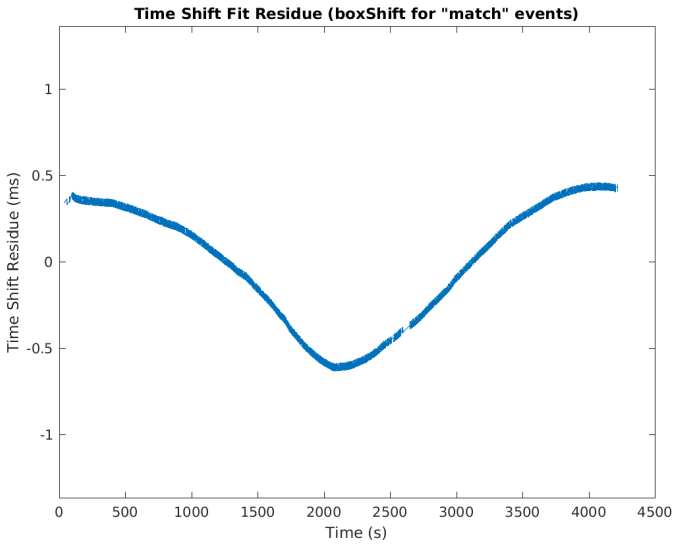
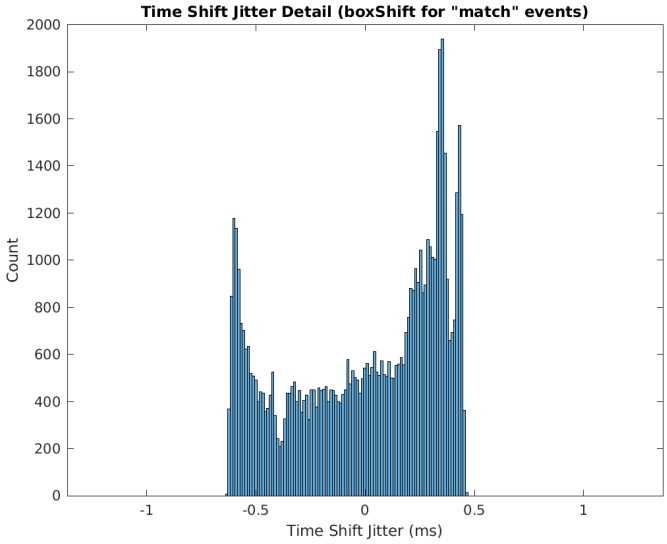
Event Code Alignment Test Script Code Reference

Written by Christopher Thomas – March 4, 2022.





Contents

Chapter 1

Overview

1.1 README.md

```
# Event Code Alignment Test Script

## Remarks

This is a script that was used to test low-level event code decoding and
time alignment functions during development.

Those functions have been packaged in ‘lib-exputils-align‘, and this script
was rewritten to use high-level function calls and Field Trip where
feasible.

While this script could be used as sample code, and is still used for
regression testing, you’re better off looking at the ‘ft_test‘ folder
for a more comprehensive example script.


## Documentation

The ‘manual‘ directory is the LaTeX build directory for documentation for
this script. Use ‘make -C manual‘ to rebuild it.

Right now this is mostly just an automated compilation of the source code.


## Notes

Notes about the sources of misalignment:

* Time misalignment is dominated by linear drift (a ramp) caused by clock
crystal frequency differences.
    * Crystal frequency mismatch is generally 0.1% to 0.2%; much wider than
```

the nominal precision of the crystals.
* With the ramp subtracted, you'll see the time delta drift on a timescale of tens of minutes due to temperature changes next to the crystals.
    * Most devices use crystals with stability rated to 100 ppm. The Intan machines use crystals with stability rated to 10 ppm. Temperature drift will stay within this range.
* On short timescales, you'll see jitter (fuzz).
    * Plotting a histogram of jitter for computer sources typically gives you a multi-peak distribution with peaks that are 0.1-0.2 ms wide and separated by 1-2 ms. These are the precision of the OS's scheduler and the polling interval for the OS's I/O drivers, respectively.
    * Plotting the histogram for the SynchBox gives a single-mode distribution with a width of 0.1 ms (the SynchBox's scheduling interval).
    * Plotting it for the Intan machines gives a single-mode distribution with a width of 0.03 or 0.05 ms (the sampling interval).

Notes about how the scripts perform time alignment:

* The goal is to align timestamps from different devices to within one sample (0.03 ms).
* This is done by subtracting a global time offset, then getting successively better estimates of the time difference between known reference points (events that we have timestamps for from multiple devices).
* If you try to do this the easy way (linear interpolation between known points), you'll be limited by jitter (fuzz) to plus or minus several samples.
* The point of the fine-tuning functions is to perform a fit in a neighbourhood that contains several points, to average out this fuzz.
* There are other ways to do time alignment. This is just the one that I used for my own implementation.

(This is the end of the file.)

Chapter 2

Top-Level Scripts

2.1 do_test.m

```
% Quick and dirty test script for event code time alignment.
% Written by Christopher Thomas.

%
% Includes

% Add root paths.
addpath('lib-exp-utils-cjt');
addpath('lib-looputil');
addpath('lib-fieldtrip');
addpath('openephys-tools');
addpath('npy-matlab');

% Add sub-folders.
addPathsExpUtilsCjt;
addPathsLoopUtil;

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


%
% Behaviour switches.

% Time alignment steps to perform.
want_align_synchbox_ephys = true;
want_align_unity_ephys = true;

% This is used for USB jitter measurement plots and not much else.
want_align_unity_synchbox = true;
```

```
% Behavior switch for time alignment (suppresses data value matching).
want_blind_alignment = false;

% This removes unmatched event code entries from their respective tables.
want_remove_unmatched = false;


% Verbosity levels are 'quiet', 'normal', and 'verbose'.
%verbosity_codes = 'normal';
verbosity_codes = 'verbose';
verbosity_align = 'normal';

% Canonical event code source.
code_source = 'ephys';
%code_source = 'synchbox';
%code_source = 'unity';



%
%
% Constants

% Output folders.
folder_plots = 'plots';


% This should be the folder with "SerialData.mat" and other processed data.
folder_usedata = 'datasets/exp-si-20210914-use/ProcessedData';

% This should be the "SessionSettings" folder in the USE tree.
folder_usesettings = ...
  'datasets/exp-si-20210914-use/RuntimeData/SessionSettings';


% This should be the folder with "settings.xml".
folder_ephysroot = 'datasets/exp-si-20210914-openephys';

% If we're using binary format, this should be the folder with
% "structure.oebin". Otherwise it should be the empty string.
%folder_ephysbinary = '';
folder_ephysbinary = sprintf('%s/experiment1/recording1', folder_ephysroot);


% Lookup table for building event codes.
% Bits that don't contribute should be set to 0.
% This is Matlab's default fill value, so we can do that implicitly.
evcode_ttl_bits = [];
evcode_ttl_bits(9) = 0x01;
evcode_ttl_bits(10) = 0x02;
```

```
evcode_ttl_bits(11) = 0x04;
evcode_ttl_bits(12) = 0x08;
evcode_ttl_bits(13) = 0x10;
evcode_ttl_bits(14) = 0x20;
evcode_ttl_bits(15) = 0x40;
evcode_ttl_bits(16) = 0x80;


% Overrides table for ranged event codes.
% NOTE - BlockCondition is 501..599. Leave it alone; it appears in
% other data files as-is.
evcode_overrides = struct( ...
  'Dimensionality', struct('offset', 200), ...
  'RewardValidity', struct('offset', 100), ...
  'TrialIndex', struct('offset', 4000), ...
  'TrialNumber', struct('offset', 12000), ...
  'TokensAdded', struct('offset', 70) );

% Event code encoding.
bytespercode = 2;
codeendian = 'big';


% Time alignment configuration.

% Columns to pull data from, for data matching during alignment.
firstdatacol = 'codeValue';
seconddatacol = 'codeValue';

% We have the option of ignoring data and matching on timestamps alone.
if want_blind_alignment
  firstdatacol = '';
  seconddatacol = '';
end


% Time alignment tuning parameters.

% Worst-case drift is 1000 ppm. Worst-case trace length is 1e+4 sec.
% Frames last 17 ms. Once we're localized to within half a frame or better,
% we can use the fine search (which freezes mapping).

% Global fixed-offset pass gives 10 sec accuracy.
% Coarse pass uses a 100 sec window and gives 0.1 sec accuracy.
% Medium pass uses a 1 sec window and gives 1 ms accuracy.
% (In practice this might be limited to one-frame accuracy, depending on
% which timestamps are being aligned.)
% Fine pass uses a 100 ms window and gives 0.1 ms accuracy.

% For coarse alignment windows, the window slides in steps equal to the
```

```
% window radius. We pick one representative sample near the middle of the
% window and consider alignment candidates for it.

% An alignment using a constant global delay is performed using the first
% coarse window value, in addition to sliding-window alignment.

% For medium alignment windows, each sample in turn is used as the center
% of the sliding window, and alignment candidates are considered for the
% central sample.

% For fine alignment, each sample in turn is used as the center of the
% sliding window, and all samples within the window are matched against
% their nearest candidates. Time offset is optimized to minimize the cost
% of all of these matchings. The resulting offset is taken as the center
% sample's true time offset.

coarsewindows = [ 100.0 ];
medwindows = [ 1.0 ];
finewindow = 0.1;

% A filtering pass is performed to remove outlier time-deltas.
% These would otherwise substantially skew time estimates around them.
outliersigma = 4.0;



%
%
% Main Program


%
%
% First step: Load event-related data.


%
% Load and parse the USE data.

disp('-- Loading SynchBox data...');

load( [ folder_usedata filesep 'SerialData.mat' ], 'serialData');
[ boxsynchA boxsynchB boxrwdA boxrwdB boxcodes ] = ...
  euUSE_parseSerialRecvData(serialData.SerialRecvData, 'dupbyte');
[ gamerwdA gamerwdB gamecodes ] = ...
  euUSE_parseSerialSentData(serialData.SerialSentData, 'dupbyte');


% FIXME - FrameData contains columns of interest:
% EventCodes, SplitEventCodes, PreSplitEventCodes, ArduinoTimeStamp,
```

```
% FrameStartUnity, FrameStartSystem


% Load the event code definition JSON file and process it.

disp('-- Loading event code definitions...');

fname = [ folder_usesettings filesep 'eventcodes_USE_05.json' ];
evcodedefsraw = jsondecode(fileread(fname));

% This parses individual codes and "Min"/"Max" range codes.
evcodedefs = euUSE_parseEventCodeDefs(evcodedefsraw, evcode_overrides);



%
% Load the TTL data and assemble event code bytes.

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

% FIXME - Suppress NPy warnings.
warnstate = warning('off');


% The header gives us the sampling rate.
rechdr = ...
  ft_read_header(folder_ephysbinary, 'headerformat', 'nlFT_readHeader');

% This reads all events.
recevents_dig = ...
  ft_read_event( folder_ephysbinary, 'headerformat', 'nlFT_readHeader', ...
    'eventformat', 'nlFT_readEvents' );

% We only care about events from the 'DigWordsA_000' channel.
evchans = { recevents_dig.type };
evmask = strcmp(evchans, 'DigWordsA_000');
recevents_dig = recevents_dig(evmask);


% FIXME - Restore warnings.
warning(warnstate);


% Extract relevant parts of the data.

disp('-- Assembling event codes...');

ttlsamprate = rechdr.Fs;

ttlrawdata = [ recevents_dig.value ];
```

```
ttlrawtimes = [ recevents_dig.sample ];

% We have the correct bit order, but we only need the top 8 bits.
% We know that the original data is 16-bit.
evcodevals = bitshift(ttlrawdata, -8, 'uint16');

% Timestamps are fine as-is.
evcodesamps = ttlrawtimes;

% Merge codes that repeat the same timestamp or that are one sample apart.
[evcodevals, evcodesamps] = euUSE_deglitchEvents(evcodevals, evcodesamps);

% Remove codes with value 0, as that's the idle state.
keepidx = (evcodevals > 0);
evcodesamps = evcodesamps(keepidx);
evcodevals = evcodevals(keepidx);

% Get timestamps in seconds.
evcodetimes = evcodesamps;
if (~isnan(ttlsamprate)) && (~isempty(evcodesamps))
  evcodetimes = evcodesamps / ttlsamprate;
end


% Turn this into a data table.
% NOTE - Make sure we're dealing with column vectors when building the table.

ephyscodes = table();
if ~isempty(evcodevals)
  if isrow(evcodevals) ; evcodevals = transpose(evcodevals); end
  if isrow(evcodesamps) ; evcodesamps = transpose(evcodesamps); end
  if isrow(evcodetimes) ; evcodetimes = transpose(evcodetimes); end

  if ~isnan(ttlsamprate)
    % We have timestamps in seconds as well.
    ephyscodes = table( evcodevals, evcodesamps, evcodetimes, ...
      'VariableNames', {'codeValue', 'ephysSample', 'ephysTime'} );
  else
    % We only have sample indices.
    ephyscodes = table( evcodevals, evcodesamps, ...
      'VariableNames', {'codeValue', 'ephysSample'} );
  end
end



%
% FIXME - Remove enormous offsets from time series.

% The real problem is the Unity timestamps, which are relative to 1 Jan 1970.
```

```
% So, pick a common offset for each _type_ of timestamp, and subtract that
% offset from all relevant table columns.

% FIXME - Just leave synchbox and ephys as-is.
synchboxoffset = 0;
ephysoffset = 0;

% Calculate a Unity timestamp offset.
% Negative values are okay, so just pick from one series.
unityoffset = min(gamecodes.unityTime);

% Apply the Unity timestamp offset.

boxcodes.unityTime = boxcodes.unityTime - unityoffset;
gamecodes.unityTime = gamecodes.unityTime - unityoffset;

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

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

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



%
%
% Second step: Do time alignment between SynchBox and ephys.

% This adds ephys timestamps to the SynchBox table and vice versa.

% This has to use the raw event bytes to get decent precision.
% Enough bytes are dropped that we get horrible mismatches with cooked
% event code words.

% Do Unity/SynchBox jitter measurements in this step too.
```

```
times_synchbox_ephys = table();

if want_align_synchbox_ephys

  %
  % SynchBox times (from replies to Unity) vs ephys times.

  % This is dominated by long-term drift in the SynchBox's clock crystal.


  disp('-- Aligning SynchBox with Ephys.');

  [ boxcodes, ephyscodes, boxmatchmask, boxephysmatchmask, ...
    times_synchbox_ephys ] = ...
    euAlign_alignTables( boxcodes, ephyscodes, ...
      'synchBoxTime', 'ephysTime', firstdatacol, seconddatacol, ...
      coarsewindows, medwindows, finewindow, ...
      outliersigma, verbosity_align );


  % Report event hit/miss rate.

  matchcount = sum(boxmatchmask);
  misscountfirst = sum(~boxmatchmask);
  misscountsecond = sum(~boxephysmatchmask);

  disp(sprintf( ...
    '.. Matched %d events; missed %d SynchBox and %d ephys.', ...
    matchcount, misscountfirst, misscountsecond ));


  % Add a time shift column. This is just ephysTime - synchBoxTime.

  boxcodes.('boxShift') = boxcodes.ephysTime - boxcodes.synchBoxTime;
  ephyscodes.('boxShift') = ephyscodes.ephysTime - ephyscodes.synchBoxTime;


  % Make plots.

  disp('-- Plotting.');

  % SynchBox timestamps vs Ephys timestamps.
  % FIXME - 2-sigma is good with raw codes, but as soon as we reassemble
  % them we get a multimodal distribution that needs 4-sigma.

  % Get match and miss tables.
  eventsboxmatch = boxcodes(boxmatchmask, :);
  eventsboxmiss = boxcodes(~boxmatchmask, :);
```

```
  eventsboxopenmiss = ephyscodes(~boxephysmatchmask, :);

  doPlotTimeStats( sprintf('%s/test-boxreply', folder_plots), ...
    struct( 'match', eventsboxmatch, ...
      'synchmiss', eventsboxmiss, 'ephysmiss', eventsboxopenmiss), ...
    { 'synchmiss', 'ephysmiss' }, ...
    struct( 'synchbox', 'synchBoxTime', 'ephys', 'ephysTime' ), ...
    struct( 'delta', ...
      struct('timelabel', 'ephysTime', 'deltalabel', 'boxShift') ), ...
    4.0 );


  % Remove unmatched entries, if requested.

  if want_remove_unmatched
    boxcodes = boxcodes(boxmatchmask,:);
    ephyscodes = ephyscodes(boxephysmatchmask,:);
  end

end


if want_align_unity_synchbox
  %
  % Unity times vs SynchBox times (from SynchBox replies to Unity).

  % This has multimodal jitter due to buffering in the USB serial link.
  % That's pretty much what this test is intended to measure.

  disp('-- Measuring SynchBox-to-Unity communications jitter.');

  scratchtab = boxcodes;
  scratchtab.('serialShift') = ...
    scratchtab.('synchBoxTime') - scratchtab.('unityTime');

  doPlotTimeStats( sprintf('%s/test-unityloopback', folder_plots), ...
    struct( 'boxreply', scratchtab ), {}, ...
    struct( 'unity', 'unityTime', 'synchbox', 'synchBoxTime' ), ...
    struct( 'delta', ...
      struct('timelabel', 'synchBoxTime', 'deltalabel', 'serialShift') ), ...
    4.0 );
end



%
%
% Third step: Reconstruct and translate event codes.

disp('-- Rebuilding event codes.');
```

```
[ ephyscodes ephyscodeindices ] = euUSE_reassembleEventCodes( ...
  ephyscodes, evcodedefs, bytespercode, codeendian, 'codeValue' );
[ boxcodes boxcodeindices ] = euUSE_reassembleEventCodes( ...
  boxcodes, evcodedefs, bytespercode, codeendian, 'codeValue' );
[ gamecodes gamecodeindices ] = euUSE_reassembleEventCodes( ...
  gamecodes, evcodedefs, bytespercode, codeendian, 'codeValue' );

% Diagnostics.
disp(sprintf( ...
  '... Found %d ephys codes, %d SynchBox codes, %d Unity codes.', ...
  length(ephyscodeindices), length(boxcodeindices), ...
  length(gamecodeindices) ));


% Copy the canonical set of codes for statistics reporting.

cookedcodes = table();

if strcmp(code_source, 'ephys')
  cookedcodes = ephyscodes;
elseif strcmp(code_source, 'synchbox')
  cookedcodes = boxcodes;
elseif strcmp(code_source, 'unity')
  cookedcodes = gamecodes;
else
  disp(sprintf( '### Unknown event code source "%s"; using ephys codes.', ...
    code_source ));
  code_source = 'ephys';
  cookedcodes = ephyscodes;
end


% Get statistics for the canonical code list.

[ totalcount goodcount goodunique gooddesc ...
  badcount badunique baddesc ] = ...
  doGetEventStatistics( cookedcodes, sprintf('%s/test', folder_plots) );

if totalcount < 1
  disp(sprintf( '.. No codes found in code source "%s".', code_source ));
elseif ~strcmp('quiet', verbosity_codes)

  disp(sprintf( ...
    '... Code source "%s" has %d good codes and %d bad codes (total %d)', ...
    code_source, goodcount, badcount, totalcount ));

  % Statistics on bad codes.
  if badcount > 0
    disp(sprintf( '... %d unique "bad" codes.', length(badunique) ));
```

```
    if strcmp('verbose', verbosity_codes)
      % Use fprintf, not disp(), since we have trailing newlines.
      fprintf('%s', baddesc);
    end
  end

  % Statistics on good codes.
  if goodcount > 0
    disp(sprintf( '... %d unique "good" codes.', length(goodunique) ));

    if strcmp('verbose', verbosity_codes)
      % Use fprintf, not disp(), since we have trailing newlines.
      fprintf('%s', gooddesc);
    end
  end

end



%
%
% Fourth step: Do time alignment between Unity and ephys.

% This adds Unity timestamps to the ephys table and vice versa.

times_unity_ephys = table();

if want_align_unity_ephys

  %
  % Unity times (from commands to synchbox) vs ephys times.

  % This is surprisingly clean; sub-ms accuracy with few outliers.

  disp('-- Aligning Unity with Ephys.');

  [ gamecodes, ephyscodes, gamematchmask, gameephysmatchmask, ...
    times_unity_ephys ] = ...
    euAlign_alignTables( gamecodes, ephyscodes, ...
      'unityTime', 'ephysTime', firstdatacol, seconddatacol, ...
      coarsewindows, medwindows, finewindow, outliersigma, ...
      verbosity_align );


  % Report event hit/miss rate.

  matchcount = sum(gamematchmask);
  misscountfirst = sum(~gamematchmask);
```

```
  misscountsecond = sum(~gameephysmatchmask);

  disp(sprintf( '.. Matched %d events; missed %d Unity and %d ephys.', ...
    matchcount, misscountfirst, misscountsecond ));


  % Add a time shift column. This is just ephysTime - unityBoxTime.

  gamecodes.('unityShift') = gamecodes.ephysTime - gamecodes.unityTime;
  ephyscodes.('unityShift') = ephyscodes.ephysTime - ephyscodes.unityTime;


  % Make plots.

  disp('-- Plotting.');

  % Unity timestamps vs Ephys timestamps.

  % Get match and miss tables.
  eventsgamematch = gamecodes(gamematchmask, :);
  eventsgamemiss = gamecodes(~gamematchmask, :);
  eventsgameopenmiss = ephyscodes(~gameephysmatchmask, :);

  doPlotTimeStats( sprintf('%s/test-unitysend', folder_plots), ...
    struct( 'match', eventsgamematch, ...
      'unitymiss', eventsgamemiss, 'ephysmiss', eventsgameopenmiss), ...
    { 'unitymiss', 'ephysmiss' }, ...
    struct( 'unity', 'unityTime', 'ephys', 'ephysTime' ), ...
    struct( 'delta', ...
      struct('timelabel', 'ephysTime', 'deltalabel', 'unityShift') ), ...
    4.0 );


  % Remove unmatched entries, if requested.

  if want_remove_unmatched
    gamecodes = gamecodes(gamematchmask,:);
    ephyscodes = ephyscodes(gameephysmatchmask,:);
  end

end



%
%
% Fifth step: Copy the canonical set of events, if we have one.

codesaligned = table();
```

```
if strcmp(code_source, 'ephys')
  codesaligned = ephyscodes;
elseif strcmp(code_source, 'synchbox')
  codesaligned = boxcodes;
elseif strcmp(code_source, 'unity')
  codesaligned = gamecodes;
else
  % This shouldn't happen; we'd have caught it the first time around.
  disp(sprintf( '### Unknown event code source "%s"; using ephys codes.', ...
    code_source ));
  code_source = 'ephys';
  codesaligned = ephyscodes;
end

% Complain if we didn't store anything.
if isempty(codesaligned)
  disp('###  Warning - No time-aligned events!');
end


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

Chapter 3

Helper Functions

3.1 doGetEventStatistics.m

```
function [ totalcount goodcount goodunique gooddesc ...
  badcount badunique baddesc ] = doGetEventStatistcs( ...
  codetable, obase )

% function [ totalcount goodcount goodunique gooddesc ...
%   badcount badunique baddesc ] = doGetEventStatistcs( ...
%   codetable, obase )
%
% This returns a human-readable digest of event code statistics. If a
% non-empty output filename is supplied, these are also written to text files.
%
% "codetable" is a table of reassembled event codes with the format
%   described in reassembleCodes(). Columns read are:
%   "codeWord" contains the code's integer value.
%   "codeData" is (code number - offset) * multiplier, per "EVCODEDEFS.txt".
%   "codeLabel" is the code definition field name for this code, if the
%     code was recognized, or '' if the code was not recognized. This is
%     usually a human-readable code name.
% "obase" is a prefix used when constructing output filenames. Use '' to
%   suppress file output.
%
% "totalcount" is the total number of events in "codetable".
% "goodcount" is the total number of events with recognized codes.
% "goodunique" is a cell array containing a list of labels associated with
%   recognized events. Each label is present only once.
% "gooddesc" is a character array containing a human-readable table of
%   event counts and data ranges for each event code label recognized. This
%   table contains newlines.
% "badcount" is the total number of events that were not recognized.
% "badunique" is a vector containing the code words that weren't recognized.
%   each code word is present only once.
```

```
% "baddesc" is a character array containing a human-readable table of event
%   counts and code word values for each unrecognized code word.


% Initialize to "nothing found".

totalcount = 0;

goodcount = 0;
goodunique = {};
gooddesc = '';

badcount = 0;
badunique = [];
baddesc = '';


if ~isempty(codetable)

  % Extract labels and see how many good/bad events we have.

  alllabels = codetable.codeLabel;
  allwords = codetable.codeWord;
  alldata = codetable.codeData;

  badmask = strcmp(alllabels, '');
  goodmask = ~badmask;

  totalcount = length(alllabels);

  badcount = sum(badmask);
  goodcount = sum(goodmask);


  % Process the "bad" list.

  if badcount > 0

    % We only care about the code words.
    % Label is always '' and data is always equal to the code word.

    badwords = allwords(badmask);
    badunique = unique(badwords);

    baddesc = '';
    for bidx = 1:length(badunique)
      thisbad = badunique(bidx);
      thiscount = sum(badwords == thisbad);
      thisdesc = sprintf( '%6d - %d times\n', thisbad, thiscount );
      baddesc = [ baddesc thisdesc ];
```

```
  end

  % Write output if requested.
  if ~isempty(obase)
    helper_writeFile( sprintf('%s-codes-bad.txt', obase), baddesc );
  end
end


% Process the "good" list.

if goodcount > 0

  goodlabels = alllabels(goodmask);
  goodwords = allwords(goodmask);
  gooddata = alldata(goodmask);

  % We're going to see a lot of different values for ranged codes, so
  % walk through unique labels rather than unique code words.

  % Sorting and "unique" work just fine with string data.
  goodunique = unique(goodlabels);

  gooddesc = '';
  for gidx = 1:length(goodunique)
    thisgoodlabel = goodunique{gidx};
    thisgoodmask = strcmp(goodlabels, thisgoodlabel);
    thiscount = sum(thisgoodmask);

    % Extract data range.
    thisgooddata = gooddata(thisgoodmask);
    thisgooddata = unique(thisgooddata);

    thisdesc = '';
    if length(thisgooddata) == 1
      thisdesc = sprintf( '%24s - %d times  (data: %d)\n', ...
        thisgoodlabel, thiscount, thisgooddata );
    else
      thisdesc = sprintf( '%24s - %d times  (data: %d to %d)\n', ...
        thisgoodlabel, thiscount, min(thisgooddata), max(thisgooddata) );
    end

    gooddesc = [ gooddesc thisdesc ];
  end

  % Write output if requested.
  if ~isempty(obase)
    helper_writeFile( sprintf('%s-codes-good.txt', obase), gooddesc );
  end
```

```
  end

end


% Done.

end


%
% Helper functions.


function helper_writeFile( fname, ftext )

  fid = fopen(fname, 'w');
  if fid < 0
    disp(sprintf( '### Unable to write to "%s".', fname ));
  else
    % This is text data, so write it as text, not bytes (in case of
    % peculiar encoding).

    fprintf( fid, '%s', ftext );

    fclose(fid);
  end

end


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

3.2 doPlotTimeStats.m

```
function doPlotTimeStats( obase, tabstruct, misslabels, ...
  timelabelstruct, shiftlabelstruct, jitterzoom )

% function doPlotTimeStats( obase, tabmatch, taball, ...
%   misstabstruct, timelabelstruct, shiftlabelstruct, jitterzoom )
%
% This function generates several plots and statistics related to event
% timing.
%
% NOTE - Time is assumed to be in seconds, for purposes of precision and
% labels.
%
```

```
% "obase" is the prefix to use when constructing plot filenames.
% "tabstruct" is a structure containing tables with data tuples for
%   events from various source lists. Structure field names are file labels
%   to use for the contained tables.
% "misslabels" is a cell array containing field names in "tabstruct" that
%   correspond to "missed" events. These get a different set of plots.
% "timelabelstruct" is a structure containing the names of table columns that
%   have time data to be plotted. Structure field names are file labels to
%   use when plotting times taken from the corresponding columns.
% "shiftlabelstruct" is a structure with field names that are file labels
%   to use when plotting alignment shifts. Each field contains a structure
%   with a "timelabel" field (containing the column name to read time series
%   data from) and a "deltalabel" field (containing the column name to read
%   alignment time-shifts from).
% "jitterzoom" is the radius of the window to plot jitter details in, in
%   standard deviations. Typical values are 2-4.


tablabels = fieldnames(tabstruct);
timelabels = fieldnames(timelabelstruct);
shiftlabels = fieldnames(shiftlabelstruct);

scratchfig = figure();


% Walk through the list of data tables to plot.

for tabidx = 1:length(tablabels)

  thistablabel = tablabels{tabidx};
  thistab = tabstruct.(thistablabel);

  if ismember(thistablabel, misslabels)

    % This is a list of "missed" events.
    % We want to generate histograms of when these happened.

    % Walk through the time series list.
    for timeidx = 1:length(timelabels)
      % Get metadata.
      thistimelabel = timelabels{timeidx};
      thistimecol = timelabelstruct.(thistimelabel);

      helper_plotEventTimes( thistab.(thistimecol), scratchfig, ...
        sprintf('%s time for "%s" misses', thistimelabel, thistablabel), ...
        sprintf('%s-times-%s-%s.png', obase, thistablabel, thistimelabel) );
    end

  else
```

```
    % This is an ordinary event list.
    % Plot time drift and jitter statistics.

    % Walk through the time-shift list.
    for shiftidx = 1:length(shiftlabels)

      % Get metadata.
      thisshiftlabel = shiftlabels{shiftidx};
      thisshiftentry = shiftlabelstruct.(thisshiftlabel);
      thistimecol = thisshiftentry.timelabel;
      thisdeltacol = thisshiftentry.deltalabel;

      % Generate drift and jitter statistics.
      helper_reportStats( ...
        thistab.(thistimecol), thistab.(thisdeltacol), ...
        sprintf('%s-stats-%s-%s.txt', obase, thistablabel, thisshiftlabel) );

      % Generate time-shift plots.
      helper_plotTimeShift( ...
        thistab.(thistimecol), thistab.(thisdeltacol), ...
        jitterzoom, scratchfig, ...
        sprintf('%s for "%s" events', thisdeltacol, thistablabel), ...
        sprintf('%s-shift-%s-%s', obase, thistablabel, thisshiftlabel) );

    end

  end

end


close(scratchfig);


% Done.

end


%
% Helper functions.


% This computes alignment/drift/jitter, and also 7-figure stats for
% jitter after first-order (ramp) and second-order (bow) fitting.
%
% "timeseries" is a sequence of event time values.
% "deltaseries" is a sequence of alignment time-shift values.
% "fname" is the name of the text file to write to.
```

```
function helper_reportStats(timeseries, deltaseries, fname)

  % Only perform fits if we have at least 3 data points.

  if length(timeseries) >= 3

    % Get ramp fit.

    lincoeffs = polyfit(timeseries, deltaseries, 1);
    linvals = polyval(lincoeffs, timeseries);
    linresidue = deltaseries - linvals;

    rampcoeff = lincoeffs(1);
    constcoeff = lincoeffs(2);

    % Get quadratic fit.

    quadcoeffs = polyfit(timeseries, deltaseries, 2);
    quadvals = polyval(quadcoeffs, timeseries);
    quadresidue = deltaseries - quadvals;


    % Try to open the output file.

    fid = fopen(fname, 'w');
    if fid < 0
      disp(sprintf( '### Unable to write to "%s".', fname ));
    else

      % Compute and report stats.
      % FIXME - Assuming seconds!

      % General statistics.
      fprintf( fid, ...
        'Mean align %.4f s, drift %.1f ppm, jitter %.3f ms.\n', ...
        constcoeff, rampcoeff * 1e6, std(linresidue) * 1e3 );

      % 7-figure summary for line fit.
      fprintf( fid, 'Line fit jitter stats (deviation from ramp):\n' );
      percvec = 1e3 * prctile(linresidue, [ 2 9 25 50 75 91 98 ]);
      fprintf( fid, ...
        '  %.2f / %.2f / %.2f / [ %.2f ] / %.2f / %.2f / %.2f\n', ...
        percvec(1), percvec(2), percvec(3), percvec(4), ...
        percvec(5), percvec(6), percvec(7) );

      % 7-figure summary for quadratic fit.
      fprintf( fid, 'Quadratic fit jitter stats (deviation from bow):\n' );
      percvec = 1e3 * prctile(quadresidue, [ 2 9 25 50 75 91 98 ]);
      fprintf( fid, ...
        '  %.2f / %.2f / %.2f / [ %.2f ] / %.2f / %.2f / %.2f\n', ...
```

```
        percvec(1), percvec(2), percvec(3), percvec(4), ...
        percvec(5), percvec(6), percvec(7) );

      % Finished with this file.
      fclose(fid);

    end

  end

end


% This plots time shift vs time, and a histogram of time jitter.
%
% "timeseries" is a sequence of event time values.
% "deltaseries" is a sequence of alignment time-shift values.
% "thisfig" is a figure handle to use for plotting.
% "caselabel" is a string to annotate the plot title with.
% "fbase" is a prefix to use when building plot filenames.

function helper_plotTimeShift( timeseries, deltaseries, coredeviations, ...
  thisfig, caselabel, fbase )

  % Select this figure.
  figure(thisfig);


  % FIXME - Configuration.
  % Polynomial fit order. Usually this is 1 (ramp) or 2 (bow).
  fitorder = 1;


  % Only make plots if we have at least two data points.
  if length(timeseries) >= 2

    % Plot time shift against time.

    clf('reset');

    plot(timeseries, deltaseries, 'HandleVisibility', 'off');

    title(sprintf('Time Shift (%s)', caselabel));
    xlabel('Time (s)');
    ylabel('Time Shift (s)');

    saveas(thisfig, sprintf('%s-plot.png', fbase));


    % Calculate a fit and the fit residue.
```

```
% This is usually a ramp fit (first-order) or bow (second-order).

fitcoeffs = polyfit(timeseries, deltaseries, fitorder);
fitvals = polyval(fitcoeffs, timeseries);
deltaresidue = deltaseries - fitvals;
deltaresidue = 1e3 * deltaresidue;

% Get a rough idea of the residue's spread.
residuemean = mean(deltaresidue);
residuedev = std(deltaresidue);
coremin = residuemean - coredeviations * residuedev;
coremax = residuemean + coredeviations * residuedev;


% Plot the fit residue against time.

clf('reset');

plot(timeseries, deltaresidue, 'HandleVisibility', 'off');

ylim([ coremin coremax ]);

title(sprintf('Time Shift Fit Residue (%s)', caselabel));
xlabel('Time (s)');
ylabel('Time Shift Residue (ms)');

saveas(thisfig, sprintf('%s-residue.png', fbase));


% Plot time shift jitter.
% FIXME - Assume there are outliers, and make two plots.

coreselect = (deltaresidue <= coremax) & (deltaresidue >= coremin);
residuecore = deltaresidue(coreselect);
residueoutliers = deltaresidue(~coreselect);


if ~isempty(residuecore)
  clf('reset');

  histogram(residuecore, 100);

  xlim([ coremin coremax ]);

  title(sprintf('Time Shift Jitter Detail (%s)', caselabel));
  xlabel('Time Shift Jitter (ms)');
  ylabel('Count');

  saveas(thisfig, sprintf('%s-jitter-core.png', fbase));
end
```

```
    if ~isempty(residueoutliers)
      clf('reset');

      histogram(residueoutliers, 100);

      title(sprintf('Time Shift Jitter Outliers (%s)', caselabel));
      xlabel('Time Shift Jitter (ms)');
      ylabel('Count');

      saveas(thisfig, sprintf('%s-jitter-outliers.png', fbase));
    end

  end


  % Reset this figure to free up plot-related memory.
  clf('reset');

end


% This plots a histogram of event arrival times.
% It's intended to map the locations of "miss" events.
%
% "timeseries" is a sequence of event time values.
% "thisfig" is a figure handle to use for plotting.
% "caselabel" is a string to annotate the plot title with.
% "fname" is the name to use for the rendered figure file.

function helper_plotEventTimes( timeseries, thisfig, caselabel, fname )

  % Select this figure.
  figure(thisfig);


  % Only make plots if we have at least two data points.
  if length(timeseries) >= 2

    % Plot a histogram of event arrival times.

    clf('reset');

    histogram(timeseries, 100);

    title(sprintf('Event Times (%s)', caselabel));
    xlabel('Time (s)');
    ylabel('Count');

    saveas(thisfig, fname);
```

```
  end


  % Reset this figure to free up plot-related memory.
  clf('reset');

end



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