This document is intended to provide a high level summary of the architecture of the homophone experiment for future reference. This will cover details about the design of the homophone experiment as well as a summary explanation of its basic operation. This will document will provide an overview of the following aspects of the homophone project:

- The method of stimuli selection for homophones
- The operation of the homophone presentation experiment
- The architecture of the data analysis pipeline
- An overview of the initial analyses that are run from this data

Section 1

Stimuli Selection:

The first phase of the homophone project consisted of developing a list of homophones as well as a set semantically related words that primed multiple senses of a given homophone. The homophones and their associated primes were chosen to optimize several phonetic and semantic properties. The final set of stimuli was created according to the following protocol.

First, a large list of homophones was assembled. This list was derived from 350 homophone pairs used in Twilley et al, as well as 563 pairs of homophones created by Dr. Neal Fox. This resulted in a combined list with 735 unique homophones. In the master google spreadsheet (Homophone Tables), this corresponds to the first sheet titled 'Homophone_Table_All'. For each of these words, we also collected several linguistic descriptors. This includes the following metrics:
Homophone-Prime LSA: Measured using the CU-Boulder Pairwise LSA web-app for the General Reading up to 1st year of College Reading Corpus.
iPHoD data: Number of Syllables, Number of Phonemes, Phoneme Density, Phonotactics
Part of Speech: Dr. Fox's personal model of english grammar.
Sense Count: WordNET part of speech tagged corpus (counts homographs only)
Mean Reaction Time: Semantic Priming Project
Free Association Probability: Taken from the USF free association norm study. Both P(Homophone|Prime) and P(Prime|Homophone) were used to create dominance and free association metrics by Dr. Fox.
SUBTLEX Word Count
Length of Stimulus: Duration of audio representation of stimuli as spoken by Dr. Hamilton

The list of possible stimuli was then down selected by Dr. Fox and Dr. Leonard to find homophones which were commonly and unambiguously associated with both of two different senses. This down selection process was done in three steps.
First, the USF free association norms were used to exclude homophones which did not have both senses given as a free association of some prime. Following this, the Semantic Priming Project database was used to select homophones where both senses were primed. This

resulted in a list of 52 homophones as well as 104 primes - one for each sense of the word. A final list was then manually selected from these 52 homophones by Dr. Matt Leonard and Dr. Neal Fox. This was done to remove idiomatic bigrams, and degenerate use cases of the prime-homophone pairs. This resulted in a final list of 30 homophones, with 60 primes.

The dominant sense of each of these words was then determined by looking at which sense was most commonly free associated to. This was measured by surveying lab members about which sense seemed to be the most probable for a given prime and then counting number of USF free associations of each sense. The final results of can be found in the main table under the sheet titled ‘FinalStatsSheet1’.
From this list of 30 homophones, a further reduction was made by Dr. Neal Fox and Professor Matt Leonard to reduce the list to 12 homophones and 24 primes which had a good distribution of the linguistic and semantic features that were intended for subsequent analysis.

Section 2

Homophone Priming Experiment:
The main experiment that was run with the homophone data was a semantic priming experiment where the subject was presented with a priming stimuli and then one of the 12 homophones. The priming stimuli would either prime the dominant sense, the subordinate sense, or be unrelated to the target homophone stimuli.
The subject was then prompted to repeat either the prime or the target. This repetition was grouped such that half of the possible homophones would always be repeated. The order of the stimuli was randomly shuffled such that the sequence would be random but subject to the constraint that a given target would not be repeated until there were at least 2 intervening stimuli. It will be noted that such sequences are in fact quite rare, and as the number of stimuli increases, the probability of such a sequence tends toward zero. Hence the ‘random’ shuffling algorithm may have problems.

The details involved in running the experiment can be found Lab’s experiment documentation files and it is hopefully intuitive in its operation.

In addition to this experiment a second homophone experiment was also created that used a lexical decision task to examine the differences in homophone perception, however this has not yet been deployed.

Section 3

Homophone Data Processing Pipeline:
A data processing pipeline has been developed to bring together the various sources of experimental data and synthesize a standard data format for subsequent analyses. This

function assumes that the subject's ECoG data has already been processed to reject bad channels and artifacts. And it further assumes that a subject's experimental data is saved in a standard file format which will be explained in greater detail below.

The homophone data processing pipeline architecture is intended to provide a comprehensive algorithm which will ensure that the ECoG, Audio and Lexical data from several blocks of experimental data can be easily accessed in a convenient and standardize form. It does this by transforming the acoustic lexical data of the experiment into an event structure using a modification of the TIMIT event detector, converting the raw ECoG of each block into 40-band envelope data. And then combining these two sources of information into ERP structures for each neural band (theta, alpha, beta, …).
To run the Data processing pipeline, the following steps must be taken.

1.) format the subject folder:
The main folder for a subject (ECXXX)
must have the followings three subfolders (the names and capitalization of these folders is important and must be done exactly as follows or the data cannot be processed into ERP structures):

behavior - contains the experimental behavior files (ex: 'Homophone_Exp_EC131_B21_1.mat',...), as a well as a list of homophones used in the experiment ('homophone_list_full.mat').

Event_Audio - contains the set of Audio recordings used in the homophone experiment ('shake.wav', 'shiver.wav')

ecog_data - contains the data for the audio and ecog recordings for each block. This folder should have a subfolder for each block of recording. These subfolders should be labeled ECXXX_BYY (e.g. EC123_B56). Since the blocks are loaded on the basis of the experimental behavior data, it is important that these should match the experimenter's input for the subject and block number of a recording. Each block's subfolder should contain the following subfolders of data: Analog - contains the audio data, Artifacts (contains the bad channels/time segments data), and ecog400 - contains the downsampled raw ECoG measurement.

Once the data has been properly formatted for a subject, run the homophone data processing wrapper with the following command:

```
>> DataProcessingPipelineWraper_2_2(subject_directory);
```

The first step of the pipeline is converting the acoustic and behavioural data into a data structure that contains information on the timing and sensory content of the stimuli. This is done by the function homophone_process_data(subj, root_dir, blocks);

This function combines the behavior files for each block into a single file and then attempt to automatically detect the timing of stimuli for the homophone task.
The homophone event detector must go through each event and attempt to find a match - hence it takes a while to run.

As an important detail for the operator, the event detector is not perfect and a manual review of the output is recommended.
If things go smoothly, ‘homophone_process_data’ will generate a graph of the confidence level for each event that was detected. A minimum threshold of 0.75 has been set, but generally events will low confidence should be noted and doubled checked to make sure that the stimulus was correctly identified.

Once the data has been run, you will notice that a new folder title 'events' has been created in the subject's data folder. This contains a series of structured arrays that contain the timing information for each.
These event structures should contain a multiple of 96 elements. About 1% of stimuli are dropped. Sometimes this is something as simple as the first/last pair of stimuli are missing from the recording, while other times a single event in the middle of a block is skipped. You will now need to manually inspect these to make sure that they each have the proper number of events. If there is not a multiple of 96 element in a blocks event structure, then a human operator must inspect this data and find out what is missing and potentially edit the event structure to prevent problems.

This inspection should be done by hand and requires some familiarity with the homophone data to perform. A few general remarks will be provided here to help locate anomalies.

1.) events come in pairs (prime - homophone), it is possible for a full pair to be dropped, especially at the start and end of a block. It is also possible for one member of a pair to be missing.
2.) Our current protocol is to simply delete the member of a pair if its partner is missing. The modified event structure must then be saved over the existing event structure, and the change noted in a log.
3.) To find missing events, sort the event structure by start time (it is not sorted at the start).
4.) Once sorted, events alternate between primes and homophones. This can be used to detect missing elements. The homophones should always come after a prime thus they must have an even numbered index. (for a list of homophones use the first column of words in the file ''homophone_list_full.mat' that was added into the behavior subfolder).
5.) Additionally, the timing of a pair has an even spacing in time, while the time between paris has a much longer and less stereotyped temporal spacing. This feature can be used to find missing events.

Once bad events have been identified, make a note of them, and delete them from the event and then resave the event file with the missing value deleted (for example, suppose that the 77th entry of Block 33 is bad, you will need to enter the following commands in matlab:

```
load([root_dir filesep 'events' filesep 'B33_evnt.mat']); % loads event structure
evnt(77) = []; % delete the partner of the pair with a missing member
save([root_dir filesep 'events' filesep 'B33_evnt.mat']); % overwrite the original structure
```

Once this has been completed, the rest of the wrapper should be able to run without further inputs. You will need to re-run the wrapper from the start, but it will automatically detect if the event structures have been generated, and will load these from memory rather than regenerating them from scratch.

After the events have been created, the processing pipeline will then generate 40 band hilbert transformed data for each block. This is a computationally intensity process and will take some time. This is done using the standard Chang lab function ‘TransformData.m’. No CAR is applied to the data, and the function generates 40 band hilbert transformed data at 400 hz for each block as well as a list of the center frequencies. These two data sources will be used later to find the subset of the 40 bands corresponding to a particular neural band (e.g. Alpha), and then average these band to form ERPs.

Once the 40 band hilbert data has been generated for a subject, the function will make data structures for each of the 5 main bands of neural data (theta, alpha, beta, low, high gamma). These data structures are all identical except for the content of the ecog erp data that the contain. The homophone ERP data structure contains the following fields:

ecog_primes: n_chanels x n_timepoints x n_trials prime onset locked erp matrix
ecog_targets: n_chanels x n_timepoints x n_trials erp target homophone onset locked erps
Time_axis: n_timepoints array that gives the time of each frame of the two erp matricies relative to event onset.
is_related_dominant: n_trials boolean array, that is true for trials where the prime is the dominant sense of the target homophone.
is_related_subordinant: n_trials boolean array, that is true for trials where the prime is the subordinate sense of the target homophone.
(Note that the logical union of these two arrays is a boolean array corresponding to trails where the prime and target are related).
target_names: n_trials cell array of the name of each target.
prime_names: n_trials cell array of the name of each target.
BadChans: a list of the channel indices for channels that were marked as bad during artifact rejection.
badTimeSegments: n_badTimesx2 matrix indicating regions that were marked bad during artifact rejection

is_good_trial: n_trials boolean array indicating whether a trial is free from bad times marked as bad during preprocessing. Currently the time span of a trial is from -500 ms pre-target onset (the z-scoring time) to the end of the target presentation.
prime_duration: n_trials array of the time duration of the prime (useful in determining stimulus timings, since the spacing between prime and target is always 500 ms).
target_duration: n_trials array of the time duration of the target.
prime_starttimes: n_trials array with the time that the primes started in a given block
target_starttimes: n_trials array with the time that the targets started in a given block
data_block: n_trials cell array containing the experimental block number of a given trail.
is_high_z_trial: n_trials boolean array containing information of whether a given trial contains high z score time points. Currently this function is not properly tuned and must be reworked before use in future analysis.
grd: [16x16 double] lists the configuration of the grid for a given experiment. This will be explained in detail below.

ECoG ERP Data:
The ecog erp data matrices are formed for a time window that is hard-coded. This is done on lines 89 and 90 of ‘make_homophone_task_erps_band.m’. By default, both the prime and the target locked erps are formed for a time window of -1.5 seconds pre-onset to 3 seconds post onset. This is most likely a wider time window than necessary, and it may be helpful to change this in the future. Additionally, the ecog for each event in the erps is z-scored using a <u>-0.5 to 0 second time window before the onset of the prime.</u> This is important since z-scoring the target at -0.5 to 0 seconds before the target onset would include the perception of the prime in the rest-state time interval.

The Grid:
Grid layout can vary across patients. The standard homophone grid ERP plotting function is designed to look for a subfield of the ERP structure called ‘grd’ that contains information on the ordering of electrodes. The simplest way to implement this is to modify the preprocessing pipeline for each subject to add their particular grid configuration. This is done on line 242 of the processing pipeline:

```
intial_corner = 'br'; tiling_dir = 'y';
ERPs.grd = grid_subplot_layout('br','y', 'plot_test_grid', true);
```

The function grid_subplot_layout generates a tiling matrix based on the user input for the corner that the first electrode is located in (bottom-right: ‘br’, top-right: ‘tr’, top-left: ‘tl’, bottom-left: ‘bl’) as well as the x-y direction of the tiling from the first tile (‘x’ or ‘y’). If, for instance, the first electrode is in the top right corner (‘tr’) and the second electrodes is one below it (‘y’), the ‘ERPs.grd = grid_subplot_layout(‘tr’, ‘y’); will generate the desired tiling grid.

As a check it can be helpful to plot the output of the tiling. This can be done using the flag, ‘plot_test_grid’, true, and this will show what the grid plot orientation will look like.

It is also possible to flip half of the grid. Using the flag ‘split_grid_flip’, the user can make the grid flip along either the ‘vertical’, or ‘horizonal’ axis. On EC133, for example, the grid is defined by:

```
ERPs.grd = grid_subplot_layout('bl','x', 'plot_test_grid', true, 'split_grid_flip', 'vertical');
```

Modifying the pipeline for non-grid data:
Currently the data processing pipeline is designed with the assumption that only the 256 channel ECoG grid is of interest to the experiment. Although the preprocessing function will perform the hilbert transform on as many electrodes as there are in the raw data folder, the ERP generating functions are currently set to only collect data from the 256 electrode grid. In order to add this feature to the data processing pipeline, the following considerations must be addressed:

1. Artifacts. Currently the list of bad channels and bad time points only covers the grid electrodes. The data must be re-processed to select artifacts from the non-grid electrodes. This poses the additional problem of whether to consider artifacts in grid and non grid electrodes together or separately. If there are two sets of artifacts, the sub-function ‘make_homophone_task_erps_band.m’ must be modified on line 165 to load a variable set of artifacts. This could be done by hand, but a more reasonable solution would be to change the function call to modify this.
2. Range of channels to load is currently set to only load the first 4 blocks of 64 channels (256 electrode grid). This is hard wired, on lines 169 and 172 of ‘make_homophone_taks_erps_band.m’
   To load non-grid electrodes, this must be altered. Since the number of non-grid electrodes varies, the function must either take an instruction of which 64 channel blocks to load, or it must read the list of files and load from the range of possible files.
3. Implementing changes to 1 and 2, will require modifications to the function call of ‘make_unified_task_erps’. This function is called on line 91 of ‘make_unified_erp_struct’, which is in turn called on line 229 of the data processing pipeline.
4. Plotting the downstream plotting functions, and the ‘.grd’ subfield of the ERP structures all assume a grid configuration. These will all need to be modified to plot non-grid formatted data.

Among the changes that will be necessary to add non-grid electrodes into the homophone data, the changes in the artifact selection and plotting are beyond the scope of this document, and will need to be addressed independently. The changes in how the data is loaded in the pre-processing script will be summarized below:

On line 229 on the pipeline script, the function ‘make_unified_erp_struct’ is called. This function is merely an agglomerating function that generates ERP data structures for individual blocks of data and then combines the. The individual block ERP data is formed by the function ‘make_homophone_task_erps_band.m’, which is called on line 91 of ‘make_unified_erp_struct’. This is where modifications must be made in order to include non-grid electrodes.

In ‘make_unified_erp_struct’, there is a helper function named ‘load_ecog_data_bands’ on line 153. This must be modified in the following two ways:

1.) on lines 161,162, the files names for artifacts must be altered for non-grid data.
2.) On lines 169,171 the number of blocks (default = 4) must be changed so that non-grid electrodes are loaded.
3.) The list of bad channels must be modified to account for the fact that only a subset of electrodes may be loaded.

A first pass at this has been made in the modified function ‘make_homophone_task_erps_band_varblocks.m’
This can be called as follows:

```
>> ERPs_grid = make_homophone_task_erps_band_varblocks(evnt, band_inds, ‘grid’, ‘bad_times_grid.mat’, ‘bad_chans_grid.txt’);
>> ERPs_nongrid = make_homophone_task_erps_band_varblocks(evnt, band_inds, ‘nongrid’, ‘bad_times_nongrid.mat’,
‘bad_chans_nongrid.txt’);
```

End of Pipeline:
Once the ERP structure for a neural band has been generated, there are two final sections sections that are optional.
The first is a section to plot a test case. This is often useful in debugging the data to ensure that the data and functions are both behaving as they ought to be. Additionally it illustrates some basic analyses that are possible with the data structure.
The second is a section to save the ERP data structure for a given band. This step is recommended.

Section 4

Data Analysis:

Once the experimental data has been gathered into a standardized data structure, it is possible to carry out several comparisons between the various classes of homophone trials (related prime-target vs unrelated prime-target, high-FA score vs low-FA score, etc…). Currently our analysis has focused on generating erp plots for each of the 5 bands for a common set of comparisons as well as generating spectral erps for the same set of comparisons. Beyond this we have also explored the use of an LDA run with a sliding time window to explore the separability of the classes, however this path will need to be expanded in the future.

ERPs:

The standard homophone data structure naturally lends itself to plotting comparisons between the erps of various classes of trials for a given neural band. This ability is briefly demonstrated in the plot_erp section at the end of the pipeline.

The core plotting function for homophone ERP comparison is the function '**PlotECogGrid_std_v2.m**', this function is designed to plot grid erps for several different sets of event classes. Full documentation for this function can be found in the matlab .m file for this function. To give a simple overview, there are two required inputs - a Homophone ERP data structure for timing and grid tiling information, and a time window array (in seconds about the event onset (this window cannot exceed the temporal extent of the ecog data).
It can then plot erps (mean +/- standard error of the mean) for any number of ecog erp matricies (a 256 x time x trials matrix). In addition, it is possible to add a number of flag inputs. These add features like shading for significant points, or a title on the entire plot.
Example Code:

```
% data:
ecog = ERPs.ecog_targets;

is_class1 = ERPs.is_good_trial &  ERPs.is_related_dominant & strcmpi(ERPs.target_names,'mouse');
is_class2 = ERPs.is_good_trial &  ERPs.is_related_subordinant & strcmpi(ERPs.target_names,'mouse');
is_class3 = ERPs.is_good_trial &  ~ERPs.is_related_subordinant...
   & ~ERPs.is_related_dominant & strcmpi(ERPs.target_names,'mouse');

% Plot all target locked erps for -1 to 2 seconds
PlotECogGrid_std_v2(ERPs, [-1 2], ecog)

% Plot Comparison of class1 and class2 and class3 for [-1 to 1] seconds
PlotECogGrid_std_v2(ERPs, [-1 1], ecog(:,:,is_class1), ecog(:,:,is_class2), ecog(:,:,is_class3))

% Add significance shading:
is_sig_ch_time = ttest2(ecog(:,:,is_class1), ecog(:,:,is_class2), 'Dim',3, 'Alpha', 0.05);
PlotECogGrid_std_v2(ERPs, [-1 1], ecog(:,:,is_class1), ecog(:,:,is_class2), ecog(:,:,is_class3), 'SigPts', is_sig_ch_time)

% Add a tile:
PlotECogGrid_std_v2(ERPs, [-1 1], ecog(:,:,is_class1), ecog(:,:,is_class2), ecog(:,:,is_class3), 'SigPts', is_sig_ch_time, 'Title', 'Comparison of Class1, Class2 and Class 3')
```

One limitation of PlotECogGrid_std_v2 is that it assumes that the data is a 256 channel grid. An initial attempt at a non-grid equivalent can be found in PlotECogGrid_std_v3, however this function has never been deployed on non-grid data.

Since several of the basic exploratory analyses are stereotyped, it is possible to run a script that automatically generates erp comparisons for the basic analyses. Such a function can be found in **‘generate_erp_plots_script.m’** This requires a homophone ERP structure as well as a specified save directory and this will automatically generate several folders with .png files of standard ERP plots.
These .png files can be agglomerated into a single pdf using the bash shell script command founds in ‘Spectrogrammer/pngs_2_mono_pdf.txt’.

Spectrograms:
A spectrogram can be a very useful way to explore the trends in ecog data since it shows the the time dynamics of several frequency bands of all channels on a single plot. In order to carry out such an analysis the following functions have been developed to compare the behavior of different classes of trials. In general these follow a similar pattern as the comparisons between high gamma erps, however there are a few differences that limit these visualizations. Most notably it is only possible to show a single time course of a frequency x time x color spectrogram plot. This can be extended to show comparisons between two different classes of data by plotting their difference, however it is not possible to make three way comparisons on a spectrogram.

As a similar constraint, it is not possible to plot an error bar on a spectrogram. Thus we are currently plotting the t-statistic between event classes. This gives the comparison some statistical interpretation, however it may be useful in the future to also include contour lines to indicate significance.

Because of the large amount of data stored in a spectrogram erp matrix (n_chans x n_timepts x n_freqbands x n_trials = 256x300x40x600 = 1.4*10^10 Bytes ~ 10GB), the spectrogram functions adopt an architecture where data is loaded, processed into and then saved one channel at a time.

In the future, it may be desirable to consolidate the functions for generating spectrogram plots into a single macro, however at the present moment, the core operations are segmented into a set of sub functions that are described below:

For a given subject the first step in forming spectral erp plots is to create event locked spectral erp data for each channel. This is done by the function ‘**create_channel_spectrogram_erps.m**’. This takes one of the several homophone ERP structures generated for a subject in the homophone data processing pipeline, the file directory that contains the data on each block of ecog data as well as the subject name. It then sequentially loops through each channel and loads the relevant 40 band data for a given trial. This time_pts x frequency_bands x trials matrix is then saved into a new folder in the subject’s data folder titled ‘spectrogram_data’. It should be noted that this function assumes a 256 channel grid of data (line 58), and also only examines target locked erps (line 42).

Once the target locked spectral erps for a given channel have been created, it is possible to plot various comparisons. The function '**plot_spectral_tstat.m**' is useful in making comparisons between two classes of trials. It takes an Homophone ERP structure (for timing, trial and grid information), the location of the spectrogram erp data folder (generated by 'create_channel_spectrogram_erps.m') and two num_trials boolean arrays indicating the two classes of trials that should be compared. The function then sequentially loads in the data for each channel, calculates the t-statistic for each time point and frequency band and plots the resulting measurement on a grid.

The function **'generate_spectrogram_plots_script.m'** serves as a first pass at automatically generating a full set of comparisons between the key classes of homophone comparisons. This includes plots for each of the 12 homophones for High vs Low Free Association Score, Dominant vs Subordinate and Related vs Unrelated. These plots are automatically saved as .png files (an attempt at saving as .pdf was made, but there were some formatting issues). These .png files can be concatenated into a single pdf file in the mac terminal by moving to the relevant directory of .png files in the terminal and entering the script found in 'pngs_2_mono_pdf.txt'.

LDA:
One of the first analyses that has been used to examine the separability of various classes of homophones has been linear discriminant analysis. Most of the work using LDA was done during the initial explorations on EC123, and it is likely that a lot of this will need to be updated. Thus this section will emphasize some of the more general considerations of running an LDA classifier rather than the specific implementation of this.

The primary concern with running any machine learning algorithm on the homophone data set is the size of the data set. Although there are generally around 500 trials, if we want to compare something like dominant vs. subordinate primed senses of a specific homophone, there will be approximately 10 measurements of each class. This means that the dimensionality of the data must be reduced, and it also means that some cross validation schemes will be less effective, and it means that a gamma parameter must be added to the classifier to mitigate the effects of overfitting.

Currently the set of electrodes used in the classifier is downselected in the function 'find_sig_chans_homophones.m'. This function takes the homophone ERP data structure as well as two 1d boolean arrays representing classes that will be compared using LDA and then finds channels that have a threshold number of time points in the interval [0, 1] seconds after stimulus onset that show significance with a two way t-test.
The main function that has been used for LDA classification is 'classify_homophones_sig_chans_ii.m'. This function is somewhat limited in its generality and is only set up to run classifications on Unrelated/Related primes and Dominant/Subordinate

primes. We found that a gamma level of 0.6 provided the most accurate results, however values between 0.3 and 0.7 all had similar effects. We used a leave one out cross validation scheme since this provides reproducible results and mitigates the high variability in cross validation accuracy that results from the small sample size of the test/train data sets.