stimevents_dig', 'stimevents_dig_tab', ...
```

```
    '-v7.3' );

  disp('-- Finished saving.');
end



%
% Inspect the waveform data, if requested.

if want_browser

  disp('-- Rendering waveforms.');

  % Analog data.

  if have_recdata_an
    doBrowseFiltered( 'Rec', ...
      recdata_wideband, recdata_lfp, recdata_spike, recdata_rect );
  end

  if have_stimdata_an
    doBrowseFiltered( 'Stim', ...
      stimdata_wideband, stimdata_lfp, stimdata_spike, stimdata_rect );
  end


  % Continuous digital data.

  if have_recdata_dig
    doBrowseWave( recdata_dig, 'Recorder TTL' );
  end

  if have_stimdata_dig
    doBrowseWave( stimdata_dig, 'Stimulator TTL' );
  end


  % Done.

  disp('-- Press any key to continue.');
  pause;

  % Clean up.
  close all;

end
```

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