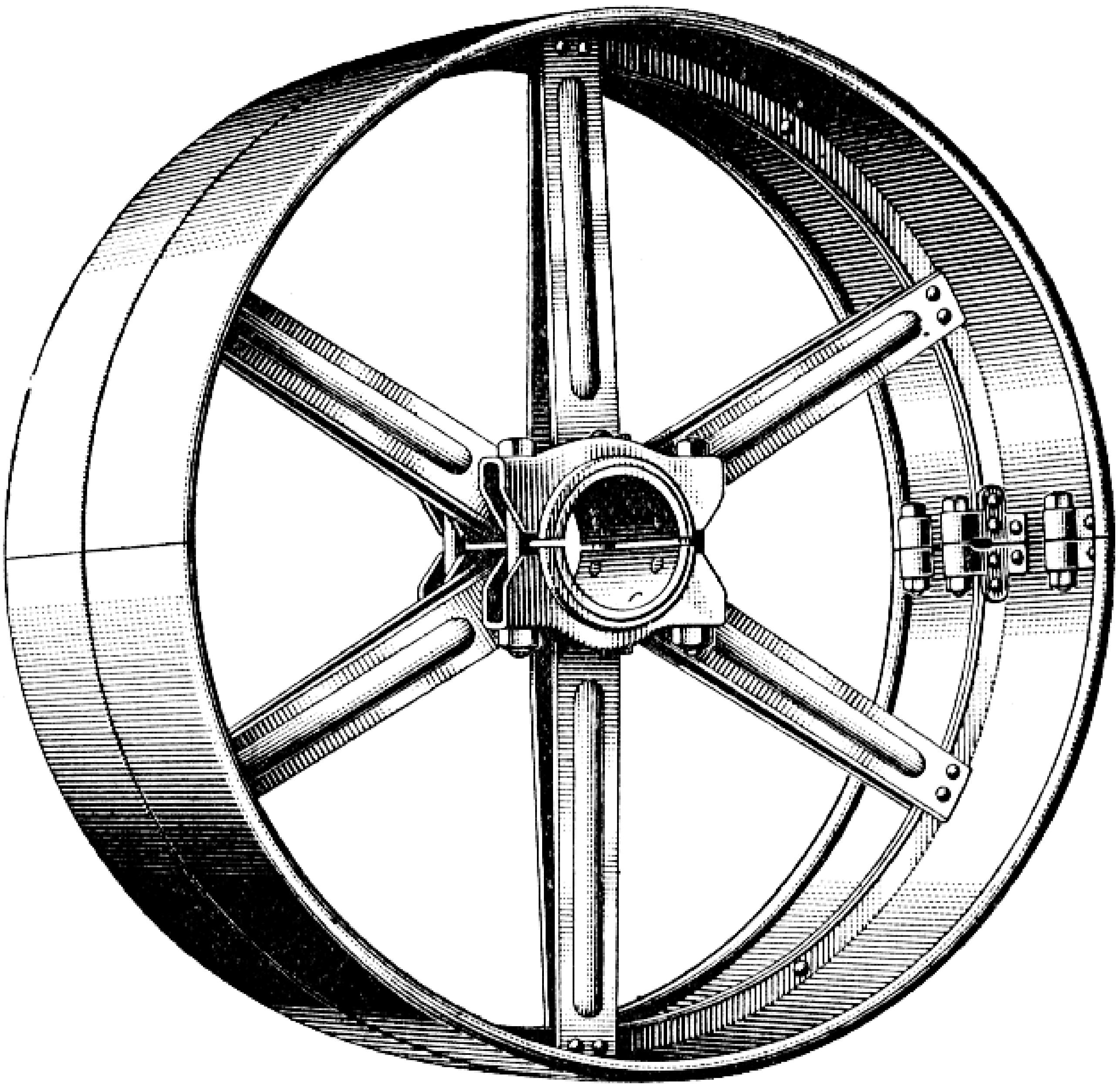

FIG. 16

OpenEars 2.03 Manual

Written by Halle Winkler, published by Politepix

Friday, January 9, 2015

Introduction and Installation

Introduction

OpenEars is an shared-source iOS framework for iPhone voice recognition and speech synthesis (TTS). It lets you easily implement round-trip English and Spanish language speech recognition and English text-to-speech on the iPhone, iPod and iPad and uses the open source CMU Pocketsphinx, CMU Flite, and CMUCLMTK libraries, and it is free to use in an iPhone, iPad or iPod app (Spanish text-to-speech is possible on the OpenEars Platform but requires using NeatSpeech since there isn't a Spanish voice for Flite). It is the most popular offline framework for speech recognition and speech synthesis on iOS and has been featured in development books such as O'Reilly's *Basic Sensors in iOS* by Alasdair Allan and *Cocos2d for iPhone 1 Game Development Cookbook* by Nathan Burba.

The OpenEars Platform is also a complete development platform for creating your speech recognition and text-to-speech apps including both the free OpenEars SDK documented on this page and a diverse set of plugins that can be added to OpenEars in order to extend and refine its default features: you can read more about the OpenEars platform here. This page is all about the free and shared-source OpenEars SDK, to please read on to learn more about it.

Highly-accurate large-vocabulary recognition (that is, trying to recognize any word the user speaks out of many thousands of known words) is not yet a reality for local in-app processing on a small handheld device given the hardware limitations of the platform; even Siri does its large-vocabulary recognition on the server side. However, Pocketsphinx (the open source voice recognition engine that OpenEars uses) is capable of local recognition of vocabularies with hundreds of words depending on the environment and other factors, and performs very well with command-and-control language models in English and Spanish. The best part is that it uses no network connectivity because all processing occurs locally on the device.

The current version of OpenEars is 2.03. Download OpenEars or read its changelog. If you are upgrading to OpenEars 2.x from a 1.x version, it is necessary to follow the upgrade guide once in order to successfully upgrade.

Features of OpenEars

OpenEars can:

- Perform speech recognition in English and in Spanish
- Perform text-to-speech in English and with the NeatSpeech plugin, can also perform text-to-speech in Spanish
- Listen continuously for speech on a background thread, while suspending or resuming speech processing on demand, all while using less than 2% CPU on average on current devices (decoding speech, text-to-speech, updating the UI and other intermittent functions use more CPU),
- Use any of 9 voices for speech, including male and female voices with a range of speed/quality level, and switch between them on the fly,
- Change the pitch, speed and variance of any text-to-speech voice,
- Know whether headphones are plugged in and continue voice recognition during text-to-speech only when they are plugged in,
- Support bluetooth audio devices (experimental),
- Dispatch information to any part of your app about the results of speech recognition and speech, or changes in the state of the audio session (such as an incoming phone call or headphones being plugged in),
- Deliver level metering for both speech input and speech output so you can design visual feedback for both states.
- Support JSGF grammars with an easy-to-use human-readable grammar specification language, only from Politepix,
- Dynamically generate probability-based language models and rule-based grammars using simple object-oriented language
- Switch between ARPA language models or JSGF grammars on the fly,
- Get n-best lists with scoring,
- Test existing recordings,
- Be easily interacted with via standard and simple Objective-C methods,
- Control all audio functions with text-to-speech and speech recognition in memory instead of writing audio files to disk and then reading them,

- Protect user privacy by performing all recognition offline and not storing speech audio,
- Drive speech recognition with a low-latency Audio Unit driver for highest responsiveness,
- Be installed in a Cocoa-standard fashion using an easy-peasy already-compiled framework.
- In addition to its various new features and faster recognition/text-to-speech responsiveness, OpenEars now has improved recognition accuracy.
- OpenEars is free to use in an iPhone or iPad app.

Warning

Before using OpenEars, please note it has to use a different audio driver on the Simulator that is less accurate, so it is always necessary to evaluate accuracy on a real device. Please don't submit support requests for accuracy issues with the Simulator.

Installation

To use OpenEars:

- Download the distribution and unpack it.

- Create your own app, and then add the iOS frameworks AudioToolbox and AVFoundation to it.

- Inside your downloaded distribution there is a folder called "Framework". Drag the "Framework" folder into your app project in Xcode.

OK, now that you've finished laying the groundwork, you have to...wait, that's everything. You're ready to start using OpenEars. Give the sample app a spin to try out the features (the sample app uses ARC so you'll need a recent Xcode version) and then visit the Politepix interactive tutorial generator for a customized tutorial showing you exactly what code to add to your app for all of the different functionality of OpenEars.

If the steps on this page didn't work for you, you can getfree support at the forums, read the FAQ, brush up on the documentation, or open a private email support incident at the Politepix shop. If you'd like to read the documentation, simply read onward.

Basic concepts

There are a few basic concepts to understand about voice recognition and OpenEars that will make it easiest to create an app.

- Local or offline speech recognition versus server-based or online speech recognition: most speech recognition on the iPhone, iPod and iPad is done by streaming the speech audio to servers. OpenEars works by doing the recognition inside the device, entirely offline without using the network. This saves bandwidth and results in faster response, but since a server is much more powerful than a phone it means that we have to work with much smaller vocabularies to get accurate recognition.
- Language Models. The language model is the vocabulary that you want OpenEars to understand, in a format that its speech recognition engine can understand. The smaller and better-adapted to your users' real usage cases the language model is, the better the accuracy. An good language model for PocketsphinxController has fewer than 500 words. You define the words that your app uses - it will not know about vocabulary other than the vocabulary that you define.
- The parts of OpenEars. OpenEars has a simple, flexible and very powerful architecture.

OEPocketsphinxController recognizes speech using a language model that was dynamically created by OELanguageModelGenerator. OEFliteController creates synthesized speech (TTS). And OEEventsObserver dispatches messages about every feature of OpenEars (what speech was understood by the engine, whether synthesized speech is in progress, if there was an audio interruption) to any part of your app.

OEAcousticModel Class Reference

Detailed Description

Convenience class for accessing the acoustic model bundles. All this does is allow you to reference your chosen model by including this header in your class and then letting you call [OEAcousticModel pathToModel:"AcousticModelEnglish"] or [OEAcousticModel pathToModel:@"AcousticModelSpanish"] in any of the methods which ask for a path to an acoustic model.

Method Documentation

```
+ (NSString *) pathToModel: (NSString *) acousticModelBundleName
```

Reference the path to any acoustic model bundle you've dragged into your project (such as AcousticModelSpanish.bundle or AcousticModelEnglish.bundle) by calling this class method like [OEAcousticModel pathToModel:"AcousticModelEnglish"] after importing this class.

OEContinuousModel Class Reference

OEEventsObserver Class Reference

Detailed Description

OEEventsObserver provides a large set of delegate methods that allow you to receive information about the events in OpenEars from anywhere in your app. You can create as many OEEventsObservers as you need and receive information using them simultaneously. All of the documentation for the use of OEEventsObserver is found in the section OEEventsObserverDelegate.

Property Documentation

```
- (id<OEEventsObserverDelegate>) delegate
```

To use the OEEventsObserverDelegate methods, assign this delegate to the class hosting OEEventsObserver and then use the delegate methods documented under OEEventsObserverDelegate. There is a complete example of how to do this explained under the OEEventsObserverDelegate documentation.

OEFliteController Class Reference

Detailed Description

The class that controls speech synthesis (TTS) in OpenEars.

Usage examples

> *Preparing to use the class:*

To use OEFliteController, you need to have at least one Flite voice added to your project. When you added the "framework" folder of OpenEars to your app, you already imported a voice called Slt, so these instructions will use the Slt voice. You can get eight more free voices in OpenEarsExtras, available at https://bitbucket.org/Politepix/openearsextras

> *What to add to your header:*

Add the following lines to your header (the .h file). Under the imports at the very top:

```
#import <Slt/Slt.h>
#import <OpenEars/OEFliteController.h>
```

Add these class properties to the other properties of your view controller or object:

```
@property (strong, nonatomic) OEFliteController *fliteController;
@property (strong, nonatomic) Slt *slt;
```

> *What to add to your implementation:*

Add the following to your implementation (the .m file): Before you want to use TTS speech in your app, instantiate an OEFliteController and a voice as follows (perhaps in your view controller's viewDidLoad method):

```
  self.fliteController = [[OEFliteController alloc] init];
  self.slt = [[Slt alloc] init];
```

> *How to use the class methods:*

After having initialized your OEFliteController, add the following message in a method where you want to call speech:

```
[self.fliteController say:@"A short statement" withVoice:self.slt];
```

Warning

There can only be one OEFliteController instance in your app at any given moment.

Method Documentation

```
- (void) say: (NSString *)      statement
   withVoice: (OEFliteVoice *)  voiceToUse
```

This takes an NSString which is the word or phrase you want to say, and the OEFliteVoice to use to say the phrase. Usage Example:

```
 [self.fliteController say:@"Say it, don't spray it."
withVoice:self.slt];
```

There are a total of nine FliteVoices available for use with OpenEars. The Slt voice is the most popular one and it ships with OpenEars. The other eight voices can be downloaded as part of the OpenEarsExtras package available at the URL http://bitbucket.org/Politepix/openearsextras. To use them, just drag the desired downloaded voice's framework into your app, import its header at the top of your calling class (e.g. import <Slt/Slt.h> or import <Rms/Rms.h>) and instantiate it as you would any other object, then passing the instantiated voice to this method.

Property Documentation

```
 - (Float32) fliteOutputLevel
```

A read-only attribute that tells you the volume level of synthesized speech in progress. This is a UI hook. You can't read it on the main thread.

```
- (float) duration_stretch
```

duration_stretch changes the speed of the voice. It is on a scale of 0.0-2.0 where 1.0 is the default.

```
- (float) target_mean
```

target_mean changes the pitch of the voice. It is on a scale of 0.0-2.0 where 1.0 is the default.

```
- (float) target_stddev
```

target_stddev changes convolution of the voice. It is on a scale of 0.0-2.0 where 1.0 is the default.

```
- (BOOL) userCanInterruptSpeech
```

Set userCanInterruptSpeech to TRUE in order to let new incoming human speech cut off synthesized speech in progress.

OELanguageModelGenerator Class Reference

Detailed Description

The class that generates the vocabulary the OEPocketsphinxController is able to understand.

Usage examples

> *What to add to your implementation:*

In offline speech recognition, you define the vocabulary that you want your app to be able to recognize. This is called a language model or grammar (you can read more about these options in the OELanguageModelGenerator documentation). A good vocabulary size for an offline speech recognition app on the iPhone, iPod or iPad is between 10 and 500 words. Add the following to your implementation (the .m file): Under the @implementation keyword at the top:

```
#import <OpenEars/OELanguageModelGenerator.h>
```

> *How to use the class methods:*

In the method where you want to create your language model (for instance your viewDidLoad method), add the following method call (replacing the placeholders like "WORD" and "A PHRASE" with actual words and phrases you want to be able to recognize):

```
OELanguageModelGenerator *lmGenerator = [[OELanguageModelGenerator
alloc] init];

NSArray *words = [NSArray arrayWithObjects:@"WORD", @"STATEMENT",
@"OTHER WORD", @"A PHRASE", nil];
NSString *name = @"NameIWantForMyLanguageModelFiles";
NSError *err = [lmGenerator generateLanguageModelFromArray:words
withFilesNamed:name forAcousticModelAtPath:[OEAcousticModel
pathToModel:@"AcousticModelEnglish"]]; // Change
"AcousticModelEnglish" to "AcousticModelSpanish" to create a
Spanish language model instead of an English one.

NSString *lmPath = nil;
NSString *dicPath = nil;

if(err == nil) {

 lmPath = [lmGenerator
pathToSuccessfullyGeneratedLanguageModelWithRequestedName:@"NameIWa
ntForMyLanguageModelFiles"];
 dicPath = [lmGenerator
pathToSuccessfullyGeneratedDictionaryWithRequestedName:@"NameIWantF
orMyLanguageModelFiles"];

} else {
 NSLog(@"Error: %@",[err localizedDescription]);
}
```

It is a requirement to enter your words and phrases in all capital letters, since the model is generated against a dictionary in which the entries are capitalized (meaning that if the words in the array aren't capitalized, they will not match the dictionary and you will not have the widest variety of pronunciations understood for the word you are using).

Method Documentation

```
- (NSError *) generateLanguageModelFromArray: (NSArray *)   languageModelArray
                              withFilesNamed: (NSString *)  fileName
                       forAcousticModelAtPath: (NSString *)  acousticModelPath
```

Generate a probabilistic language model from an array of NSStrings which are the words and phrases you want OEPocketsphinxController or OEPocketsphinxController+RapidEars to understand, using your chosen acoustic model.

Putting a phrase in as a string makes it somewhat more probable that the phrase will be recognized as a phrase when spoken. If you only ever want certain phrases or word sequences to be recognized at the exclusion of other combinations, use -(NSError *) generateGrammarFromDictionary:(NSDictionary *)grammarDictionary withFilesNamed:(NSString *)fileName forAcousticModelAtPath:(NSString *)acousticModelPath below instead to create a rules-based grammar instead of a probabilistic language model.

fileName is the way you want the output files to be named, for instance if you enter "MyDynamicLanguageModel" you will receive files output to your Caches directory titled MyDynamicLanguageModel.dic, MyDynamicLanguageModel.arpa, and MyDynamicLanguageModel.DMP.

The words and phrases in languageModelArray must be written with capital letters exclusively, for instance "word" must appear in the array as "WORD".

If this method is successful it will return nil. If it returns nil, you can use the methods pathToSuccessfullyGeneratedDictionaryWithRequestedName: and pathToSuccessfullyGeneratedLanguageModelWithRequestedName: or pathToSuccessfullyGeneratedGrammarWithRequestedName: to get your paths to your newly-generated language models and grammars and dictionaries for use with OEPocketsphinxController. If it doesn't return nil, it will return an error which you can check for debugging purposes.

```
- (NSString *)
pathToSuccessfullyGeneratedDictionaryWithRequestedName:    (NSString *)    name
```

If generateLanguageModelFromArray:withFilesNamed:forAcousticModelAtPath: does not return an error, you can use this method to receive the full path to your generated phonetic dictionary for use with OEPocketsphinxController.

```
- (NSString *)
pathToSuccessfullyGeneratedLanguageModelWithRequestedName:    (NSString *)    name
```

If generateLanguageModelFromArray:withFilesNamed:forAcousticModelAtPath: does not return an error, you can use this method to receive the full path to your generated language model for use with OEPocketsphinxController.

```
- (NSString *) pathToSuccessfullyGeneratedGrammarWithRequestedName: (NSString *) name
```

If generateLanguageModelFromArray:withFilesNamed:forAcousticModelAtPath: does not return an error, you can use this method to receive the full path to your generated grammar for use with OEPocketsphinxController.

```
- (NSError *) generateGrammarFromDictionary: (NSDictionary *) grammarDictionary
                             withFilesNamed: (NSString *)     fileName
                     forAcousticModelAtPath: (NSString *)     acousticModelPath
```

Dynamically generate a JSGF grammar using OpenEars' natural language system for defining a speech recognition ruleset. This will recognize exact phrases instead of probabilistically recognizing word combinations in any sequence.

The NSDictionary you submit to the argument generateGrammarFromDictionary: is a key-value pair consisting of an NSArray of words stored in NSStrings indicating the vocabulary to be listened for, and an NSString key which is one of the following #defines from GrammarDefinitions.h, indicating the rule for the vocabulary in the NSArray:

```
ThisWillBeSaidOnce
ThisCanBeSaidOnce
ThisWillBeSaidWithOptionalRepetitions
ThisCanBeSaidWithOptionalRepetitions
OneOfTheseWillBeSaidOnce
OneOfTheseCanBeSaidOnce
OneOfTheseWillBeSaidWithOptionalRepetitions
OneOfTheseCanBeSaidWithOptionalRepetitions
```

Breaking them down one at a time for their specific meaning in defining a rule:

```
ThisWillBeSaidOnce // This indicates that the word or words in the
array must be said (in sequence, in the case of multiple words),
one time.
ThisCanBeSaidOnce // This indicates that the word or words in the
array can be said (in sequence, in the case of multiple words), one
time, but can also be omitted as a whole from the utterance.
ThisWillBeSaidWithOptionalRepetitions // This indicates that the
word or words in the array must be said (in sequence, in the case
of multiple words), one time or more.
ThisCanBeSaidWithOptionalRepetitions // This indicates that the
word or words in the array can be said (in sequence, in the case of
multiple words), one time or more, but can also be omitted as a
whole from the utterance.
OneOfTheseWillBeSaidOnce // This indicates that exactly one
selection from the words in the array must be said one time.
OneOfTheseCanBeSaidOnce // This indicates that exactly one
selection from the words in the array can be said one time, but
that all of the words can also be omitted from the utterance.
OneOfTheseWillBeSaidWithOptionalRepetitions // This indicates that
exactly one selection from the words in the array must be said, one
time or more.
OneOfTheseCanBeSaidWithOptionalRepetitions // This indicates that
exactly one selection from the words in the array can be said, one
time or more, but that all of the words can also be omitted from
the utterance.
```

Since an NSString in these NSArrays can also be a phrase, references to words above should also be understood to apply to complete phrases when they are contained in a single NSString.

A key-value pair can also have NSDictionaries in the NSArray instead of NSStrings, or a mix of NSStrings and NSDictionaries, meaning that you can nest rules in other rules.

Here is an example of a complex rule which can be submitted to the generateGrammarFromDictionary: argument followed by an explanation of what it means:

```
 @{
     ThisWillBeSaidOnce : @[
         @{ OneOfTheseCanBeSaidOnce : @[@"HELLO COMPUTER", 
@"GREETINGS ROBOT"]},
         @{ OneOfTheseWillBeSaidOnce : @[@"DO THE FOLLOWING", 
@"INSTRUCTION"]},
         @{ OneOfTheseWillBeSaidOnce : @[@"GO", @"MOVE"]},
         @{ThisWillBeSaidWithOptionalRepetitions : @[
             @{ OneOfTheseWillBeSaidOnce : @[@"10", @"20",@"30"]}, 
             @{ OneOfTheseWillBeSaidOnce : @[@"LEFT", @"RIGHT", 
@"FORWARD"]}
         ]},
         @{ OneOfTheseWillBeSaidOnce : @[@"EXECUTE", @"DO IT"]},
         @{ ThisCanBeSaidOnce : @[@"THANK YOU"]}
     ]
 };
```

Breaking it down step by step to explain exactly what the contents mean:

```
 @{
     ThisWillBeSaidOnce : @[ // This means that a valid utterance
for this ruleset will obey all of the following rules in sequence
in a single complete utterance:
         @{ OneOfTheseCanBeSaidOnce : @[@"HELLO COMPUTER",
@"GREETINGS ROBOT"]}, // At the beginning of the utterance there is
an optional statement. The optional statement can be either "HELLO
COMPUTER" or "GREETINGS ROBOT" or it can be omitted.
         @{ OneOfTheseWillBeSaidOnce : @[@"DO THE FOLLOWING",
@"INSTRUCTION"]}, // Next, an utterance will have exactly one of
the following required statements: "DO THE FOLLOWING" or
"INSTRUCTION".
         @{ OneOfTheseWillBeSaidOnce : @[@"GO", @"MOVE"]}, //
Next, an utterance will have exactly one of the following required
statements: "GO" or "MOVE"
         @{ThisWillBeSaidWithOptionalRepetitions : @[ // Next, an
utterance will have a minimum of one statement of the following
nested instructions, but can also accept multiple valid versions of
the nested instructions:
             @{ OneOfTheseWillBeSaidOnce : @[@"10", @"20",@"30"]},
// Exactly one utterance of either the number "10", "20" or "30",
             @{ OneOfTheseWillBeSaidOnce : @[@"LEFT", @"RIGHT",
@"FORWARD"]} // Followed by exactly one utterance of either the
word "LEFT", "RIGHT", or "FORWARD".
         ]},
         @{ OneOfTheseWillBeSaidOnce : @[@"EXECUTE", @"DO IT"]}, //
Next, an utterance must contain either the word "EXECUTE" or the
phrase "DO IT",
         @{ ThisCanBeSaidOnce : @[@"THANK YOU"]} and there can be
an optional single statement of the phrase "THANK YOU" at the end.
     ]
 };
```

So as examples, here are some sentences that this ruleset will report as hypotheses from user utterances:

```
"HELLO COMPUTER DO THE FOLLOWING GO 20 LEFT 30 RIGHT 10 FORWARD
EXECUTE THANK YOU"
"GREETINGS ROBOT DO THE FOLLOWING MOVE 10 FORWARD DO IT"
"INSTRUCTION 20 LEFT 20 LEFT 20 LEFT 20 LEFT EXECUTE"
```

But it will not report hypotheses for sentences such as the following which are not allowed by the rules:

```
"HELLO COMPUTER HELLO COMPUTER"
"MOVE 10"
"GO RIGHT"
```

Since you as the developer are the designer of the ruleset, you can extract the behavioral triggers from your app from hypotheses which observe your rules.

The words and phrases in languageModelArray must be written with capital letters exclusively, for instance "word" must appear in the array as "WORD".

The last two arguments of the method work identically to the equivalent language model method. The withFilesNamed: argument takes an NSString which is the naming you would like for the files output by this method. The argument acousticModelPath takes the path to the relevant acoustic model.

This method returns an NSError, which will either return an error code or it will return noErr with an attached userInfo NSDictionary containing the paths to your newly-generated grammar (a .gram file) and corresponding phonetic dictionary (a .dic file). Remember that when you are passing .gram files to the Pocketsphinx method:

```
- (void) startListeningWithLanguageModelAtPath:(NSString
*)languageModelPath dictionaryAtPath:(NSString *)dictionaryPath
acousticModelAtPath:(NSString *)acousticModelPath
languageModelIsJSGF:(BOOL)languageModelIsJSGF;
```

you will now set the argument languageModelIsJSGF: to TRUE.

```
- (NSError *) generateLanguageModelFromTextFile: (NSString *) pathToTextFile
                                 withFilesNamed: (NSString *) fileName
                         forAcousticModelAtPath: (NSString *) acousticModelPath
```

Generate a language model from a text file containing words and phrases you want OEPocketsphinxController to understand, using your chosen acoustic model. The file should be formatted with every word or contiguous phrase on its own line with a line break afterwards. Putting a phrase in on its own line makes it somewhat more probable that the phrase will be recognized as a phrase when spoken.

Give the correct full path to the text file as a string. fileName is the way you want the output files to be named, for instance if you enter "MyDynamicLanguageModel" you will receive files output to your Caches directory titled MyDynamicLanguageModel.dic, MyDynamicLanguageModel.arpa, and MyDynamicLanguageModel.DMP.

If this method is successful it will return nil. If it returns nil, you can use the methods pathToSuccessfullyGeneratedDictionaryWithRequestedName: and pathToSuccessfullyGeneratedLanguageModelWithRequestedName: to get your paths to your newly-generated language models and grammars and dictionaries for use with OEPocketsphinxController. If it doesn't return nil, it will return an error which you can check for debugging purposes.

Property Documentation

```
- (BOOL) verboseLanguageModelGenerator
```

Set this to TRUE to get verbose output

```
- (BOOL) useFallbackMethod
```

Advanced: if you are using your own acoustic model or an custom dictionary contained within an acoustic model and these don't use the same phonemes as the English or Spanish acoustic models, you will need to set useFallbackMethod to FALSE so that no attempt is made to use the English or Spanish fallback method for finding pronunciations of words which don't appear in the custom acoustic model's phonetic dictionary.

OELogging Class Reference

Detailed Description

A singleton which turns logging on or off for the entire framework. The type of logging is related to overall framework functionality such as the audio session and timing operations. Please turn OELogging on for any issue you encounter. It will probably show the problem, but if not you can show the log on the forum and get help.

Warning

The individual classes such as OEPocketsphinxController and OELanguageModelGenerator have their own verbose flags which are separate from OELogging.

Method Documentation

```
+ (id) startOpenEarsLogging
```

This just turns on logging. If you don't want logging in your session, don't send the startOELogging message.

> *Example Usage:*

Before implementation:

```
#import <OpenEars/OELogging.h>;
```

In implementation:

```
[OELogging startOpenEarsLogging];
```

OEPocketsphinxController Class Reference

Detailed Description

The class that controls local speech recognition in OpenEars.

Usage examples

> *What to add to your header:*

To use OEPocketsphinxController, the class which performs speech recognition, you need a language model and a phonetic dictionary for it. These files define which words OEPocketsphinxController is capable of recognizing. We just created them above by using OELanguageModelGenerator. You also need an acoustic model. OpenEars ships with an English and a Spanish acoustic model.

First, add the following to your implementation (the .m file): Under the @implementation keyword at the top:

```
#import <OpenEars/OEPocketsphinxController.h>
#import <OpenEars/OEAcousticModel.h>
```

> *How to use the class methods:*

In the method where you want to recognize speech (to test this out, add it to your viewDidLoad method), add the following method call:

```
[[OEPocketsphinxController sharedInstance] setActive:TRUE
error:nil];
[[OEPocketsphinxController sharedInstance]
startListeningWithLanguageModelAtPath:lmPath
dictionaryAtPath:dicPath acousticModelAtPath:[OEAcousticModel
pathToModel:@"AcousticModelEnglish"] languageModelIsJSGF:NO]; //
Change "AcousticModelEnglish" to "AcousticModelSpanish" to perform
Spanish recognition instead of English.
```

Warning

OEPocketsphinxController is a singleton which is called with [OEPocketsphinxController sharedInstance]. You cannot initialize an instance of OEPocketsphinxController.

Method Documentation

```
- (void) startListeningWithLanguageModelAtPath: (NSString *)  languageModelPath
                              dictionaryAtPath: (NSString *)  dictionaryPath
                           acousticModelAtPath: (NSString *)  acousticModelPath
                           languageModelIsJSGF: (BOOL)        languageModelIsJSGF
```

Start the speech recognition engine up. You provide the full paths to a language model and a dictionary file which are created using OELanguageModelGenerator and the acoustic model you want to use (for instance [OEAcousticModel pathToModel:"AcousticModelEnglish"]).

```
- (NSError *) stopListening
```

Shut down the engine. You must do this before releasing a parent view controller that contains OEPocketsphinxController.

```
- (void) suspendRecognition
```

Keep the engine going but stop listening to speech until resumeRecognition is called. Takes effect instantly.

```
- (void) resumeRecognition
```

Resume listening for speech after suspendRecognition has been called.

```
- (void) changeLanguageModelToFile: (NSString *) languageModelPathAsString
                    withDictionary: (NSString *) dictionaryPathAsString
```

Change from one language model to another. This lets you change which words you are listening for depending on the context in your app. If you have already started the recognition loop and you want to switch to a different language model, you can use this and the model will be changed at the earliest opportunity. Will not have any effect unless recognition is already in progress. It isn't possible to change acoustic models in the middle of an already-started listening loop, just language model and dictionary.

```
- (void) runRecognitionOnWavFileAtPath: (NSString *) wavPath
              usingLanguageModelAtPath: (NSString *) languageModelPath
                      dictionaryAtPath: (NSString *) dictionaryPath
                   acousticModelAtPath: (NSString *) acousticModelPath
                   languageModelIsJSGF: (BOOL)       languageModelIsJSGF
```

You can use this to run recognition on an already-recorded WAV file for testing. The WAV file has to be 16-bit and 16000 samples per second.

```
- (void) requestMicPermission
```

You can use this to request mic permission in advance of running speech recognition.

```
+ (OEPocketsphinxController *) sharedInstance
```

The OEPocketsphinxController singleton, used for all references to the object.

```
- (BOOL) setActive: (BOOL)       active
             error: (NSError **) outError
```

This needs to be called with the value TRUE before setting properties of OEPocketsphinxController for the first time in a session, and again before using OEPocketsphinxController in case it has been called with the value FALSE.

Property Documentation

```
- (Float32) pocketsphinxInputLevel
```

Gives the volume of the incoming speech. This is a UI hook. You can't read it on the main thread or it will block.

```
- (BOOL) micPermissionIsGranted
```

Returns whether your app has record permission. This is expected to be used after the user has at some point been prompted with requestMicPermission and the result has come back in the permission results OEEventsObserver delegate methods. If this is used before that point, accuracy of results are not guaranteed. If the user has either granted or denied permission in the past, this will return a boolean indicating the permission state.

```
- (float) secondsOfSilenceToDetect
```

This is how long OEPocketsphinxController should wait after speech ends to attempt to recognize speech. This defaults to .7 seconds.

```
- (BOOL) returnNbest
```

Advanced: set this to TRUE to receive n-best results.

```
- (int) nBestNumber
```

Advanced: the number of n-best results to return. This is a maximum number to return – if there are null hypotheses fewer than this number will be returned.

```
- (BOOL) verbosePocketSphinx
```

Turn on extended logging for speech recognition processes. In order to get assistance with a speech recognition issue in the forums, it is necessary to turn this on and show the output.

```
- (BOOL) returnNullHypotheses
```

By default, OEPocketsphinxController won't return a hypothesis if for some reason the hypothesis is null (this can happen if the perceived sound was just noise). If you need even empty hypotheses to be returned, you can set this to TRUE before starting OEPocketsphinxController.

```
- (BOOL) isSuspended
```

Check if the listening loop is suspended

```
- (BOOL) isListening
```

Check if the listening loop is in progress

```
- (BOOL) removingNoise
```

Try not to decode probable noise as speech (this can result in more noise robustness, but it can also result in omitted segments â€“ defaults to YES, override to set to NO)

```
- (BOOL) removingSilence
```

Try not to decode probable silence as speech (this can result in more accuracy, but it can also result in omitted segments â€“ defaults to YES, override to set to NO)

```
- (float) vadThreshold
```

Speech/Silence threshhold setting. You may not need to make any changes to this, however, if you are experiencing quiet background noises triggering speech recognition, you can raise this to a value from 2-5 to 3.5 for the English acoustic model, and between 3.0 and 4.5 for the Spanish acoustic model. If you are experiencing too many words being ignored you can reduce this. The maximum value is 5.0 and the minimum is .5. For the English model, values less than 1.5 or more than 3.5 are likely to lead to poor results. For the Spanish model, higher values can be used. Please test any changes here carefully to see what effect they have on your user experience.

```
- (NSString*) audioMode
```

If you are using 5.0 or greater, you can set audio modes for the audio session manager to use. This can be set to the following:

```
@"Default" to use AVAudioSessionModeDefault
@"VoiceChat" to use AVAudioSessionModeVoiceChat
@"VideoRecording" AVAudioSessionModeVideoRecording
@"Measurement" AVAudioSessionModeMeasurement
```

If you don't set it to anything, default will automatically be used.

```
- (NSString*) pathToTestFile
```

By setting pathToTestFile to point to a recorded audio file you can run the main Pocketsphinx listening loop (not runRecognitionOnWavFileAtPath but the main loop invoked by using startListeningWithLanguageModelAtPath:) over a pre-recorded audio file instead of using it with live input.

In contrast with using the method runRecognitionOnWavFileAtPath to receive a single recognition from a file, with this approach the audio file will have its buffers injected directly into the audio driver circular buffer for maximum fidelity to the goal of testing the entire codebase that is in use when doing a live recognition, including the whole driver and the listening loop including all of its features. This is for creating tests for yourself and for sharing automatically replicable issue reports with Politepix.

To use this, make an audio recording on the same device (i.e., if you are testing OEPocketsphinxController on an iPhone 5 with the internal microphone, make a recording on an iPhone 5 with the internal microphone, for instance using Apple's Voice Memos app) and then convert the resulting file to a 16-bit, 16000 sample rate, mono WAV file. You can do this with the output of Apple's Voice Memos app by taking the .m4a file that Voice Memos outputs and run it through this command in Terminal.app:

```
afconvert -f WAVE -d LEI16@16000 -c 1 ~/Desktop/Memo.m4a
~/Desktop/Memo.wav
```

Then add the WAV file to your app, and right before sending the call to startListeningWithLanguageModelAtPath, set this property pathToTestFile to the path to your audio file in your app as an NSString (e.g. [[NSBundle mainBundle] pathForResource:"Memo" ofType:@"wav"]).

Note: when you record the audio file you will be using to test with, give it a second of quiet lead-in before speech so there is time for the engine to fully start before listening begins.

SmartCMN is disabled during testing so that the test gets the same results when run for different people. Please keep in mind that there are some settings in Pocketsphinx which may prevent a deterministic outcome from a recognition, meaning that you should expect a **similar** score over multiple runs of a test but you may not always see the **identical** score. For this reason and the fact that OEPocketsphinxController is asynchronous and results in real practice are delivered via uncoupled callback it has not been designed as a purely automated test, but as an observed practical test.

If it were designed as a purely automated test it would be testing something other than the way OEPocketsphinxController/OEEventsObserver works in an app, which is designed for good speech implementations rather than tests.

```
- (BOOL) playbackTestFileDuringTest
```

If you are using a test file, you can set playbackTestFileDuringTest to TRUE in order to hear the test file playing back out of the device as it is run. This is an early experimental version of this functionality, implemented as simply as possibly, and it might not work very well yet. It is unlikely to be perfectly synchronized with the actual engine head position in the test file, but it should be off by less than a quarter-second so you should be able to understand where you are in the recording when turning this on.

There is a big warning when using this, which is that it will be emitted with an AVAudioPlayer, which means that if what you are testing also uses AVAudioPlayers or things which conflict with them, and/or you are reading AVAudioPlayer delegates for particular events, this setting is almost certainly incompatible with your test case and you should run your test file silently.

```
- (BOOL) useSmartCMNWithTestFiles
```

If you are doing testing, you can toggle SmartCMN on or off (it defaults to off and should usually be left off since using it can lead to nondeterministic results on the first runs with new devices).

<OEEventsObserverDelegate> Protocol Reference

Detailed Description

OEEventsObserver provides a large set of delegate methods that allow you to receive information about the events in OpenEars from anywhere in your app. You can create as many OEEventsObservers as you need and receive information using them simultaneously.

Usage examples

> *What to add to your header:*

OEEventsObserver is the class which keeps you continuously updated about the status of your listening session, among other things, via delegate callbacks. Add the following lines to your header (the .h file). Under the imports at the very top:

```
#import <OpenEars/OEEventsObserver.h>
```

at the @interface declaration, add the OEEventsObserverDelegate inheritance. An example of this for a view controller called ViewController would look like this:

```
@interface ViewController : UIViewController
<OEEventsObserverDelegate> {
```

And add this property to your other class properties (OEEventsObserver must be a property of your class or it will not work):

```
@property (strong, nonatomic) OEEventsObserver
*openEarsEventsObserver;
```

> *What to add to your implementation:*

Add the following to your implementation (the .m file): Before you call a method of either OEFliteController or OEPocketsphinxController (perhaps in viewDidLoad), instantiate OEEventsObserver and set its delegate as follows:

```
self.openEarsEventsObserver = [[OEEventsObserver alloc] init];
[self.openEarsEventsObserver setDelegate:self];
```

> *How to use the class methods:*

Add these delegate methods of OEEventsObserver to your class, which is where you will receive information about received speech hypotheses and other speech UI events:

```
- (void) pocketsphinxDidReceiveHypothesis:(NSString *)hypothesis
recognitionScore:(NSString *)recognitionScore utteranceID:(NSString
*)utteranceID {
 NSLog(@"The received hypothesis is %@ with a score of %@ and an ID
of %@", hypothesis, recognitionScore, utteranceID);
}

- (void) pocketsphinxDidStartListening {
 NSLog(@"Pocketsphinx is now listening.");
}

- (void) pocketsphinxDidDetectSpeech {
 NSLog(@"Pocketsphinx has detected speech.");
}

- (void) pocketsphinxDidDetectFinishedSpeech {
 NSLog(@"Pocketsphinx has detected a period of silence, concluding
an utterance.");
}

- (void) pocketsphinxDidStopListening {
 NSLog(@"Pocketsphinx has stopped listening.");
}

- (void) pocketsphinxDidSuspendRecognition {
 NSLog(@"Pocketsphinx has suspended recognition.");
}

- (void) pocketsphinxDidResumeRecognition {
```

```
- (void) pocketsphinxDidResumeRecognition {
 NSLog(@"Pocketsphinx has resumed recognition.");
}

- (void) pocketsphinxDidChangeLanguageModelToFile:(NSString
*)newLanguageModelPathAsString andDictionary:(NSString
*)newDictionaryPathAsString {
 NSLog(@"Pocketsphinx is now using the following language model:
\n%@ and the following dictionary:
%@",newLanguageModelPathAsString,newDictionaryPathAsString);
}

- (void) pocketSphinxContinuousSetupDidFailWithReason:(NSString
*)reasonForFailure {
 NSLog(@"Listening setup wasn't successful and returned the failure
reason: %@", reasonForFailure);
}

- (void) pocketSphinxContinuousTeardownDidFailWithReason:(NSString
*)reasonForFailure {
 NSLog(@"Listening teardown wasn't successful and returned the
failure reason: %@", reasonForFailure);
}

- (void) testRecognitionCompleted {
 NSLog(@"A test file that was submitted for recognition is now
complete.");
}
```

Warning

It is a requirement that any OEEventsObserver you use in a view controller or other object is a property of that object, or it won't work.

Method Documentation

```
- (void) audioSessionInterruptionDidBegin
```

There was an interruption.

```
- (void) audioSessionInterruptionDidEnd
```

The interruption ended.

```
- (void) audioInputDidBecomeUnavailable
```

The input became unavailable.

```
- (void) audioInputDidBecomeAvailable
```

The input became available again.

```
- (void) audioRouteDidChangeToRoute: (NSString *) newRoute
```

The audio route changed.

```
- (void) pocketsphinxRecognitionLoopDidStart
```

Pocketsphinx isn't listening yet but it has entered the main recognition loop.

```
- (void) pocketsphinxDidStartListening
```

Pocketsphinx is now listening.

```
- (void) pocketsphinxDidDetectSpeech
```

Pocketsphinx heard speech and is about to process it.

```
- (void) pocketsphinxDidDetectFinishedSpeech
```

Pocketsphinx detected a second of silence indicating the end of an utterance

```
- (void) pocketsphinxDidReceiveHypothesis: (NSString *) hypothesis
                         recognitionScore: (NSString *) recognitionScore
                              utteranceID: (NSString *) utteranceID
```

Pocketsphinx has a hypothesis.

```
- (void) pocketsphinxDidReceiveNBestHypothesisArray: (NSArray *) hypothesisArray
```

Pocketsphinx has an n-best hypothesis dictionary.

```
- (void) pocketsphinxDidStopListening
```

Pocketsphinx has exited the continuous listening loop.

```
- (void) pocketsphinxDidSuspendRecognition
```

Pocketsphinx has not exited the continuous listening loop but it will not attempt recognition.

```
- (void) pocketsphinxDidResumeRecognition
```

Pocketsphinx has not existed the continuous listening loop and it will now start attempting recognition again.

```
- (void) pocketsphinxDidChangeLanguageModelToFile: (NSString *) newLanguageModelPathAsString
                                    andDictionary: (NSString *) newDictionaryPathAsString
```

Pocketsphinx switched language models inline.

```
- (void) pocketSphinxContinuousSetupDidFailWithReason: (NSString *) reasonForFailure
```

Some aspect of setting up the continuous loop failed, turn onOELogging for more info.

```
- (void) pocketSphinxContinuousTeardownDidFailWithReason: (NSString *) reasonForFailure
```

Some aspect of tearing down the continuous loop failed, turn onOELogging for more info.

```
- (void) pocketsphinxTestRecognitionCompleted
```

Your test recognition run has completed.

```
- (void) pocketsphinxFailedNoMicPermissions
```

Pocketsphinx couldn't start because it has no mic permissions (will only be returned on iOS7 or later).

```
- (void) micPermissionCheckCompleted: (BOOL) result
```

The user prompt to get mic permissions, or a check of the mic permissions, has completed with a TRUE or a FALSE result (will only be returned on iOS7 or later).

```
 - (void) fliteDidStartSpeaking
```

Flite started speaking. You probably don't have to do anything about this.

```
 - (void) fliteDidFinishSpeaking
```

Flite finished speaking. You probably don't have to do anything about this.