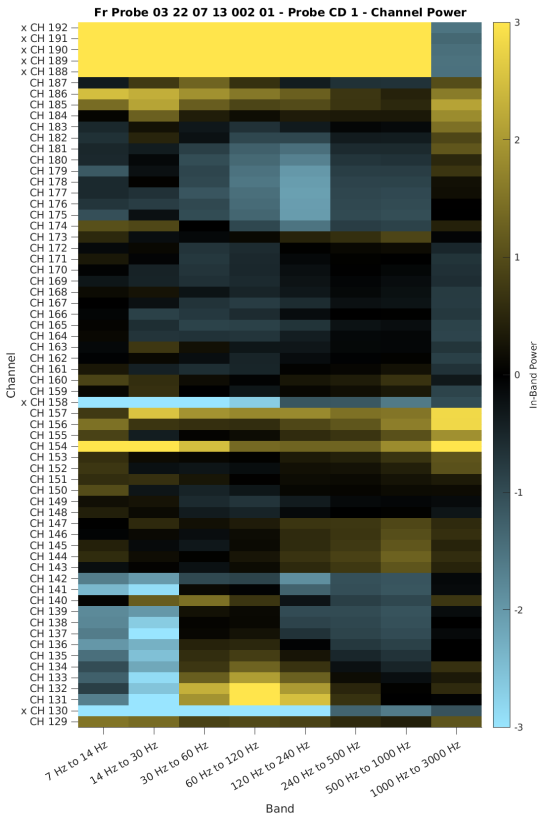
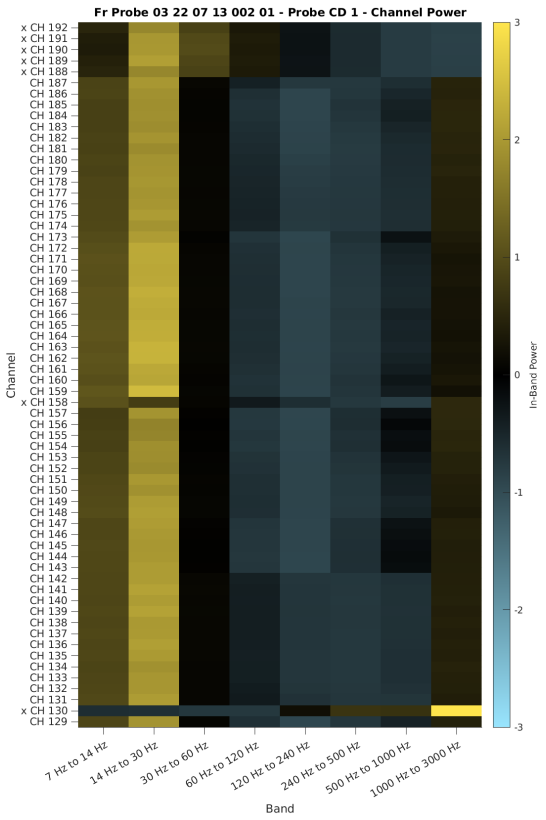
Preprocessing Pipeline Libraries – Function Reference

Written by Christopher Thomas – August 21, 2024.





Overview

This is a set of libraries written to support preprocessing of ephys data collected by Thilo's lab.

The library routines are intended to cover the following tasks:

- Reading experiment metadata.
- Reading raw ephys data, game data, and gaze data.
- Segmenting experiment data into per-trial records.
- Performing artifact rejection on ephys data.
- Assisting bad channel detection for ephys data.
- Producing per-trial records with derived ephys signals (MUA, LFP, etc).

Not presently implemented but within the scope of this project:

- Performing filtering and cleanup of gaze data.
- Saving per-trial gaze data in Field Trip format.

The idea is to make it easy and fast to write pre-processing code so that effort may be focused on analysis of ephys and behavior data after pre-processing.

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

The following sets of library functions are provided:

- **"`epIter`"** functions (Chapter 3) are functions intended to perform various preprocessing steps for each session/probe/trial, and a top-level entry-point function (`epIter_processSessions.m`) that calls these helper functions. Further documentation is provided in Chapter 2.

The following sample code is provided (in the "`code-examples`" folder):

- **"`do_config.m`"** – Specifies where to find raw datasets, specifies configuration information for each of the preprocessing operations.

- **"`do_sessionmeta.m`"** – Reads eyphs headers and auxiliary (game) files, builds event lists and trial definitions, and saves all of this metadata in a consolidated format.
- **"`do_preproc.m`"** – Reads the raw ephys datasets, performs several preprocessing steps, and saves the results of each step in Field Trip format. Preprocessing steps are segmentation into trials, artifact removal, bad channel detection, and filtering, rectification, and downsampling to produce derived signals.
- **"`do_epoch.m`"** – Reads per-trial Field Trip files, time-aligns them to desired events, and crops them to a region of interest around these events.
- **"`do_manual_badchans.m`"** – Manually specifies bad channels. These can be used instead of automatically detected bad channels by changing a configuration flag in `do_config.m`.

To run the sample analysis code:

- Make sure that the configuration file points to the FLToken dataset.
- Make sure that all needed libraries are on Matlab's search path, or are available via the symbolic links in the "`libs-ext`" folder.
- If using "`make`" from the Linux or MacOS command line:

```
make allclean
make session
make preproc
make epoch
```

- To run from within the Matlab GUI:
  - Make sure the "`data-sessions`", "`data-trials`", "`data-epoched`", and "`plots`" folders exist.
  - Run the following commands:

```
do_sessionmeta
do_preproc
do_epoch
```

- Plots will be produced by the pre-processing script (bad channel analysis plots) and by the epoching script (strip-chart plots of timelocked signals and of selected individual trials). Use "`make gallery`" from the Linux or MacOS command line to build a gallery web page showing these pictures, if desired.

Contents

Part I

Examples

Chapter 1

Preprocessing Example Script

1.1 do_config.m

```
% Preprocessing pipeline demo scripts - Configuration.


%
% Behavior Switches


% Whether to skip or force-renegerate output files that are already present.
want_force_redo = true;

% Whether to auto-detect bad channels (the alternative is to use a list).
want_detect_badchans = true;

% Whether to emit messages to the console.
want_messages = true;

% Which derived ephys signals to generate.
derived_wanted = { 'wb', 'hp', 'lfp', 'mua' };

% Which signals to generate epoched data for.
epoch_sigs_wanted = { 'mua', 'lfp' };

% Whether to use the Parallel Computing Toolbox.
want_parallel = false;


% Which plots to make for bad channel detection.

badplotconfigall = struct();
badplotconfigall.spect = struct();
badplotconfigall.spect.plot_only_average = true;
```

```
% Also available: 'powerbychan', 'tonebychan'.
% Set this to {} to suppress plots.
badplotconfigall.spect.plots_wanted = ...
  { 'powerheatmap', 'toneheatmap', 'powerheatband', 'toneheatband', ...
    'powerheatdual', 'toneheatdual' };


% Which plots to make for epoched data.

plotconfig_epoch = struct();
plotconfig_epoch.want_timelock = true;
plotconfig_epoch.want_trials = true;

% Maximum number of plots of any given type to generate.
% For strip-charts, this gives the number of trials plotted.
plotconfig_epoch.max_plots = 6;

% Options are 'oneplot', 'perchannel', and 'stripchart'.
plotconfig_epoch.timelock_types = { 'stripchart' };

% Options are 'oneplot', 'perchannel', and 'pertrial'.
% None of these are particularly useful. 'stripchart' _is_ useful.
plotconfig_epoch.trial_types = { 'stripchart' };


% The specific session to use (we'll discard the other sessions for the
% demo).
%debug_specific_sessions = {};
debug_specific_sessions = { 'FrProbe0322071300201' };

% Reduced trial count, to keep demo time and output size reasonable.
debug_few_trials = true;



%
% Folders


% Output folders.

plotdir = 'plots';
sessiondir = 'data-sessions';
trialdir = 'data-trials';
epochdir = 'data-epoched';


% Sample input. This is from the Frey/Wotan FLToken dataset.

% These are lists; multiple paths and file expressions can be given.
```

```
fltoken_rawdirs = { [ 'datasets' filesep '2022-frey-token-03' ] };
fltoken_louielogs = { [ 'datasets' filesep '*token*' filesep '*m' ] };

fltoken_desiredmeta = euMeta_getDesiredSessions_FLToken_2022_2023;
fltoken_conditionlut = euMeta_getBlockConditions_FLToken_2022_2023;

fltoken_mapmethod = 'fromsequence';



%
% Trial definition parameters and metadata.


% Number of seconds to pad around trial start/end codes.
trial_pad_secs = 1;

% Method for pruning non-incremented ("bad") trials. "keep", "strict", or
% "forgiving". "strict" discards end-of-list trials, "forgiving" doesn't.

trial_prune_method = 'strict';

% Trial event code to align t=0 on.
% Chalie used 'FixCentralCueStart'.

trial_align_evcode = 'FixCentralCueStart';



%
% Epoch parameters.

% This should be one of the metadata fields in "metabytrial" from
% "XXX-trialmeta.mat". Typical fields are 'lastfixationstart',
% 'lastfixationend', 'tokentime', 'correcttime', and 'rewardtime'.
% Metadata was extracted from evcodes by "euMeta_getTrialMetadataFromCodes".

epoch_align_feature = 'lastfixationstart';

% The ROI is -0.75 sec to +1.5 sec. We're padding a bit.
epoch_timespan_sec = [ -1 2 ];



%
% Filtering and resampling parameters.


% Signal conditioning.
```

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

% FIXME - Not doing artifact removal.
% This data had no stimulation, but there will still be licking artifacts.
artifact_passes = 2;
artifact_func = NaN;


% Derived signals.

% NOTE - This should be 10x-20x the highest frequency in the signal, to
% avoid aliasing. The anti-aliasing filter is far from perfect.
downsampled_rate = 2000;

derived_config = struct();

% Typical LFP features are 2-200 Hz.
% Charlie used a 300 Hz corner and 1 ksps.
% If the filter corner frequency is too high, we'll get leakage from MUA.
derived_config.lfp_maxfreq = 300;
derived_config.lfp_samprate = downsampled_rate;

% Spikes are typically on a ms timescale but have broad tails, so
% low-frequency components matter.
% Charlie used a 100 Hz corner.
% If the filter corner frequency is too low, we'll get leakage from the LFP.
derived_config.highpass_minfreq = 100;

% For MUA, we don't care about spike tail shapes, so we can set the lower
% corner higher.
% Charlie used 750 Hz - 5 Hz, a 300 Hz low-pass corner, and 1 ksps.
% Thilo says 200 Hz low-pass is standard.
derived_config.mua_band = [ 1000 5000 ];
derived_config.mua_lowpass = 200;
derived_config.mua_samprate = downsampled_rate;


% FIXME - Gaze NYI.

% We're resampling gaze as a continuous signal.
% The raw data is nonuniformly sampled; we're interpolating.
% As long as this sampling rate is higher than the device rate (300/600 Hz),
% the exact rate shouldn't be critical.
gaze_samprate = downsampled_rate;



%
% Bad channel detection configuration.
```

```
% Hand-annotated good and bad channel lists.

badconfiglog = struct( 'annotated_are_cooked', true );


% Manual override bad channel lists.

badconfigforce = struct( 'annotated_are_cooked', true );


% Spectral detection via nlProc_getBandPower().

% If the frequency bins are wider than an octave, low frequencies will
% dominate, since the LFP is somewhere between pink and red noise.
% At higher-than-LFP frequencies, we can pretend it's white noise.

badconfigspect = struct();

% Low-frequency bin midpoints are 10, 20, 40, and 80 Hz.
badconfigspect.freqedges = [ 7 14 30 60 120 240 500 1000 3000 ];

% Z-scored acceptance ranges for in-band power and tone power.
% A value of NaN is replaced with a default. [] is read as [ nan nan ].
% The number of rows (in either of these) is the desired number of passes.
% (Defalt power range is [-2 inf], default tone range is [-inf 2].)

% NOTE - Different probes / brain regions have different suitable thresholds.
% Semi-automated user-assisted inspection is probably the way to go.

badconfigspect.bandrange = [ -1.5 inf ];
badconfigspect.tonerange = [ -inf 1.5 ];


% Top-level metadata config for bad channel detection.
% Any method with a config gets called.

badconfigall = struct();
badconfigall.log = badconfiglog;
badconfigall.force = badconfigforce;
badconfigall.spect = badconfigspect;



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

1.2 do_epoch.m

```
% Preprocessing pipeline demo scripts - Epoch extraction.
% Written by Christopher Thomas.


%
% Setup and configuration.

do_setup_stuff;



%
% Read global metadata.

infile = [ sessiondir filesep 'dataset-meta.mat' ];
if ~exist(infile)
  disp('###  Can''t find global metadata. Bailing out.');
  edit;
else
  % This provides "sessionlist", "ttldefs", "probelabels", and "probetitles".
  % We only care about "sessionlist".
  load(infile);
end



%
% Do epoch segmentation and alignment.

% Iterate signal types as the outer loop.

for sigidx = 1:length(epoch_sigs_wanted)

  thissig = epoch_sigs_wanted{sigidx};

  disp([ '== Processing "' thissig '".' ]);


  % We have to set up handles inside the signal type loop, since filenames
  % and "ftdata_XXX" variable names depend on signal type.


  % Use "before trial" to get the bad channel list.
  % These will be dropped from the trials.

  beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
    epIter_beforeFunc_readBadChanList( ...
```

```
    sprintf( [ sessiondir filesep '%s-%s-badlist.mat' ], ...
      sessionmeta.sessionlabel, probemeta.label ), wantmsgs );

% Use "after processing" to merge and save the trimmed trials.

afterfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, ...
    beforedata, trialresults, wantmsgs ) ...
    epIter_afterFunc_epoch( ...
      [ epochdir filesep '%s-%s-' thissig '-ephys.mat' ], ...
      [ epochdir filesep '%s-%s-' thissig '-meta.mat' ], ...
      sessionmeta, probemeta, trialdefmeta, trialresults, wantmsgs );

% Use "per trial" processing to read and trim individual trials.

trialfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
    epIter_trialFunc_epoch( ...
      [ trialdir filesep '%s-%s-%s-ephys%s.mat' ], ...
      thissig, sessionmeta, probemeta, trialdefmeta, tidx, ...
      beforedata, trialdefmeta.metabytrial(tidx).(epoch_align_feature), ...
      epoch_timespan_sec, wantmsgs );


% Iterate.

epIter_processSessions( sessionlist, ...
  [ sessiondir filesep '%s-trialmeta.mat' ], ...
  beforefunc, afterfunc, trialfunc, want_parallel, want_messages );


%
% Make plots if requested.

beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
  helper_plotEpochData( ...
    sprintf( [ epochdir filesep '%s-%s-' thissig '-ephys.mat' ], ...
      sessionmeta.sessionlabel, probemeta.label ), ...
    sprintf( [ epochdir filesep '%s-%s-' thissig '-meta.mat' ], ...
      sessionmeta.sessionlabel, probemeta.label ), ...
    plotdir, thissig, plotconfig_epoch, wantmsgs );

if plotconfig_epoch.want_timelock || plotconfig_epoch.want_trials
  epIter_processSessions( sessionlist, ...
    [ sessiondir filesep '%s-trialmeta.mat' ], ...
    beforefunc, NaN, NaN, want_parallel, want_messages );
end


% Ending banner.
```

```
  disp([ '== Finished processing "' thissig '".' ]);

end



%
% Helper functions.


function retval = helper_plotEpochData( ...
  ephysfile, metafile, plotfolder, siglabel, plotconfig, wantmsgs )

  % Return value isn't used, so set it to NaN.
  retval = NaN;


  % This has "ftdata" and "ftlabels_cooked".
  load(ephysfile);

  % Swap in cooked lables for plotting.
  ftdata.label = ftlabels_cooked;


  % This has metadata variables per PREPROCFILES.txt.
  % Put them all into a structure for convenience.
  epochmeta = load(metafile);

  % Unpack some of the metadata.
  sessionlabel = epochmeta.sessionmeta.sessionlabel;
  sessiontitle = epochmeta.sessionmeta.sessiontitle;
  probelabel = epochmeta.probemeta.label;
  probetitle = epochmeta.probemeta.title;
  trialtitles = epochmeta.newtrialmeta.trialnames;

  % Reasonable plotting defaults.
  plotsigma = 2.0;
  zoomranges = {[]};
  zoomlabels = {'wide'};


  if plotconfig.want_timelock
    % Generate timelock data.
    % We need to force alignment sanity first.

    ftdata = nlFT_roundTrialTimestamps( ftdata );
    fttimelock = ft_timelockanalysis( struct(), ftdata );
```

```
    % Render the plots.

    if wantmsgs
      disp('-- Plotting time-locked data.');
    end

    euPlot_plotFTTimelock( fttimelock, ...
      plotsigma, plotconfig.timelock_types, zoomranges, zoomlabels, ...
      plotconfig.max_plots, ...
      [ sessiontitle ' - ' probetitle ' - ' siglabel ], ...
      [ plotfolder filesep 'timelock-' sessionlabel '-' probelabel ...
        '-' siglabel ] );
  end


  if plotconfig.want_trials
    % Render the plots.

    if wantmsgs
      disp('-- Plotting trials.');
    end

    euPlot_plotFTTrials( ftdata, ftdata.fsample, [], ...
      trialtitles, NaN, struct(), NaN, plotconfig.trial_types, ...
      zoomranges, zoomlabels, plotconfig.max_plots, ...
      [ sessiontitle ' - ' probetitle ' - ' siglabel ], ...
      [ plotfolder filesep 'trials-' sessionlabel '-' probelabel ...
        '-' siglabel ] );
  end

end



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

1.3 do_manual_badchans.m

```
% Preprocessing pipeline demo scripts - Manual bad channels.
% We can override the results of bad channel detection with the manual lists
% defined here.
% Written by Christopher Thomas.


% These are manually-compiled bad channel lists.

% Automated detection isn't perfect, so for each session, I plotted the
```

```
% results of the bad channel analysis and comiled lists of bad channels by
% hand based on those plots.

% These are cooked channel labels, just like the experiment log.


badchansbyhand = struct();


% 2022 07 13 session.

scratch = struct();

% Also suspicious: 259, 260, 284, 291, 292, 304.
scratch.('prACC1') = ...
{ 'CH_257', 'CH_258', 'CH_269', 'CH_285', 'CH_286', 'CH_297', 'CH_300', ...
  'CH_316', 'CH_317', 'CH_318', 'CH_319', 'CH_320' };

% This looks like it had four very different layers.
% Also suspicious: 001, 002, 044.
% 016/017/018/045 might be a bad group or might be a real feature.
scratch.('prPFC1') = ...
{ 'CH_029', 'CH_030', 'CH_060', 'CH_061', 'CH_062', 'CH_063', 'CH_064' };

% This also seems to have layers and several different mixed types.
% Might be bad or might be real: 131/132/133 blob at 60-120 Hz.
scratch.('prCD1') = ...
{ 'CH_130', 'CH_158', 'CH_188', 'CH_189', 'CH_190', 'CH_191', 'CH_192' };

badchansbyhand.('FrProbe0322071300201') = scratch;


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

1.4 do_paths.m

```
% Preprocessing pipeline demo scripts - Path configuration.
% Written by Christopher Thomas.

% Paths.

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

```
addPathsExpPreproc;
addPathsExpUtilsCjt;
addPathsLoopUtil;


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

1.5 do_preproc.m

```
% Preprocessing pipeline demo scripts - Preprocessing.
% This reads the session metadata and pre-processes sessions and trials.
% Written by Christopher Thomas.


%
% Setup and configuration.

do_setup_stuff;



%
% Read global metadata.

infile = [ sessiondir filesep 'dataset-meta.mat' ];
if ~exist(infile)
  disp('###  Can''t find global metadata. Bailing out.');
  exit;
else
  % This provides "sessionlist", "ttldefs", "probelabels", and "probetitles".
  % We only care about "sessionlist".
  load(infile);
end



%
% Do raw segmentation.


% Use "before trial" processing to get the probe channel map.

beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
  epIter_beforeFunc_readChanMap( ...
    [ sessiondir filesep sessionmeta.sessionlabel '-chanmap.mat' ], ...
    wantmsgs );
```

```
% No "after trial" processing.

% Use the FLToken 2022-2023 raw data extraction function for trial iteration.

trialfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
    epIter_trialFunc_raw_FLToken2022( ...
      [ trialdir filesep '%s-%s-%s-ephysraw.mat' ], want_force_redo, ...
      sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs );

% Iterate.

epIter_processSessions( sessionlist, ...
  [ sessiondir filesep '%s-trialmeta.mat' ], beforefunc, NaN, ...
  trialfunc, want_parallel, want_messages );



%
% Do signal conditioning (artifact removal and notch filtering).


% No "before trial" processing.

% No "after trial" processing.

trialfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
    epIter_trialFunc_sigclean( ...
      [ trialdir filesep '%s-%s-%s-ephysclean.mat' ], ...
      [ trialdir filesep '%s-%s-%s-ephysraw.mat' ], want_force_redo, ...
      sessionmeta, probemeta, trialdefmeta, tidx, ...
      artifact_passes, artifact_func, ...
      notch_filter_freqs, notch_filter_bandwidth, wantmsgs );

% Iterate.

epIter_processSessions( sessionlist, ...
  [ sessiondir filesep '%s-trialmeta.mat' ], NaN, NaN, ...
  trialfunc, want_parallel, want_messages );



%
% Do bad channel analysis.


if want_detect_badchans
  badchansbyhand = struct();
```

```
end

% No "before trial" processing.

afterfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, ...
    beforedata, trialresults, wantmsgs ) ...
    epIter_afterFunc_badInfoAll( ...
      [ sessiondir filesep '%s-%s-badinfo.mat' ], want_force_redo, ...
      sessionmeta, probemeta, trialdefmeta, ...
      trialresults, badconfigall, badchansbyhand, wantmsgs );

trialfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
    epIter_trialFunc_badInfoAll( ...
      [ sessiondir filesep '%s-%s-badinfo.mat' ], ...
      [ trialdir filesep '%s-%s-%s-ephysclean.mat' ], want_force_redo, ...
      sessionmeta, probemeta, trialdefmeta, tidx, badconfigall, wantmsgs );

% Iterate.

epIter_processSessions( sessionlist, ...
  [ sessiondir filesep '%s-trialmeta.mat' ], NaN, afterfunc, ...
  trialfunc, want_parallel, want_messages );



%
% Use the bad channel analysis to build bad channel lists.


% "Before trial" processing handles list consolidation.

beforefunc = ...
  @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
    epIter_beforeFunc_badListMergeInfo( ...
      [ sessiondir filesep '%s-%s-badlist.mat' ], ...
      [ sessiondir filesep '%s-%s-badinfo.mat' ], want_force_redo, ...
      sessionmeta, probemeta, badconfigall, wantmsgs );

% "After trial" processing makes plots.

afterfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, ...
    beforedata, trialresults, wantmsgs ) ...
    epIter_afterFunc_badListPlotAll( ...
      [ plotdir filesep '%s-%s' ], ...
      [ sessiondir filesep '%s-%s-badinfo.mat' ], ...
      beforedata, sessionmeta, probemeta, trialdefmeta, ...
      badplotconfigall, wantmsgs );
```

```
% No "per trial" processing.

% Iterate.
% NOTE - Only the per trial stuff is parallelizable, so want_parallel
% doesn't do anything here.

epIter_processSessions( sessionlist, ...
  [ sessiondir filesep '%s-trialmeta.mat' ], beforefunc, afterfunc, ...
  NaN, want_parallel, want_messages );



%
% Get derived signals, discarding bad channels.
% This includes doing re-referencing, which is why bad channels have to be
% known.


% Use "before trial" to get the bad channel list.
% We need to omit these from the re-referencing reference list.

beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
  epIter_beforeFunc_readBadChanList( ...
    sprintf( [ sessiondir filesep '%s-%s-badlist.mat' ], ...
      sessionmeta.sessionlabel, probemeta.label ), wantmsgs );

% No "after trial" processing.

% Use the "getDerivedSignals" wrapper as the trial function.

trialfunc = ...
  @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
    epIter_trialFunc_derived( ...
      [ trialdir filesep '%s-%s-%s-ephys%s.mat' ], ...
      [ trialdir filesep '%s-%s-%s-ephysclean.mat' ], want_force_redo, ...
      sessionmeta, probemeta, trialdefmeta, tidx, ...
      beforedata, derived_wanted, derived_config, wantmsgs );


% Iterate.

epIter_processSessions( sessionlist, ...
  [ sessiondir filesep '%s-trialmeta.mat' ], beforefunc, NaN, ...
  trialfunc, want_parallel, want_messages );



%
% Done.
```

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

1.6 do_sessionmeta.m

```
% Preprocessing pipeline demo scripts - Session extraction.
% Written by Christopher Thomas.


%
% Setup and configuration.

do_setup_stuff;



%
% Get dataset metadata.


% Get a list of all sessions present.

sessionlist = ...
  euMeta_getLouieFoldersAndLogs( fltoken_rawdirs, fltoken_louielogs );



% Prune this based on the manual list of good sessions.

if isempty(fltoken_desiredmeta)
  disp('###  No desired metadata specified. Reverse-engineering.');
  fltoken_desiredmeta = euMeta_getDesiredSessions_guessLouie( sessionlist );
end

if isempty(fltoken_desiredmeta)
  disp('###  Couldn''t figure out probe groups. Bailing out.');
  sessionlist = struct([]);
  exit;
end

scratch = length(sessionlist);
sessionlist = euMeta_pruneLouieSessionList( sessionlist, fltoken_desiredmeta );

% Debug tattle.
disp(sprintf( '--  %d sessions before pruning, %d sessions after:', ...
  scratch, length(sessionlist) ));
```

```
disp(transpose( { sessionlist.sessionid } ));



% If we want only a few specific sessions, filter the list.
% If we only want one session, crop the list.

if ~isempty(debug_specific_sessions)

  sessionmask = ...
    ismember( { sessionlist.sessionlabel }, debug_specific_sessions );
  sessionlist = sessionlist(sessionmask);

  disp(sprintf( 'xx  Selected %d of %d sessions by name:', ...
    sum(sessionmask), length(sessionmask) ));
  disp(transpose( { sessionlist.sessionid } ));

end



% Sanity check.

if isempty(sessionlist)
  disp('###  No sessions in list! Bailing out.');
  exit;
end



% Get an aggregated list of probes.
% FIXME - We don't actually use this for anything. It's mostly for the
% user's benefit.

[ probelabels probetitles ] = ...
  euMeta_getAllProbeNames( { sessionlist.probedefs } );

disp('-- Probes detected:');
scratch = [ transpose(probelabels), transpose(probetitles) ];
disp(scratch);



% Other global metadata.

% This specifies which SynchBox signals correspond to which TTL inputs.
ttldefs = euMeta_getTTLSignals_FLToken_2022_2023;
```

```
% Write global metadata to disk.

disp('-- Writing dataset metadata to disk.');
save( [ sessiondir filesep 'dataset-meta.mat' ], ...
  'sessionlist', 'ttldefs', 'probelabels', 'probetitles', '-v7.3' );



%
% Walk through the sessions, getting session metadata and trialdefs.

sessioncount = length(sessionlist);

for sidx = 1:sessioncount

  sessionmeta = sessionlist(sidx);

  sessiontitle = sessionmeta.sessiontitle;
  sessionlabel = sessionmeta.sessionlabel;

  sessionfileprefix = [ sessiondir filesep sessionlabel ];

  foldersession = sessionmeta.folder_session;

  foldergame = '';
  if ~isempty(sessionmeta.folders_game)
    foldergame = sessionmeta.folders_game{1};
  end

  folderopenephys = '';
  if ~isempty(sessionmeta.folders_openephys)
    folderopenephys = sessionmeta.folders_openephys{1};
  end

  folderintanrec = '';
  if ~isempty(sessionmeta.folders_intanrec)
    folderintanrec = sessionmeta.folders_intanrec{1};
  end

  folderintanstim = '';
  if ~isempty(sessionmeta.folders_intanstim)
    folderintanstim = sessionmeta.folders_intanstim{1};
  end


  % Banner.

  disp(sprintf( '== Processing "%s" (%d/%d) (%s).', ...
    sessiontitle, sidx, sessioncount, sessionmeta.monkey ));
```

```
%
% Read event data, frame data, and eye-tracker data only once per session.


% TTL events.

outfile = [ sessionfileprefix '-events-raw.mat' ];

tic;

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

[ boxevents gameevents evcodedefs gamereftime ...
  deveventsraw deveventscooked devnames ] = ...
  euHLev_readAllTTLEvents( ttldefs, ...
    foldergame, folderopenephys, folderintanrec, folderintanstim );

disp('.. Writing raw TTL events to disk.');
save( outfile, 'boxevents', 'gameevents', 'evcodedefs', ...
  'gamereftime', 'deveventsraw', 'deveventscooked', 'devnames', ...
  '-v7.3' );

durstring = nlUtil_makePrettyTime(toc);
disp([ '-- Finished reading TTL events (' durstring ').' ]);

reportmsg = euHLev_reportTTLEvents( boxevents, gameevents, evcodedefs, ...
  deveventsraw, deveventscooked, devnames );

nlIO_writeTextFile( [ plotdir filesep sessionlabel '-eventreport.txt' ], ...
  reportmsg );


% Frame data.

outfile = [ sessionfileprefix '-framedata-raw.mat' ];

tic;

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

gameframedata = euUSE_readRawFrameData( foldergame, gamereftime );

disp('.. Writing frame data to disk.');
save( outfile, 'gameframedata', '-v7.3' );

durstring = nlUtil_makePrettyTime(toc);
disp([ '-- Finished reading frame data (' durstring ').' ]);
```

```
% Gaze data.

outfile = [ sessionfileprefix '-gazedata-raw.mat' ];

tic;

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

gamegazedata = euUSE_readRawGazeData(foldergame);

disp('.. Writing gaze data to disk.');
save( outfile, 'gamegazedata', '-v7.3' );

durstring = nlUtil_makePrettyTime(toc);
disp([ '-- Finished reading gaze data (' durstring ').' ]);



%
% Do time alignment.

outfile = [ sessionfileprefix '-align.mat' ];

tic;

disp('-- Performing time alignment.');

timetables = euHLev_alignAllDevices( ...
  boxevents, gameevents, deveventscooked, gameframedata );

disp('.. Writing time alignment tables to disk.');
save( outfile, 'timetables', '-v7.3' );

[ boxevents gameevents deveventscooked gameframedata gamegazedata ] = ...
  euHLev_propagateAlignedTimestamps( timetables, boxevents, gameevents, ...
    deveventscooked, gameframedata, gamegazedata );

durstring = nlUtil_makePrettyTime(toc);
disp([ '-- Finished performing time alignment (' durstring ').' ]);



%
% Clean up event lists.

% If any ephys machines are missing event tables, this copies them.
% Device event code tables also get augmented with TTL reward/synch events.

% NOTE - This needs the folders so that it can read FT headers to get
```

```
% sampling rates. Folders given as '' are skipped.

% FIXME - There's a note that I might want to augment game codes too.
% Skipping this, since the game emits event codes saying that it asserted
% the reward and stim lines.

disp('-- Propagating events.');

scratch = deveventscooked;

% We don't want to propagate these; USE's reward indicators are enough.
want_add_ttl_to_evcodes = false;

deveventscooked = euHLev_augmentEphysEvents( ...
  deveventscooked, boxevents, want_add_ttl_to_evcodes, ...
  folderopenephys, folderintanrec, folderintanstim );

disp('-- Finished propagating events.');



%
% Get the channel map.

disp('.. Reading channel map.');

% FIXME - We know we're using Open Ephys, and we know the channel map is
% saved in a subfolder of the session folder.

[ chanlabels_raw chanlabels_cooked ] = euMeta_getLabelChannelMap_OEv5( ...
  foldersession, folderopenephys, fltoken_mapmethod );


% Save this.
disp('.. Writing channel map to disk.');
save( [ sessiondir filesep sessionlabel '-chanmap.mat' ], ...
  'chanlabels_raw', 'chanlabels_cooked', '-v7.3' );



%
% Get the trial definitions.

disp('.. Getting full-trial trial definitions.');

% FIXME - Just getting these for the recorder for now (no stimulator).

% FIXME - Assume Open Ephys.
[ codesbytrial trialdefs trialdeftable ] = euHLev_getTrialDefsWide( ...
  folderopenephys, deveventscooked.openephys.cookedcodes, ...
```

```
  'recTime', trial_pad_secs, trial_align_evcode, trial_prune_method );

% Diagnostics.
disp(sprintf( '.. Found %d trials (and %d event code lists).', ...
  height(trialdeftable), length(codesbytrial) ));


% If we want to only process a subset of trials, crop the lists here.

oldtrialcount = length(codesbytrial);

if debug_few_trials
  trialcount = round( oldtrialcount / 40 );
  trialcount = max(trialcount, 1);

  % Only prune the list if we reduced the number of trials.
  % Since "trialcount" is at least 1, this handles the "no trials" case.

  if trialcount < oldtrialcount
    codesbytrial = codesbytrial(1:trialcount);
    trialdefs = trialdefs(1:trialcount,:);
    trialdeftable = trialdeftable(1:trialcount,:);

    disp(sprintf( 'xx  Keeping %d of %d trials.', ...
      trialcount, oldtrialcount ));
  end
end


% Follow Charlie's convention, and make trial labels based on the USE
% "trial index", which is insensitive to how we do good/bad trial weeding.

trialindices = trialdeftable.trialindex;
triallabels = nlUtil_sprintfCellArray( 'tridx%04d', trialindices );
trialnames = nlUtil_sprintfCellArray( 'TrIdx %04d', trialindices );


% Build additional per-trial metadata.

metabytrial = euMeta_getTrialMetadataFromCodes( ...
  codesbytrial, fltoken_conditionlut, 'recTime' );

if isempty(gameframedata)
  disp('###  Warning: No frame data to augment trial metadata with!');
end
metabytrial = ...
  euMeta_addFrameDataToTrialMetadata( metabytrial, gameframedata );


% Save trial definitions and other trial metadata.
```

```
  % NOTE - The event code list is pretty big.

  disp('-- Writing trial definitions and trial metadata to disk.');
  save( [ sessiondir filesep sessionlabel '-trialmeta.mat' ], ...
    'codesbytrial', 'metabytrial', ...
    'trialdefs', 'trialdeftable', 'trial_align_evcode', ...
    'trialindices', 'triallabels', 'trialnames', ...
    'sessionlabel', 'sessiontitle', '-v7.3' );

end  % Session iteration.


disp('== Finished processing raw session data.');



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

1.7 do_setup_stuff.m

```
% Preprocessing pipeline demo scripts - Common setup tasks.
% Written by Christopher Thomas.


%
% First, set up paths.

do_paths;


%
% Next, do configuration.
% Some of this calls library functions, so we need the paths.

do_config;
do_manual_badchans;


%
% Finally, do other setup tasks, which may depend on configuration switches.


% Matlab warnings.

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

```
% Field Trip initialization and messages.

nlFT_initFTQuietly;


% LoopUtil configuration.

% We use about 1 GB per channel-hour. The sweet spot to reduce loading time
% is 4 channels or more.
nlFT_setMemChans(4);


% Initialize parallel processing if requested.

if want_parallel
  % Ask for 4-8 workers.
  parpool([4 8]);
end


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

Part II

Library Functions

Chapter 2

“epIter” Notes

2.1 BADCHANCONFIG.txt

```
Iteration functions can perform bad channel detection by several methods.

The methods desired and their configuration parameters are specified by a
configuration structure with the following fields:

"log" is a structure with configuration for hand-annotated channel lists.
"force" is a structure with configuration for forced channel list overrides.
"spect" is a structure with configuration for spectral band analysis.



Bad channel detection via hand-annotated channel lists reads lists of good
and bad channels that were annotated in log files that use Louie's format.

Configuration structures for hand-annotated bad channel parsing have the
following fields:

"annotated_are_cooked" is true if channel names are cooked (i.e. translated
  using the channel map), and false if channel names are raw (i.e. the
  labels listed in ft_datatype_raw structures).



Bad channel list overrides replace the final bad channel list with a user-
-supplied list.

Configuration structures for user-specified bad channel lists have the
following fields:

"annotated_are_cooked" is true if channel names are cooked (i.e. translated
  using the channel map), and false if channel names are raw (i.e. the
```

```
  labels listed in ft_datatype_raw structures).


Bad channel detection via spectral band analysis measures power in several
frequency bands and looks for channels that are anomalous compared to other
channels.

Configuration structures for spectral band analysis have the following
fields (per nlProc_getBandPower and euHLev_guessBadChansFromBandPower):

"freqedges" is a vector with the edge frequencies of the bands to test.
"bandrange" is the range of in-band power values to accept, in standard
  deviations.
"tonerange" is the range of (log-scale) relative tone power values to
  accept, in standard deviations.



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

2.2 BADCHANPLOTS.txt

```
Configuration for plotting bad channel data is specified by a structure with
the following fields:

"spect" is a configuration structure for plotting spectral band detection
  data. If this field is absent, no plots are generated.



Configuration structures for spectral band detection plots have the following
fields:

"plot_only_average" is true to suppress per-trial plots and false otherwise.
"plots_wanted" is a cell array with zero or more of the following:
  'powerbychan' plots in-band power vs channel for each band.
  'tonebychan' plots relative tone power vs channel for each band.
  'powerheatmap' plots in-band power vs channel and band, normalized across
    channels.
  'toneheatmap' plots relative tone power vs channel and band, normalized
    across channels.
  'powerheatband' plots in-band power vs channel and band, normalized across
    bands.
  'toneheatband' plots relative tone power vs channel and band, normalized
    across bands.
```

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

2.3 ITERFUNCS.txt

```
The following types of function handle are used by epIter_processSessions.
Passing NaN instead of a function handle skips that function (so there's
no need to define do-nothing functions).



beforedata = probefuncbefore( sessionmeta, probemeta, trialdefmeta, wantmsgs )

This function is called during probe iteration, before trial iteration. It
is intended to gather session- or probe-specific data needed when processing
trials.

"sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
"probemeta" is a probe definition structure, per PROBEDEFS.txt.
"trialdefmeta" is a structure containing all of the variables in the trial
  definition metadata file, per PREPROCFILES.txt.
"wantmsgs" is true to emit console messages and false otherwise.

"beforedata" is a return variable of any type (typically a struct).



trialresult = ...
  trialfunc( sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs )

This function is called during trial iteration. It is intended to do two
things: First, it may read per-trial input files and generate per-trial
output files. Second, it may generate per-trial output data that is
aggregated for analysis (or just recording) by "probefuncafter".

"sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
"probemeta" is a probe definition structure, per PROBEDEFS.txt.
"trialdefmeta" is a structure containing all of the variables in the trial
  definition metadata file, per PREPROCFILES.txt.
"tidx" is the row index of the present trial within trial metadata tables.
"beforedata" is the data variable returned by "probefuncbefore".
"wantmsgs" is true to emit console messages and false otherwise.

"trialresult" is a return variable of any type (typically NaN if unused,
  and either a struct or a scalar if used).
```

```
probefuncafter( ...
  sessionmeta, probemeta, trialdefmeta, beforedata, trialresults, wantmsgs )

This function is called during probe iteration, after trial iteration. It
is intended to consolidate per-trial return data and to write either this
consolidated data or some derived analysis output to disk.

"sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
"probemeta" is a probe definition structure, per PROBEDEFS.txt.
"trialdefmeta" is a structure containing all of the variables in the trial
  definition metadata file, per PREPROCFILES.txt.
"beforedata" is the data variable returned by "probefuncbefore".
"trialresults" is a cell array indexed by trial row number that contains
  the data variables returned by "trialfunc".
"wantmsgs" is true to emit console messages and false otherwise.

No return value.



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

2.4 PREPROCFILES.txt

```
Files saved by the preprocessing library functions are as follows.
Filenames are configurable; the ones shown are typical.


Global metadata:

"dataset-meta.mat" contains:
  sessionlist  (struct array per SESSIONMETA.txt)
  ttldefs  (per TTLSIGNALDEFS.txt)
  probelabels  (from getAllProbeNames({ sessionlist.probedefs }))
  probetitles  (from getAllProbeNames({ sessionlist.probedefs }))



Per-session metadata that we don't have to revisit:

"(session)-events-raw.mat" contains everything returned by readAllTTLEvents(),
without time alignment or timestamp propagation:
  boxevents
  gameevents
  evcodedefs  (per EVCODEDEFS.txt)
  gamereftime  (time subtracted from all game timestamps)
  deveventsraw
  deveventscooked
```

```
  devnames  (structure with human-readable dev names indexed by dev label)


"(session)-framedata-raw.mat" contains the data table returned by
readRawFrameData(), without time alignment or timestamp propagation:
  gameframedata  (data table)


"(session)-gazedata-raw.mat" contains the data table returned by
readRawGazeData(), without time alignment or timestamp propagation:
  gamegazedata  (data table)


"(session)-align.mat" contains the tables returned by alignAllDevices():
  timetables  (structure with several tables containing timestamp tuples)


Note that the event tables and gaze/frame data tables will have to have
timestamps propagated via a call to propagateAlignedTimestamps().

Note that "deveventscooked" may need to be augmented with the canon list of
events from "boxevents" by calling augmentEphysEvents().



Per-session metadata that we do want to revisit:

"(session)-chanmap.mat" contains:
  chanlabels_raw  (Field Trip channel labels)
  chanlabels_cooked  (channel labels as used in the experiment analysis)


Note that Charlie's convention is to keep the FT channel labels as-is but
to sort channels in the FT structure based on the cooked labels.


"(session)-trialmeta.mat" contains:
  codesbytrial  (cell array with per-trial event code tables)
  metabytrial  (struct array returned by getTrialMetadataFromCodes())
  trialdefs  (Field Trip trial definition matrix plus metadata columns)
  trialdeftable  (table with "trialdefs" info, plus column names)
  trial_align_evcode  (event code name used for time zero in trials)
  trialindices  ("trial index" from the trial definition table)
  triallabels  (filename-safe trial index labels)
  trialnames  (human-readable plot-safe trial index names)
  sessionlabel  (a copy of the filename-safe session label)
  sessiontitle  (a copy of the human-readable plot-safe session name)


Note that the tables in "codesbytrial" are agumented and have timestamps
```

```
propagated from other devices.

Trial definitions were produced by getTrialDefsWide(), which is intended to
segment entire trials aligned to some common event. Subsequent analyses can
extract smaller time windows aligned to other features within these large
segments.



Per-probe data:

"(session)-(probe)-badinfo.mat" contains:
  ftlabels_raw  (a copy of ftdata_XXX.label)
  ftlabels_cooked  (cooked channel labels corresponding to ftlabels_raw)
  triallabels  (filename-safe trial labels, from "trial index")
  trialnames  (human-readable plot-safe trial names, from "trial index")
  badchandata (a structure with per-algorithm analysis output structures):
    log  (raw hand-annotated good and bad channels)
      good  (cell array with raw labels for hand-annotated good channels)
      bad  (cell array with raw labels for hand-annotated bad channels)
    force
      bad  (raw list of channels to store in "badraw", ignoring methods)
    spect  (spectral analysis for bad channel evaluation)
      freqedges  (frequency band edges)
      bandpower  (nChans x nBands x ntrials in-band power)
      tonepower  (nChans x nBands x nTrials log10(max / median) power in-band)

"(session)-(probe)-badlist.mat" contains:
  badchanmeta (a structure with the following fields:)
    goodraw  (raw FT channel labels of hand-annotated good channels)
    badraw  (raw FT channel labels of any-method bad channels)
    goodcooked  (cooked channel labels of hand-annotated good channels)
    badcooked  (cooked channel labels of any-method bad channels)
    ftlabels_raw  (raw channel labels in data order)
    ftlabels_cooked  (cooked channel labels in data order)
    goodrawbymethod (structure with per-algorithm raw good channel labels):
      log  (raw hand-annotated good channels)
    badrawbymethod (structure with per-algorithm raw bad channel labels):
      log  (raw hand-annotated bad channels)
      force  (raw list of channels to store in "badraw", ignoring methods)
      spect  (raw bad channel labels from spectral analysis)

"(session)-(probe)-(label)-ephys.mat" contains:
  ftdata  (ft_datatype_raw structure with event-aligned cropped ephys data)
  ftlabels_cooked  (cooked channel labels corresponding to ftdata.label)

"(session)-(probe)-(label)-meta.mat" contains:
  "sessionmeta"  (a single session metadata structure, per SESSIONMETA.txt)
  "probemeta"  (a probe definition structure, per PROBEDEFS.txt)
  "ftlabels_raw"  (raw FT labels in data order, with bad channels removed)
```

```
  "ftlabels_cooked"  (cooked FT labels in data order, with bad chans removed)
  "oldtrialmeta"  (structure with variables from "(session)-trialmeta.mat")
  "trialmask"  (boolean vector indicating which old trials were kept)
  "trial_maskedfromorig"  (vector with new trial indices, indexed by old)
  "trial_origfrommasked"  (vector with old trial indices, indexed by new)
  "newtrialmeta"  (copy of "oldtrialmeta" with some trials removed, and
     without "trial_align_evcode", "trialdefs", and "trialdeftable")



Per-trial data:

"(session)-(probe)-(trial)-ephysXXX.mat" contains:
  ftdata_XXX  (ft_datatype_raw structure with trial ephys data)
  ftlabels_cooked  (cooked channel labels corresponding to ftdata_XXX.label)

XXX values are:
  raw  (unprocessed ephys data)
  clean  (ephys data after artifact rejection and notch filtering)
  wb  (ephys data after re-referencing; reference is from non-bad channels)
  lfp  (low-pass-filtered ephys data; Local Field Potential)
  hp  (high-pass-filtered ephys data, for extracting spike shapes)
  mua  (band-pass/rectified/low-pass ephys data; Multi-Unit Activity)

Note that channels in ftdata_XXX have been sorted in cooked order, despite
keeping the raw names.



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

Chapter 3

"epIter" Functions

3.1 epIter_afterFunc_badInfoAll.m

```
% function epIter_afterFunc_badInfoAll( ...
%   outfilepat, wantforce, sessionmeta, probemeta, trialdefmeta, ...
%   trialresults, configall, badforcelist, wantmsgs )
%
% This function aggregates per-trial bad channel information and writes it
% to a file.
%
% This is intended to be called as a "probefuncafter" function, per
% ITERFUNCS.txt. A typical implementation would be:
%
% afterfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, ...
%     beforedata, trialresults, wantmsgs ) ...
%     epIter_afterFunc_badInfoAll( ...
%       [ destfolder filesep '%s-%s-badinfo.mat' ], ...
%       want_force_badinfo, sessionmeta, probemeta, trialdefmeta, ...
%       trialresults, bad_config_struct_all, bad_force_lut, wantmsgs );
%
% "outfilepat" is a sprintf pattern used to generate the output file name
%   for saving per-probe bad channel information. This needs two '%s' tokens,
%   for the session label and probe label (in that order).
% "wantforce" is true to redo processing even if the output file already
%   exists, and false to skip processing if the output file is present.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "trialresults" is a cell array indexed by trial row number that contains
%   the data variables returned by epIter_trialFunc_badInfoAll().
% "configall" is a configuration structure for bad channel analysis. Each
%   field corresponds to an algorithm and contains a configuration structure,
```

```
%   per BADCHANCONFIG.txt:
%   "log" contains configuration information for hand-annotated log entries.
%   "force" contains configuration for forced-override bad channel lists.
%   "spect" contains configuration information for spectral analysis.
% "badforcelist" is either a cell array or a structure. If it's a cell array,
%   it contains a list of (raw or cooked) FT channel labels that are known
%   to be bad and that should override the aggregated bad channel list. If
%   it's a struct, then badforcelist.(session).(probe) contains a cell
%   array of bad channels. If "badforcelist" is a structure but the relevant
%   session or probe field is absent, no override is performed.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% No return value.
```

3.2 epIter_afterFunc_badListPlotAll.m

```
% function epIter_badListPlotAll( ...
%   outprefixpat, infilepat, badlists, ...
%   sessionmeta, probemeta, trialdefmeta, plotconfig, wantmsgs )
%
% This function plots bad channel analysis data.
%
% This is intended to be called as a "probefuncafter" function, per
% ITERFUNCS.txt. A typical implementation would be:
%
% afterfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, ...
%     beforedata, trialresults, wantmsgs ) ...
%     epIter_afterFunc_badListPlotAll( ...
%       [ plotfolder filesep '%s-%s-badchans' ], ...
%       [ srcfolder filesep '%s-%s-badinfo.mat' ], ...
%       bad_chan_list_struct, sessionmeta, probemeta, trialdefmeta, ...
%       bad_chan_plot_config, wantmsgs );
%
% "outprefixpat" is a sprintf pattern used to generate prefixes for output
%   filenames (such as plots, lists, and CSV data tables). This needs two
%   '%s' tokens, for the session label and probe label (in that order).
% "infilepat" is a sprintf pattern used to generate the input file name
%   for reading per-probe bad channel analysis output. This needs two '%s'
%   tokens, for the session label and probe label (in that order).
% "badlists" is a structure containing the per-probe bad channel list
%   variables described in PREPROCFILES.txt. This is the same structure
%   returned by epIter_beforeFunc_badListMergeInfo().
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "plotconfig" is a structure describing the desired plots. Each field
```

```
%   corresponds to an algorithm and contains a structure with plot
%   configuration for that algorithm, per BADCHANPLOTS.txt:
%   "spect" contains configuration for spectral analysis plots.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% No return value.
```

3.3 epIter_afterFunc_epoch.m

```
% function epIter_afterFunc_epoch( ...
%   outfilepat_ephys, outfilepat_meta, ...
%   sessionmeta, probemeta, trialdefmeta, trialresults, wantmsgs )
%
% This function aggregates per-trial ephys data into per-probe ephys data.
% This is intended to be used with cropped downsampled data. Otherwise this
% will take up a very large amount of memory.
%
% This is intended to be called as a "probefuncafter" function, per
% ITERFUNCS.txt. A typical implementation would be:
%
% afterfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, ...
%     beforedata, trialresults, wantmsgs ) ...
%     [ destfolder filesep '%s-%s-seg-ephys.mat' ], ...
%     [ destfolder filesep '%s-%s-seg-meta.mat' ], ...
%     sessionmeta, probemeta, trialdefmeta, trialresults, wantmsgs );
%
% "outfilepat_ephys" is a sprintf pattern used to generate the output file
%   name for saving per-probe merged Field Trip data. This needs two '%s'
%   tokens, for the session label and probe label (in that order). This
%   file contains "ftdata" and "ftlabels_cooked", per PREPROCFILES.txt.
% "outfilepat_meta" is a sprintf pattern used to generate the output file
%   name for saving metadata for the merged ephys data. This needs two '%s'
%   tokens, for the session label and probe label (in that order). This file
%   contains several metadata structures, per PREPROCFILES.txt.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "trialresults" is a cell array indexed by trial row number that contains
%   the structures returned by epIter_trialFunc_epoch(). These contain
%   "ftdata" and "ftlabels_cooked" fields. The structures will be struct([])
%   for trials that were dropped.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% No return value.
```

3.4 epIter_beforeFunc_badListMergeInfo.m

```
% function badliststruct = epIter_beforeFunc_badListMergeInfo( ...
%   outfilepat, infilepat, wantforce, sessionmeta, probemeta, ...
%   configall, wantmsgs )
%
% This function builds bad channel lists based on precomputed analyses of
% channel badness metrics.
%
% This is intended to be called as a "probefuncbefore" function, per
% ITERFUNCS.txt. A typical implementation would be:
%
% beforefunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
%     epIter_beforeFunc_badListMergeInfo( ...
%       [ destfolder filesep '%s-%s-badlist.mat' ], ...
%       [ srcfolder filesep '%s-%s-badinfo.mat' ], ...
%       want_force_badlist, sessionmeta, probemeta, ...
%       bad_config_struct_all, wantmsgs );
%
% "outfilepat" is a sprintf pattern used to generate the output file name
%   for saving per-probe bad channel lists. This needs two '%s' tokens,
%   for the session label and probe label (in that order).
% "infilepat" is a sprintf pattern used to generate the input file name
%   for reading per-probe bad channel analysis output. This needs two '%s'
%   tokens, for the session label and probe label (in that order).
% "wantforce" is true to redo processing even if the output file already
%   exists, and false to skip processing if the output file is present.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "configall" is a configuration structure for bad channel analysis. Each
%   field corresponds to an algorithm and contains a configuration structure,
%   per BADCHANCONFIG.txt:
%   "log" contains configuration for hand-annotated log entries.
%   "spect" contains configuration information for spectral analysis.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% "badliststruct" is a structure with the following fields (corresponding
%   to variables written to the output file, per PREPROCFILES.txt):
%   "goodraw" (raw FT labels of channels highlighted as "good")
%   "badraw" (raw FT labels of channels highlighted as "bad")
%   "goodcooked" (cooked versions of "goodraw" labels)
%   "badcooked" (cooked versions of "badraw" labels)
%   "ftlabels_raw" (raw FT labels in data order, for channel mapping)
%   "ftlabels_cooked" (cooked FT labels in data order, for channel mapping)
%   "goodrawbymethod" (structure with per-algorithm lists of good channels)
%   "badrawbymethod" (structure with per-algorithm lists of bad channels)
```

3.5 epIter_beforeFunc_readBadChanList.m

```
% function badliststruct = ...
%   epIter_beforeFunc_readBadChanList( listfile, wantmsgs )
%
% This function reads a bad channel list metadata file and returns its
% contents in a structure.
%
% This is intended to be called as a "prebebeforefunc" function, per
% ITERFUNCS.txt. A typical implementation would be:
%
% beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
%   epIter_beforeFunc_readBadChanList( ...
%     sprintf(listfilepat, sessionmeta.sessionlabel, probemeta.label), ...
%     wantmsgs );
%
% "listfile" is the name of the file to read bad channel list metadata
%   from, per PREPROCFILES.txt.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% "badliststruct" is a structure containg all of the variables that were in
%   the bad channel list metadata file, per PREPROCFILES.txt.
```

3.6 epIter_beforeFunc_readChanMap.m

```
% function chanstruct = epIter_beforeFunc_readChanMap( chanfile, wantmsgs )
%
% This function reads channel map information and returns it in a structure.
%
% This is intended to be called as a "probebeforefunc" function, per
% ITERFUNCS.txt, or as part of such a function. A typical implementation
% would be:
%
% beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
%   epIter_beforeFunc_readChanMap( ...
%      sprintf(chanfilepat, sessionmeta.sessionlabel), wantmsgs );
%
% Or, to aggregate channel map information with other information:
%
% beforefunc = @(sessionmeta, probemeta, trialdefmeta, wantmsgs) ...
%   struct( 'chanmap', epIter_beforeFunc_readChanMap( ... ), ... );
%
%
% "chanfile" is the name of the name of the file to read channel map
%   metadata from, per PREPROCFILES.txt.
% "wantmsgs" is true to emit console messages and false otherwise.
%
```

```
% "chanstruct" is a structure with the following fields:
%   "chanlabels_raw" has unmapped channel labels from the map.
%   "chanlabels_cooked" has mapped channel labels from the map.
```

3.7 epIter_processSessions.m

```
% function epIter_processSessions( sessionlist, trialmetafilepat, ...
%   probefuncbefore, probefuncafter, trialfunc, want_parallel, want_messages )
%
% This iterates through sessions, probes, and trials, calling processing
% functions.
%
% The intention is that processing functions read input data from per-trial
% files and write output data to per-trial files, in most cases.
%
% This uses "parfor", which executes in parallel if the Parallel Computing
% Toolbox is installed and falls back to single-threaded operation if not.
%
% "sessionlist" is a struct array with session metadata, per SESSIONMETA.txt.
% "trialmetafilepat" is a sprintf pattern used when building per-session
%   trial metadata filenames. The pattern should have the form
%   'prefix-%s-suffix.mat', where '%s' is replaced with the session label.
%   This is typically '(folder)/%s-trialmeta.mat'.
% "probefuncbefore" is a function handle that's called during probe iteration,
%   before trial iteration. The function template is per ITERFUNCS.txt.
%   Passing NaN omits the call to this function.
% "probefuncafter" is a function handle that's called during probe iteration,
%   after trial iteration. The function template is per ITERFUNCS.txt.
%   Passing NaN omits the call to this function.
% "trialfunc" is a function handle that's called during trial iteration.
%   The function template is per ITERFUNCS.txt. Passing NaN omits the call
%   to this function (it actually omits trial iteration entirely).
% "want_parallel" is true to use Parallel Computing Toolbox multithreading
%   for trial iteration, and false not to.
% "want_messages" is true to emit progress messages and false not to.
%
% No return value.
```

3.8 epIter_trialFunc_badInfoAll.m

```
% function trialresult = epIter_trialFunc_badInfoAll( ...
%   outfilepat, infilepat, wantforce, sessionmeta, probemeta, ...
%   trialdefmeta, tidx, configall, wantmsgs )
%
% This function reads per-trial signal-conditioned ephys data and performs
% bad channel detection by various methods. The results structure contains
```

```
% raw analysis data from each algorithm (the "badchansXXX" fields described
% in PREPROCFILES.txt for "XXX-badinfo.mat").
%
% Algorithms are enabled by passing algorithm-specific configuration
% parameters as fields in "configall".
%
% NOTE - While this does not generate an output file itself, it does test
% for the presence of an output file to see if the analysis needs to be
% performed or can be skipped.
%
% This is intended to be called as a "trialfunc" function, per ITERFUNCS.txt.
% A typical implementation would be:
%
% trialfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
%     epIter_trialFunc_badInfoAll( ...
%       [ destfolder filesep '%s-%s-badinfo.mat' ], ...
%       [ srcfolder filesep '%s-%s-%s-ephysclean.mat' ], ...
%       want_force_badinfo, sessionmeta, probemeta, trialdefmeta, tidx, ...
%       bad_config_struct_all, wantmsgs );
%
% "outfilepat" is a sprintf pattern used to generate the output file name
%   for saving per-probe bad channel information. This needs two '%s' tokens,
%   for the session label and probe label (in that order).
%   NOTE - This function doesn't generate the output file. It tests for it
%   to skip analysis if the file exists and "wantforce" is false.
% "infilepat" is a sprintf pattern used to generate the input file name for
%   reading signal-conditioned trials in Field Trip format. This needs three
%   '%s' tokens, for the session label, probe label, and trial label (in that
%   order). The input file should contain "ftdata_clean" and
%   "ftlabels_cooked", per PREPROCFILES.txt.
% "wantforce" is true to redo processing even if the output file already
%   exists, and false to skip processing if the output file is present.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "tidx" is the row index of the present trial within trial metadata tables.
% "configall" is a configuration structure for bad channel analysis. Each
%   field corresponds to an algorithm and contains a configuration structure,
%   per BADCHANCONFIG.txt:
%   "log" contains configuration information for hand-annotated log entries.
%   "spect" contains configuration information for spectral analysis.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% "trialresult" is a structure with fields with the same field names as
%   "configall". Each field contains a structure holding analysis results
%   and metadata. Additional fields "ftlabels_raw" and "ftlabels_cooked"
%   store raw and cooked channel labels.
```

3.9 epIter_trialFunc_derived.m

```
% function trialresult = epIter_trialFunc_derived( ...
%   outfilepat, infilepat, wantforce, sessionmeta, probemeta, ...
%   trialdefmeta, tidx, badchans, derived_wanted, derived_config, wantmsgs )
%
% This function reads per-trial signal-conditioned ephys data and performs
% re-referencing, filtering, and rectification to produce derived signals.
% These are saved to disk in Field Trip format (per PREPROCFILES.txt).
%
% This is intended to be called as a "trialfunc" function, per ITERFUNCS.txt.
% A typical implementation would be:
%
% trialfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
%     epIter_trialFunc_derived( ...
%       [ destfolder filesep '%s-%s-%s-ephys%s.mat' ], ...
%       [ srcfolder filesep '%s-%s-%s-ephysclean.mat' ], ...
%       want_force_derived, sessionmeta, probemeta, trialdefmeta, tidx, ...
%       beforedata, derived_wanted_list, derived_config_struct, wantmsgs );
%
% "outfilepat" is a sprintf pattern used to generate the output file name
%   for saving Field Trip data. This needs four '%s' tokens, for the
%   session label, probe label, trial label, and signal type (in that order).
%   The output file will contain "ftdata_XXX" and "ftlabelscooked", per
%   PREPROCFILES.txt.
% "infilepat" is a sprintf pattern used to generate the input file name
%   for reading raw trials in Field Trip format. This needs three '%s'
%   tokens, for the session label, probe label, and trial label (in that
%   order). The input file should contain "ftdata_clean" and
%   "ftlabels_cooked", per PREPROCFILES.txt.
% "wantforce" is true to redo preprocessing even if the output files already
%   exist, and false to avoid overwriting existing output files.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "tidx" is the row index of the present trial within trial metadata tables.
% "badchans" is a structure containing all of the variables in the bad
%   channel list file, per PREPROCFILES.txt.
% "derived_wanted" is a cell array containing zero or more of 'wb', 'lfp',
%   'hp', and 'mua', per PREPROCFILES.txt.
% "derived_config" is a structure with the following fields (defined even
%   if the specified processing steps aren't requested):
%   "lfp_maxfreq" is the low-pass filter corner frequency for extracting LFP
%     signals.
%   "lfp_samprate" is the downsampled sampling rate for LFP signals.
%   "highpass_minfreq" is the high-pass filter corner frequency for
%     extracting spike waveforms.
```

```
%   "mua_band" [ min max ] specifies the band-pass filter corners for
%     extracting multi-unit activity prior to rectification.
%   "mua_lowpass" is the low-pass filter corner for smoothing multi-unit
%     activity after rectification.
%   "mua_samprate" is the downsampled sampling rate for MUA signals.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% "trialresult" is always NaN (unused).
```

3.10 epIter_trialFunc_epoch.m

```
% function trialresult = epIter_trialFunc_epoch( ...
%   infilepat, sigtype, sessionmeta, probemeta, trialdefmeta, tidx, ...
%   badchanmeta, newtrigtime, newtimespan, wantmsgs )
%
% This function reads per-trial ephys data and re-aligns and crops it.
% Aligned cropped ephys data is returned as the trial result (this can get
% big if aggregating full-sample-rate data).
%
% This is intended to be called as a "trialfunc" function, per ITERFUNCS.txt.
% A typical implementation would be:
%
% trialfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
%     epIter_trialFunc_epoch( ...
%       [ srcfolder filesep '%s-%s-%s-ephys%s.mat' ], 'mua', ...
%       sessionmeta, probemeta, trialdefmeta, tidx, ...
%       beforedata, new_trig_times(tidx), new_time_span, wantmsgs );
%
% "infilepat" is a sprintf pattern used to generate the input file name
%   for reading individual trial ephys data in Field Trip format. This needs
%   four '%s' tokens, for the session label, probe label, trial label, and
%   signal type (in that order). The input file should contain "ftdata_XXX"
%   and "ftlabels_cooked", per PREPROCFILES.txt. "XXX" is the signal type.
% "sigtype" is the signal type used when reading ephys data. This is
%   typically 'mua', but may be any other type listed in PREPROCFILES.txt.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "tidx" is the row index of the present trial within trial metadata tables.
% "badchanmeta" is a structure containing all of the variables in the bad
%   channel list metadata file, per PREPROCFILES.txt.
% "newtrigtime" is the revised trigger time, using the same time baseline
%   as the trigger times listed in "trialdefmeta.trialdeftable". If no
%   suitable trigger was present, this is NaN.
% "newtimespan" [ start stop ] is the desired timestamp range in seconds for
%   the cropped trials.
```

```
% "wantmsgs" is true to emit console messages and false otherwise.
%
% "trialresult" is a structure with the following fields, or struct([]) if
%   trial data was discarded:
%   "ftdata" is a ft_datatype_raw structure with the realigned cropped trial.
%   "ftlabels_cooked" has cooked channel labels corresponding to ftdata.label.
```

3.11 epIter_trialFunc_raw_FLToken2022.m

```
% function trialresult = epIter_trialFunc_raw_FLToken2022( ...
%   outfilepat, wantforce, sessionmeta, probemeta, trialdefmeta, tidx, ...
%   chanmapdata, wantmsgs )
%
% This function reads raw ephys data from a FLToken 2022-2023 session folder
% and saves the specified trial to disk in Field Trip Format (per
% PREPROCFILES.txt).
%
% This follows Charlie's conventions for channel labelling and ordering.
%
% This is intended to be called as a "trialfunc" function, per ITERFUNCS.txt.
% A typical implementation would be:
%
% trialfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
%     epIter_trialFunc_raw_FLToken2022( ...
%       [ destfolder filesep '%s-%s-%s-ephysraw.mat' ], want_force_raw, ...
%       sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs );
%
% "outfilepat" is a sprintf pattern used to generate the output file name
%   for saving Field Trip data. This needs three '%s' tokens, for the
%   session label, probe label, and trial label (in that order). The output
%   file will contain "ftdata_raw" and "ftlabels_cooked", per PREPROCFILES.txt.
% "wantforce" is true to redo processing even if the output file already
%   exists, and false to skip processing if the output file is present.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "tidx" is the row index of the present trial within trial metadata tables.
% "chanmapdata" is a structure with the following fields:
%   "chanlabels_raw" has unmapped channel labels from the map.
%   "chanlabels_cooked" has mapped channel labels from the map.
% "wantmsgs" is true to emit console messages and false otherwise.
%
% "trialresult" is always NaN (unused).
```

3.12 epIter_trialFunc_sigclean.m

```
% function trialresult = epIter_trialFunc_sigclean( ...
%   outfilepat, infilepat, wantforce, sessionmeta, probemeta, ...
%   trialdefmeta, tidx, passes, artfunc, notchfreqs, notchbw, wantmsgs )
%
% This function reads per-trial raw ephys data and performs notch filtering
% and artifact removal. Clean trials are saved to disk in Field Trip format
% (per PREPROCFILES.txt).
%
% This accepts an arbitrary artifact rejection function. This function may
% remove curve-fit artifacts and/or may replace artifact regions with NaN.
%
% This is intended to be called as a "trialfunc" function, per ITERFUNCS.txt.
% A typical implementation would be:
%
% trialfunc = ...
%   @(sessionmeta, probemeta, trialdefmeta, tidx, beforedata, wantmsgs) ...
%     epIter_trialFunc_sigclean( ...
%       [ destfolder filesep '%s-%s-%s-ephysclean.mat' ], ...
%       [ srcfolder filesep '%s-%s-%s-ephysraw.mat' ], want_force_clean, ...
%       sessionmeta, probemeta, trialdefmeta, tidx, ...
%       clean_passes, artifact_func, notch_freqs, notch_bw, wantmsgs );
%
% "outfilepat" is a sprintf pattern used to generate the output file name
%   for saving Field Trip data. This needs three '%s' tokens, for the
%   session label, probe label, and trial label (in that order). The output
%   file will contain "ftdata_clean" and "ftlabels_cooked", per
%   PREPROCFILES.txt.
% "infilepat" is a sprintf pattern used to generate the input file name
%   for reading raw trials in Field Trip format. This needs three '%s'
%   tokens, as with "outfilepat". The input file should contain "ftdata_raw"
%   and "ftlabels_cooked", per PREPROCFILES.txt.
% "wantforce" is true to redo processing even if the output file already
%   exists, and false to skip processing if the output file is present.
% "sessionmeta" is a single session metadata structure, per SESSIONMETA.txt.
% "probemeta" is a probe definition structure, per PROBEDEFS.txt.
% "trialdefmeta" is a structure containing all of the variables in the trial
%   definition metadata file, per PREPROCFILES.txt.
% "tidx" is the row index of the present trial within trial metadata tables.
% "passes" is the number of processing passes to make (typically 2). More
%   passes reduces the amount of tone filter ringing around artifacts.
% "artfunc" is a function handle to call for artifact removal. This has the
%   form:   newtrial = artfunc( fttime, fttrial )
%   ...where "fttime" is a 1xNsamples vector and "fttrial" and "newtrial" are
%   Nchans*Nxamples matrices. If "artfunc" is NaN, no function call is made.
% "notchfreqs" is a vector with notch filter frequencies, or [] to disable.
% "notchbw" is the notch filter bandwidth in Hz, or NaN to disable filtering.
% "wantmsgs" is true to emit console messages and false otherwise.
```

```
%
% "trialresult" is always NaN (unused).
```