AT&T API Platform

Speech SDK for iOS™

Publication Date: August 29 2013

Legal Disclaimer

Table of Contents

Contents

1 Introduction 1

1.1 Sample Code 1

1.2 Related Documentation 1

1.3 Terms and Acronyms 2

2 Introduction to Speech API and Speech SDK 3

2.1 Speech API Overview 3

2.1.1 RESTful Web Service 3

2.1.2 Speech Contexts 4

2.1.3 AT&T Developer Program On-Boarding 5

2.1.4 Security and Authentication 5

2.2 Speech SDK Overview 7

2.2.1 Platform Integration 7

2.2.2 Audio Capture and Streaming 7

2.2.3 Recording and Progress User Interface 8

3 Speech Recognition Web Service 9

3.1 Speech Recognition Requests 9

3.1.1 Speech Recognition Service URL 9

3.1.2 Speech Recognition Request HTTP Headers 9

3.2 Speech Recognition Request Formats 10

3.2.1 Audio Request Format 11

3.2.2 Inline Grammar Request Format 11

3.3 Request Streaming 12

3.4 Extra Arguments 13

3.5 Using OAuth Credentials 14

Table of Contents

3.5.1 OAuth Access Token Request 14
3.5.2 OAuth Access Token Response 14
3.5.3 Authorization Header 15
3.6 Speech Recognition Responses 16
3.6.1 Response Status Codes 16
3.7 Speech Recognition Response Formats 17
3.8 Speech to Text JSON Format 17
3.8.1 JSON Nomenclature 18
3.8.2 General Assumptions about the JSON Data 19
3.8.3 JSON Objects and Fields 19
3.9 Speech Recognition Examples 22
3.9.1 Running cURL 22
3.9.2 Example of an OAuth Request 22
3.9.3 Example of Successful Speech Recognition Response 23
3.9.4 Example of Unauthorized Credentials 24
3.9.5 Example of Error Response 25
4 Using Speech SDK for iOS 26
4.1 How the Speech SDK Works on iOS 26
4.2 Speech SDK Prerequisites for iOS 27
4.3 Setting up your iOS Project 27
4.4 Speech Recognition Tasks on iOS 30
4.4.1 Configure ATTSpeechService Runtime Properties 31
4.4.2 Acquire OAuth Access Token 32
4.4.3 Start Speech Interaction 33
4.4.4 Send Inline Grammar Data 33
4.4.5 Customize UI 34

Table of Contents

4.4.6 Speech Endpointing 35

4.4.7 Handle Speech Recognition Results and Errors 35

4.5 Testing Speech Applications on iOS 37

4.6 Deploying Speech Applications on iOS 37

5 Speech SDK Reference for iOS 38

5.1 ATTSpeechService Methods 38

5.2 ATTSpeechService Request Properties and Constants 41

5.3 ATTSpeechService Delegate Callback Methods 47

5.4 ATTSpeechService Response Properties 50

5.5 ATTSpeechService Error Codes 51

5.6 ATTSpeechService State Transitions 53

Table of Figures

Figure 4-1: Selecting the ATTSpeechKit subfolder. 27
Figure 4-2: Dragging the ATTSpeechKit folder to your Xcode window. 28
Figure 4-3. Completed Xcode target. 30
Figure 4-4: Speech SDK standard UI on iOS. 34

Table of Tables

Table 1-1: Terms and acronyms. 2
Table 3-1: Speech recognition request HTTP headers. 10
Table 3-2: Inline grammar parts. 12
Table 3-3: Speech SDK X-Arg parameters 13
Table 3-4: Response status codes. 16
Table 3-5: Response status codes. 17
Table 3-6: Speech recognition response formats. 17
Table 3-7: cURL options. 22
Table 5-1: ATTSpeechService methods. 40
Table 5-2: ATTSpeechService request properties. 45
Table 5-3: ATTSpeechService request property constants. 46
Table 5-4: ATTSpeechService delegate callback methods. 49
Table 5-5: ATTSpeechService response properties 50
Table 5-6: ATTSpeechService error codes. 52
Table 5-7: ATTSpeechService state transitions. 53

Table of Examples

Example 3-1: HTTP headers of the request. 10
Example 3-2: X-Arg header. 13
Example 3-3: OAuth access token response. 15
Example 3-4: Schematic hierarchy for the JSON format of a successful transcription. 18
Example 3-5: cURL OAuth request. 23
Example 3-6: Successful cURL request. 24
Example 3-7: Unauthorized cURL response. 24
Example 3-8: cURL error response. 25
Example 4-1: Configure `ATTSpeechService` in `applicationDidBecomeActive:`. 32
Example 4-2: Start speech interaction on iOS. 33
Example 4-3: Send inline grammar on iOS. 34
Example 4-4: Successful Transaction delegate method. 36
Example 4-5: Failed transaction delegate method. 36

1 Introduction

This document is designed for software developers on the iOS platform who intend to build applications using the Speech API and Speech SDK provided by AT&T.

The purpose of this document is to explain the configuration, development, and testing of applications that use the Speech SDK for iOS. This document should provide developers with a thorough understanding of all Speech API and Speech SDK data formats, operations, and parameters.

Note: The Speech SDK supports application development on the Android and iOS platforms. This document has full details on developing for iOS.

1.1 Sample Code

Sample code and response messages for most common tasks are provided in this document. For more complete sample applications, visit the AT&T Developer Program website at the following location: https://developer.att.com/developer/forward.jsp?passedItemId=12500023

1.2 Related Documentation

For additional information on the Speech API, refer to the following documentation:

- ***Speech API Technical Documentation:*** Contains detailed information on the RESTful Speech API provided by AT&T, including request parameters and the format of the response data. The technical documentation is found at the following location: https://developer.att.com/developer/forward.jsp?passedItemId=12500023
- ***Speech SDK Release Notes:*** The distribution of Speech SDK includes a text file with release notes, describing the changes in that release of the SDK.

1.3 Terms and Acronyms

The following table defines terms and acronyms used in this document.

Acronym	Definition
AMR	Adaptive Multi-Rate audio format
API	Application Programming Interface
APK	Application Package
JSON	JavaScript Object Notation format
MIME	Multipurpose Internet Mail Extensions
OS	Operating System
RESTful	Representational State Transfer
SDK	Software Development Kit
SMS	Short Message Service
Speex	A patent-free audio compression format designed for speech
SSL	Secure Sockets Layer
UI	User interface
WAV	Waveform audio format

Table 1-1: Terms and acronyms.

2 Introduction to Speech API and Speech SDK

This section describes the core functions of the AT&T Speech API and the Speech SDK, including features of the web service and client libraries.

2.1 Speech API Overview

The Speech API provides speech recognition and generation for third-party apps using a client-server RESTful architecture. The Speech API supports HTTP 1.1 clients and is not tied to any wireless carrier. The Speech API includes the following web services.

- **Speech to Text:** Performs speech recognition, accepting audio data and returning a text transcription of the speech in the audio. Powered by the AT&T Watson^{SM} speech engine, this web service includes several speech contexts that perform speech recognition that are optimized for particular usage scenarios. It can also accept custom grammar information to enhance recognition in application-specific domains.
- **Speech to Text Custom:** An extension of the Speech to Text service that accepts custom grammar information in addition to speech data. This allows enhanced recognition in application-specific domains.
- **Text to Speech:** Generates audio that speaks a snippet of text. The web service accepts text data and returns an audio stream that the application can play to the user. Using AT&T Natural Voices$^{®}$ technology, the service lets applications customize the language, voice, and tempo of the spoken audio.

The bulk of this document covers the Speech to Text and Speech to Text Custom services. The phrase “speech recognition service” refers to the features that both web services have in common.

2.1.1 RESTful Web Service

The Speech API converts speech to text in a RESTful web service architecture. Client applications send an HTTP POST request that contains audio data and optional grammar information to the web service. The server returns transcription text in the HTTP response.

The recognition services support a streaming form of speech processing. Client applications can use the HTTP standard chunked encoding to send audio data while the user is speaking. The recognition service will perform speech processing incrementally as it receives data from the client. This reduces the perceived latency of getting the transcription response.

The Speech API is a fully stateless web service. It neither maintains persistent connections with clients nor does it keeps track of sessions among requests.

2.1.2 Speech Contexts

Speech applications can have many different usage scenarios with different requirements for vocabulary, accuracy, and speed of transcription. For example, a local search app might want the user to speak short keyword phrases such as “find restaurants near me”, whereas a dictation program might want the user to speak full natural sentences. The Speech API supports these various scenarios with a set of speech contexts. Each speech context is tuned for a different scenario of user vocalization.

The contexts for Speech to Text include the following:

- **Voice Mail (**`VoiceMail`**)** Transcribes long-form speech, such as voice mail messages. The audio comes from the client app; the API does not provide access to voice mail systems.
- **Business Search (**`BusinessSearch`**):** Recognizes short speech phrases that include businesses and geographic locations. The resulting text can be used as input to a Yellow Pages search engine; it is the responsibility of the client app to parse the transcribed text and perform actual searches for the user.
- **Question and Answer (**`QuestionAndAnswer`**):** Transcribes short speech sentences that include questions. The resulting text can be input to a question-answering engine; it is the responsibility of the client app to parse the transcribed text and perform actual queries for the user.
- **Web Search (**`WebSearch`**):** Recognizes short speech query phrases that are common in Web searches. The resulting text can be passed to a web search engine; it is the responsibility of the client app to parse the transcribed text and perform actual searches for the user.
- **Short Message Service (**`Sms`**):** Transcribes spoken audio into text output for text messaging. The speech package works best with spoken audio content that is typical of texting between two individuals (for example, "hi, I am running fifteen minutes late"). The resulting text can be used as the body of a text message; it is the responsibility of the client app to submit the text to any external messaging systems.
- **Generic (**`Generic`**):** Recognizes a broad range of full sentences.
- **Television (**`TV`**):** Recognizes search terms that include the names of television shows, actors, and networks. The resulting text can be used as input to an electronic programming guide (EPG) search engine.

The Speech to Text Custom service also allows applications to use custom recognition grammars. This is geared towards for use cases with a specialized vocabulary. The contexts for Speech to Text Custom include the following:

- **Generic with Hints (`GenericHints`):** Recognizes both a broad range of full sentences and the specific phrases contained in the custom grammar.
- **Grammar List (`GrammarList`):** Optimized for command interfaces, it recognizes only the phrases contained in the custom grammar.

2.1.3 AT&T Developer Program On-Boarding

The Speech API is one of several APIs provided by AT&T that are available to third-party developers. Developers who use the Speech API must first enroll in the AT&T API Program. For more information on the program, see the AT&T Developer Program web site at the following location: http://developer.att.com/

Once you have an account in the program, you must register your app on the AT&T Developer Program web site for access to the Speech API. For each app that you register, the system generates two values called the App Key and Secret Key. Your application will use those values to authenticate itself with the Speech API using the OAuth protocol.

It is very important that you keep the values for your App Key and Secret Key private. The Speech API assumes that all requests with your OAuth credentials are authorized by you. If your App Key and Secret Key are revealed, other parties could make Speech API requests that count against your thresholds. Apps that exceed those thresholds may see subsequent requests throttled or shut down. If that occurs, you should be prepared to issue new versions of your app with new credentials.

Note: You can contact the AT&T Developer Program to make arrangements for high-traffic apps.

2.1.4 Security and Authentication

The Speech API encrypts all speech recognition services through secure HTTPS connections, also known as SSL. This ensures that your app credentials and the users' speech remain secure as they travel across the Internet.

All connections to the Speech API are authenticated with the application developer's credentials. The mechanism for authentication is the OAuth 2.0 protocol, which is a standard authentication protocol for securing API transactions from mobile, desktop, and web applications. For more information on the OAuth standard, see http://oauth.net/.

The Speech API uses the OAuth 2.0 Client Credentials flow for authentication. This is a two-party authentication in which the client app sends credentials to the Speech API that associate the HTTP request to an application developer. There are three steps involved in the Client Credentials flow:

1. The client application sends its App Key (also known as the *client ID*) and Secret Key (also known as the *client secret*) to the Speech API authentication server.
2. The Speech API authentication server returns an access token to the client app. The access token has a lifetime of two hours, after which a new access token must be acquired.
3. The client app includes the access token in subsequent speech-to-text requests that it posts to the Speech API.

Note: The OAuth 2.0 Client Credentials flow does *not* involve the end user of the application. The authentication is two-party flow between the application and the service, not a three-party flow involving the user. There is no need for the user to log in for an app to use the Speech API.

2.2 Speech SDK Overview

The Speech API, as a RESTful web service, works with any internet-connected device. It is straightforward for developers to send audio data over HTTP and interpret the response. However, users demand applications with a responsive user interface and fast speech recognition time. That entails streaming data from the microphone to the server so that Speech API can perform recognition while the user is talking. Writing code on mobile platforms to perform microphone input and stream data to a server can be a challenge.

The Speech SDK is a set of software libraries and sample code for mobile platforms that assist developers in building interactive mobile applications using the Speech API recognition services. The libraries in the Speech SDK handle many of the difficult tasks in performing speech recognition using the Speech API, including:

- Capturing microphone input on a device.
- Compressing audio data for efficient transport.
- Streaming audio data to the Speech API web service.
- Displaying feedback while the microphone is active.
- Providing notifications of success or failure in speech recognition.

It is not required for developers who use the Speech API to also use the Speech SDK. Developers can use the plain RESTful web service interface and make direct HTTP connections to the web services of the Speech API.

2.2.1 Platform Integration

The Speech SDK supports application development on the Android and iOS platforms. This document has full details on developing for iOS.

For each supported platform, the Speech SDK is distributed as a ZIP archive that contains a static library called ATTSpeechKit. When developers link their applications to the ATTSpeechKit library, the Speech SDK code becomes embedded within the application binary that the users run.

2.2.2 Audio Capture and Streaming

The Speech SDK captures audio from the microphone on the client device and generates data in the Speex audio format for the Speech API recognition service. When microphone input starts, the Speech SDK captures audio until it detects the end of the phrase spoken by the user or until it reaches the maximum recording time.

The Speech SDK streams the audio data over a chunked HTTP connection to the speech recognition service simultaneously as it captures data from the microphone. This lets the Speech API server perform speech processing while the user is talking, reducing the perceived latency of recognition.

2.2.3 Recording and Progress User Interface

The Speech SDK provides a standard user interface (UI) that an app can display while the user performs speech input. The UI gives feedback that the microphone has accepted the input and shows the progress of the speech recognition on the server. The UI also provides a way for the user to cancel the request. Displaying the standard interface is optional; apps can implement a custom UI if desired.

3 Speech Recognition Web Service

This section provides information on creating HTTP requests for the Speech API recognition service and interpreting its responses. It focuses on the features relevant to developers using the Speech SDK. For full reference information on the Speech API, see the documentation links at the following location: https://developer.att.com/developer/forward.jsp?passedItemId=12500023.

3.1 Speech Recognition Requests

The Speech to Text and Speech to Text Custom services performs speech recognition on audio data submitted in a POST request. The web service request contains the following parts.

- URL of the web service.
- HTTP headers that contain credentials of the developer, processing instructions, and standard HTTP handling.
- HTTP body that contains audio and grammar data for the service to process.

3.1.1 Speech Recognition Service URL

All Speech API web service requests connect to a URL that varies based on the actual service that the app has been on-boarded to. The URL for the speech recognition service has the template:

`https://`*`<hostname>`*`/speech/v3/`*`<service>`*

Where the fields of the URL are as follows:

- *`<hostname>`*: The address of the Speech API server. The production hostname is `api.att.com`. Applications on-boarded to other versions of the service use different hostnames.
- *`<service>`*: The name of the speech recognition service. Use "`speechToText`" for the Speech to Text service, and "`speechToTextCustom`" for the Speech to Text Custom service.

Note: The Speech API recognition service does not use query strings in URLs. All parameter data are contained in HTTP headers.

3.1.2 Speech Recognition Request HTTP Headers

All requests to the Speech recognition service use a common set of HTTP headers. The following table describes the HTTP headers that a client app can add to the request.

Header	Required	Definition
`X-SpeechContext`	Yes	This header contains the name of the speech context, such as `VoiceMail` or `BusinessSearch`. For more information on speech context, see Section 2.1.2 Speech Contexts.
`Authorization: Bearer`	Yes	This header contains the OAuth access token that authenticates this request. For more information on generating this header, see Section 3.5 Using OAuth Credentials.
`Content-Type`	Yes	This header is the MIME type of the data that the client app sends in the POST request. For information on the supported content types, see Section 3.2 Speech Recognition Request Formats.
`Transfer-Encoding: chunked` or `Content-Length`	Yes	These headers indicate a streaming or non-streaming request. For more information, see Section 2.2.2 Audio Capture and Streaming.
`Accept`	No	This header is the MIME type that the client wants for the response. For a list of the supported response types, see Section 3.7 Speech Recognition Response Formats.
`X-Arg`	No	This header (which can appear multiple times) contains a set of extra parameters for the request. For more information on the supported parameters see Section3.4 Extra Arguments.

Table 3-1: Speech recognition request HTTP headers.

The following example illustrates the HTTP headers of the request.

Request Headers

```
X-SpeechContext: BusinessSearch
Authorization: Bearer ZmFrZWVudHJ5
Content-Type: audio/amr
Transfer-Encoding: chunked
```

Example 3-1: HTTP headers of the request.

3.2 Speech Recognition Request Formats

The Speech API recognition services accept audio data in the body of the HTTP POST request. The web service processes the speech in the audio data and returns the text transcription. The Speech to Text service requires that the body of the HTTP request contains the audio data directly. It must not, for example, be wrapped in a HTML form upload format.

Clients of the Speech to Text Custom service must submit grammar, audio, and optional pronunciation data in the HTTP request. The Speech to Text Custom service requires the HTTP body to be in a MIME multipart format, which combines the grammar, pronunciation, and audio data into one HTTP request. This document will refer to the multipart format as the *inline grammars* format.

3.2.1 Audio Request Format

Audio can be submitted in several formats, which are detailed in the Speech API technical documentation. The Speech SDK libraries can capture audio in the following formats:

- AMR (Adaptive Multi-Rate) narrowband format, 12.2 kbit/sec, 8 kHz sampling, MIME type `audio/amr`.
- Speex wideband format, 16 kHz sampling, variable bitrate (averaging 14 kbit/sec), MIME type `audio/x-speex-with-header-byte;rate=16000`.
- WAV (Waveform) format, 16 bit PCM data, single channel, 8 or 16 kHz sampling, 128 or 156 kbit/sec, MIME type `audio/wav`. The Speech SDK sends a special “streaming” version of WAV in which the RIFF header doesn’t specify a file length.

3.2.2 Inline Grammar Request Format

The inline grammars format has a content type of `multipart/x-srgs-audio`. It is a specialization of the standard MIME `multipart/form-data` format. This section describes the key features of the inline grammars format for Speech SDK users. Refer to the Speech API technical documentation for a full description of this format. For details on the underlying `multipart/form-data` format, see the standard RFC 2388 at the following location: http://tools.ietf.org/html/rfc2388

The multipart data format consists of several sections (or parts), and each part contains a part body and a collection of part headers. The following headers are required for each part:

- `Content-Type`: The MIME type of the data in the part.
- `Content-Disposition`: The role that the part plays in the overall request.

A request in `multipart/x-srgs-audio` format is required to have 2 or 3 parts in a specific order and with specific content disposition values. The following table describes the parts in the required order:

Role	Required	Content Disposition	Content Type	Definition
Dictionary	No	`form-data; name="x-dictionary"`	`application/pls+xml`	Pronunciation lexicon of words in grammar. Data are in standard PLS format
Grammar	Yes	`form-data; name="x-grammar"`	`application/srgs+xml`	Context-free grammar describing the recognizable phrases. Data are in standard SRGS format.
Speech	Yes	`form-data; name="x-voice"`	any audio format supported by Speech API	Speech data to transcribe into text.

Table 3-2: Inline grammar parts.

For more information on the contents of the grammar and dictionary parts, see the Speech API technical documentation.

3.3 Request Streaming

The Speech API recognition service supports clients that use HTTP streaming when posting audio data to the service. In a streaming request, the service performs speech processing while receiving data from the client. By contrast, in a non-streaming request, the service waits until it receives all data before performing speech recognition.

A client signals a streaming request to the server through a standard HTTP 1.1 mechanism called chunked transfer encoding. Adding the `Transfer-Encoding: chunked` header marks a streaming request. When the streaming request is signaled by adding this header, the client segments the posted data into chunks; a chunk size of 512 bytes is good for speech recognition. For more details on chunked transfer encoding, see the following HTTP specification: http://tools.ietf.org/html/rfc2616#section-3.6.1

If the client does not include a `Transfer-Encoding` header, it must include a `Content-Length` header instead, which specifies the number of bytes being sent in the request.

3.4 Extra Arguments

The Speech API recognition service allows client applications to tune the speech recognition process by specifying extra parameters in the request. These arguments are passed in `X-Arg` HTTP headers in the request. The header consists of a set of name-value pairs which are comma delimited. Special characters in the values must be percent-encoded; note that this is different from "form encoding" in that space characters must be encoded as `%20`, not as a plus character. Refer to the Speech API technical documentation for a full list of arguments and their usage with different speech contexts.

The following is an example of an `X-Arg` header:

X-Arg Header

```
X-Arg: ClientSdk=ATTSpeechKit-Android-1.6.0,
   DeviceType=HTC%09HTC%20PH39100,DeviceOs=Android-2.3.4,
   DeviceTime=2013-03-13%2020%3A05%3A51%20PDT,
   ClientApp=example.myapp,ClientVersion=1.0,ClientScreen=main
```

Example 3-2: X-Arg header.

The following table lists a subset of the `X-Arg` parameters that the clients can add to a recognition request. The Speech SDK adds most of these automatically.

Parameter	Definition
`ClientSdk`	The name and version of the Speech SDK software capturing audio for the Speech API. The value has the format `<name>-<platform>-<version>`.
`DeviceType`	The manufacturer and model name of device making the request. The value has the format `<manufacturer>\t<model>`.
`DeviceOs`	The name and version of the operating system of the device. The value has the format `<name>-<version>`.
`DeviceTime`	The local time on the device, including the local time zone. Use the Unix strftime format "`%Y-%m-%d %H:%M:%S %Z`" or the Java SimpleDateFormat "`yyyy-MM-dd HH:mm:ss z`".
`ClientApp`	The unique name of the app making the request. On iOS use the bundle identifier.
`ClientVersion`	The version of the app making the request.
`ClientScreen`	An identifier of the screen or mode of the application as it makes the request. It is recommended to use the class name of the iOS UIViewController making the request. (Speech SDK does not automatically add this value.)

Table 3-3: Speech SDK X-Arg parameters

3.5 Using OAuth Credentials

Each HTTP request sent to the Speech API must be authenticated using the OAuth 2.0 Client Credentials flow. Client applications do this by adding an `Authorization` HTTP header with valid access token to their requests. Tokens have a lifetime of two hours. When a token is not valid, the web service request fails with an HTTP 401 `Unauthorized` error.

3.5.1 OAuth Access Token Request

To get a new access token, the client app makes a POST request to the Speech API authentication server. The URL of the token request has the format:

```
https://<auth-hostname>/oauth/token
```

The fields of the URL are as follows:

- `<auth-hostname>` : The address of the Speech API authentication server. Production and test versions of the service use different hostnames.

The body of the POST request includes the App Key and Secret Key of the app developer. The request body uses standard form encoding (`application/x-www-form-urlencoded`) and has the following template:

```
client_id=<app-key>&client_secret=<secret-key>&
grant_type=client_credentials&scope=<OAuth-scope>
```

The fields of the request body are:

- `client_id`*:* The OAuth client ID (App Key) for the app of the developer.
- `client_secret`*:* The OAuth client secret (Secret Key) for the app of the developer.
- `scope` — A parameter that describes the services that the app will use. For the Speech to Text service, use a scope of “`SPEECH`”; for Speech to Text Custom use “`STTC`”.

The developer obtains these values from the AT&T Developer Program web site. It is very important that the developer keep the OAuth parameters secret. The code generating the values should be obfuscated when deployed in apps.

3.5.2 OAuth Access Token Response

If the credentials are valid, the authentication server returns an HTTP 200 response. The body of the response is a JSON document, and the access token is in the JSON field `access_token`. The response has the following template:

OAuth Access Token Response

```
{
  "access_token":"<access-token>",
  "expires_in":"<seconds>"
  "refresh_token":"<refresh-token>"
}
```

Example 3-3: OAuth access token response.

The OAuth response contains the following fields:

- `access_token`***:*** The string value of the OAuth access token. Add this value to the `Authorization` header of the speech recognition request.
- `refresh_token`***:*** The string value of the OAuth refresh token. When the access token expires, the refresh token can be used as an alternative to sending the App Key and Secret Key in an OAuth token request. The refresh token has a lifetime of about one day. After the refresh token expires, the client app has to use the App Key and Secret Key to renew an access token. As a consequence, it's probably easier for a client app to forego using refresh tokens and simply use the App Key and Secret Key for authentication.
- `expires_in`***:*** The number of seconds that the token is valid. When this value is 0, the token has an unspecified duration. It is recommended that apps request a new access token every two hours.

If the credentials are not accepted, the authentication server returns an HTTP 401 response. The following example shows a response for unauthorized credentials:

```
{
  "error":"invalid_client"
}
```

3.5.3 Authorization Header

Once the client application has a valid OAuth access token, it adds the token to the `Authorization` header of the HTTP requests it makes to the Speech API. The header has the following template:

```
Authorization: Bearer <access-token>
```

The fields of the header are:

- `<access-token>`***:*** The string value of the OAuth access token. This value comes from the JSON returned by the OAuth token request.

3.6 Speech Recognition Responses

The Speech API returns an HTTP response after processing a speech recognition request. The response contains the following data:

- HTTP status code, which signals the success or failure of the request
- HTTP headers that can contain metadata about the response
- HTTP body that contains structured data with the recognized text or error.

3.6.1 Response Status Codes

The status code indicates whether a Speech API request was processed successfully or had an error. The following table describes the status codes returned by the Speech API.

Code	Meaning	Description
200	Success	The speech request was processed successfully. The HTTP body will contain data in the requested format with the recognized text.
400	Client Error	The request was malformed. Some data in the request could not be processed by the web service.
401	Authorization Required	The request did not include an OAuth token or the token was not valid.
500 or 503	Server Error	An error occurred on the server while processing the request.

Table 3-4: Response status codes.

For more information on processing the response codes, see Section 3.9 Speech Recognition Examples.

The following table describes the HTTP headers that are part of a Speech API response.

Header	Description
`Content-Type`	The MIME type of the HTTP body of the response. For details on response types, see Section 3.7 Speech Recognition Response Formats.
`Transfer-Encoding: chunked` or `Content-Length`	The server may either send the response incrementally by using a chunked encoding, or it may send the response all at once by specifying a content size.

Header	Description
`WWW-Authenticate: Bearer`	This header is included in the error responses when the request omits the credentials of the developer.

Table 3-5: Response status codes.

Note: The server may also send custom `X-` prefixed headers. However, they are for internal use and clients can ignore them.

3.7 Speech Recognition Response Formats

When the Speech API recognition service is able to process the audio in a request, it returns structured data in the response that describes the speech information in the audio. The following table lists the formats that the service can return.

Format	MIME type	Description
Speech to Text JSON	`application/json`	The default format returned by the Speech to Text and Speech to Text Custom services. The Speech SDK library is able to parse this format for an application. For more information on this format, see Section 3.8 Speech to Text JSON Format.
Speech to Text XML	`application/xml`	An XML rendering of the same information as the Speech to Text JSON format.
EMMA	`application/emma+xml`	A W3C standard format for recognition data.

Table 3-6: Speech recognition response formats.

For complete reference information on the three formats, see the technical documentation for the Speech API.

3.8 Speech to Text JSON Format

The default format for a response from the Speech API recognition service is a structured JSON document. This section describes the fields of that format that are most often used by client apps. For complete reference information on the fields of the JSON document, see the technical documentation for the Speech API.

The following example illustrates the schematic hierarchy for the JSON format of a successful transcription.

Speech Recognition Response

```
{
:   "Recognition":
:   {
:   :   "ResponseId":"e6bf3236ad938ca19c28f95a9065c813",
:   :   "NBest":
:   :   [
:   :   :   {
:   :   :   :   "WordScores":
:   :   :   :   [
:   :   :   :   :   0.159,
:   :   :   :   :   0.569,
:   :   :   :   :   0.569
:   :   :   :   ],
:   :   :   :   "Confidence":0.433333327,
:   :   :   :   "Grade":"accept",
:   :   :   :   "ResultText":"Hey Boston Celtics.",
:   :   :   :   "Words":
:   :   :   :   [
:   :   :   :   :   "Hey",
:   :   :   :   :   "Boston",
:   :   :   :   :   "Celtics."
:   :   :   :   ],
:   :   :   :   "LanguageId":"en-us",
:   :   :   :   "Hypothesis":"Hey Boston Celtics."
:   :   :   }
:   :   ]
:   }
}
```

Example 3-4: Schematic hierarchy for the JSON format of a successful transcription.

The useful data in the response is contained in the `Recognition` object. There is an `NBest` field within the `Recognition` object, which is an array of possible ways to interpret the speech. In general, the `NBest` array has a single item, as in this example. The actual transcribed text is in the `Hypothesis` field. For a full description of the JSON objects and fields in the response format, see Section 3.8.3 JSON Objects and Fields.

3.8.1 JSON Nomenclature

The following terms are used to describe the JSON format:

- **Field:** A key-value pair.
- **Class:** A pre-defined set of key-type pairs that can describe an object.

- **Object:** A brace-delimited collection of key-value pairs, each of which is defined in the object's class.
- **Required field:** A key-value pair that must be present in an object of a given class.
- **Optional field:** A key-value pair that may be omitted in an object of a given class.

3.8.2 General Assumptions about the JSON Data

The following assumptions can be made about the JSON data returned by the Speech API.

- The Speech API never returns JSON fields with a value of `null`. Required fields always have non-`null` values.
- When an optional field is omitted, the key is left out of the object.
- Objects may have fields in addition to those defined in this specification. Clients must ignore any extra fields.
- The Speech API returns JSON according to this format when its response has a HTTP 200 status code. Responses for other status codes (for example, 400 or 500) can be in other formats that may be inappropriate for machine interpretation.

3.8.3 JSON Objects and Fields

The following sections describe the hierarchy of JSON objects and fields.

- Top Level Field
- Recognition Object Fields
- Hypothesis Object Fields

3.8.3.1 Top Level Field

The sole field at the top-level is as follows:

- `Recognition`: Object (required)
  The top-level `Recognition` object contains all of the useful data of a transcription. For information about the fields of this object, see Section 3.8.3.2 Recognition Object Fields

Note: Other fields may also appear at the top-level. Client applications should be prepared to ignore them.

3.8.3.2 Recognition Object Fields

The fields of a Recognition object are as follows.

- `ResponseId`: String (required)
  A unique string that identifies this particular transaction for follow-up queries or tracking issues. Developers must reference the ResponseId value in any problem reports. Some Speech API services may have features that respond to previous ResponseId values if they are supplied in a follow-up request.
- `Status`: String (required)
  The value of this field is a machine-readable hint for recognition status. The standard values are as follows:
  - “`OK`” implies that the `NBest` field contains non-empty hypotheses.
  - “`No Speech`” implies that the `NBest` field is empty because the speech was silent or unintelligible.
  - The following values are uncommon: “`Not Enough Speech`”`,` “`Too Much Speech`”`,` “`Too Quiet`”`,` “`Speech Too Soon`”
- `NBest`: Array of Hypothesis objects (required for successful recognition, omitted for recognition failure)
  This field is present whenever recognition is successful and it always includes at least one element. Each object represents a possible interpretation of the audio.

  Note: Most contexts in the Speech API return either zero or one entries in the `NBest` array.

3.8.3.3 Hypothesis Object Fields

The fields of a Hypothesis object are as follows:

- `Hypothesis`: String (required)
  This field contains the transcription of the audio. The `Hypothesis` field may be an empty string when speech is silent or unintelligible.
- `LanguageId`: String (optional)
  This field identifies the language used to decode the hypothesis. The `LanguageId` string is made up of the two-letter ISO 639 language code, a hyphen, and the two-letter ISO 3166 country code in lower case (for example, *en-us)*. The country code may not match the country of the user; for example, the system may use an *en-us* language model to recognize the speech of Canadian English speakers. Packages that include multilingual output or language detection must always supply this field. Packages for which language is irrelevant may omit this field or use this field to indicate the supported language.

- `Confidence`: Number (required)
  The confidence value of the hypothesis. The value ranges between 0.0 and 1.0 inclusive.
- `Grade`: String (required)
  A machine-readable string that indicates an assessment of recognition quality and the recommended treatment of the hypothesis. The grade reflects a confidence region based on prior experience with similar results. Applications may ignore this value to make their own independent assessment. The grade string contains one of the following values:
  - "`Accept`": The hypothesis value has acceptable confidence.
  - "`Confirm`": The user should confirm the hypothesis due to lower confidence.
  - "`Reject`": The hypothesis should be rejected due to low confidence.
- `ResultText`: String (required)
  A text string that is prepared according to the use cases of the speech context. The string is a reformatted version of the `Hypothesis` field, and the words may be altered through insertions, deletions, or substitutions to make the result more readable or usable for the user.

  Common alterations include conversion of a series of letters into acronyms and a series of digits into numerals. For example:

  Address:

  ```
  Hypothesis="twenty three hundred forrester lane cambridge mass"
  ResultText="2300 Forrester Ln, Cambridge, MA 02238"
  ```

  Text Message:

  ```
  Hypothesis="text john the game is on E S P N at eight PM"
  ResultText="text John, the game is on ESPN at 8pm"
  ```
- `Words`: Array of strings (required)
  Contains the words of the ResultText hypothesis split into separate strings. The Words array may omit some of the words of the ResultText string, and the array can be empty, but the Words array contains only words that are in the ResultText string. The WordScores array contains the confidence values for each of these words.
- `WordScores`: Array of numbers (required)
  Contains the confidence scores for each of the strings in the Words array. Each value ranges from 0.0 to 1.0 inclusive.

3.9 Speech Recognition Examples

As you begin using the Speech API recognition service, it's useful to try simple HTTP requests to understand the details of the system. For example you can:

- Verify that your developer credentials pass the authentication process.
- Send audio files to the service to check the proper encoding.
- View the JSON output and transcription text.
- Observe the HTTP response codes that arise from various client and server errors.

This section shows some examples of using the Speech API recognition service with cURL, an open source command-line tool.

3.9.1 Running cURL

cURL is included with Mac OS X and many Linux distributions. If your system does not have it already, you can download it from http://curl.haxx.se. Refer to that site also for full documentation on cURL.

The following table lists the cURL options used in the subsequent examples:

Option	Description
`--request POST`	Tells cURL to issue an HTTP POST.
`--data-binary "@filename"`	Identifies the audio file to be sent in the body of the POST. A quirk of cURL is that the name of the file is prefixed with an @ character.
`-H "header: value"` or `--header "header: value"`	Adds a header to the HTTP request. The Speech API requires the `Content-Type`, `Authorization`, and `X-SpeechContext` headers on speech requests.
`-v` or `--verbose`	Tells cURL to be verbose. It echoes the HTTP headers that it sends and receives, and it also prints debugging information about SSL connections. It is a very useful flag if you are having trouble getting the Speech API to return the expected data.
`-i` or `--include`	Tells cURL to echo the headers of the HTTP response.

Table 3-7: cURL options.

3.9.2 Example of an OAuth Request

Each Speech API request needs to include an `Authorization` header containing an OAuth client credentials access token. To test this functionality in your application, you can use cURL to request the token. The following example shows a cURL client credentials request and the JSON response with the access token. Before you run this example, set the shell variables `$APP_KEY` and `$SECRET_KEY` to the values obtained from the AT&T Developer Program web site for your application.

cURL OAuth Request

```
curl --include --data
"client_id=$APP_KEY&client_secret=$SECRET_KEY&
grant_type=client_credentials&scope=SPEECH"
"https://api.att.com/oauth/token"

HTTP/1.1 200 OK
Content-Length: 131
Content-Type: application/json

{
"access_token":"0123456789abdcdef0123456789abcdef",
"expires_in":"0",
"refresh_token":"0123456789abcdef0123456789abcdef01234567"
}
```

Example 3-5: cURL OAuth request.

3.9.3 Example of Successful Speech Recognition Response

A success response code of 200 from the Speech API recognition service indicates that the audio data was received and processed successfully. The HTTP body of the response contains a JSON or XML document with details on the transcription. Note that a response code of 200 does not guarantee that there is a useful transcription. For example, if the posted audio contains silence or unintelligible speech, then the service returns a 200 status and a response document with an empty transcription. Client apps must check both the status code and the contents of the response data.

To test Speech to Text with cURL, you need an audio file in Speex, AMR or WAV format and a valid OAuth access token. In the following example, `audiofile.amr` is the AMR audio to convert, and `$OAUTH_TOKEN` is a shell variable containing the OAuth access token.

Successful cURL Request

```
curl --include --request POST --data-binary "@audiofile.amr" --
header "Authorization: BEARER $OAUTH_TOKEN" --header "Content-
Type: audio/amr" --header "X-SpeechContext: VoiceMail"
"https://api.att.com/speech/v3/speechToText"

HTTP/1.1 200 OK
Content-Length: 428
Content-Type: application/json
```

```

{"Recognition":{"Status":"OK",
"ResponseId":"59c186b4e376962f6ac21d56f0bfcd8d",
"Info":{"metrics":{"audioTime":2.49000001,"audioBytes":39986}},
"NBest":[{
"Confidence":0.376718715,"Grade":"accept","LanguageId":"en-US",
"Hypothesis":"I'll be back around three PM",
"ResultText":"I will be back around _TIME 03:00 PM? _END_TIME",
"Words":["I","will","be","back","around","03:00 PM?"],
"WordScores":[0.389,0.389,0.389,0.379,0.389,0.32],
}]}}
```

Example 3-6: Successful cURL request.

3.9.4 Example of Unauthorized Credentials

The Speech API recognition service returns an HTTP 401 status code if the request does not have valid credentials: either the request lacked an `Authorization` header, or the token in the header is not valid. When a client app receives a response with a status code of 401, it should perform a new OAuth client credentials request to get a valid access token, and then it can re-issue the speech recognition request with the new credentials.

The following example illustrates a cURL request with credentials that are not valid. The important line in this example is line 3, which has a `401 Unauthorized` status response.

Unauthorized cURL Response

```
1    |  curl --include --request POST --data-binary "@audiofile.amr" --
        header "Authorization: BEARER $OAUTH_TOKEN" --header "Content-
        Type: audio/amr" --header "X-SpeechContext: VoiceMail"
        "https://api.att.com/speech/v3/speechToText"
2    |
3    |  HTTP/1.1 401 Unauthorized
4    |  Content-Length: 153
5    |  Content-Type: application/json
6    |  WWW-Authenticate: Bearer realm=api.att.com, error=invalid_token,
        error_description=the token is not valid
7    |
8    |  {"RequestError": {
9    |      "PolicyException": {
10   |          "MessageId": "POL0001",
11   |          "Text": "UnAuthorized Request"
12   |               "Variables": "accesstoken"
13   |      }
14   |  }}
```

Example 3-7: *Unauthorized cURL response.*

3.9.5 Example of Error Response

The Speech API returns HTTP response codes of 400 or greater when there are errors handling the request. The response body should be machine-readable JSON, though sometimes it will return HTML or plain text.

The Speech API recognition service returns a status code of 400 when the request was malformed. This can occur if the client app sends data in the wrong format or if one of the HTTP headers had a bad value. When an application receives a 400 response, it should interpret it as a bug in the client app rather than user error.

The service may return a status code of 500 or 503 if there is an error on the Speech API server. When an app receives such status code, it should let the user try the request again, because the server problem might be fixed over time.

The following example illustrates the use of cURL to send non-audio data to the Speech API. The key lines are the `400 Bad Request` status (line 3) and the JSON body with a `RequestError` element (line 7).

cURL Error Response

```
1    |  curl --include --request POST --data-binary "@file.txt" --header
        "Authorization: BEARER $OAUTH_TOKEN" --header "Content-Type:
        audio/amr" --header "X-SpeechContext: VoiceMail"
        "https://api.att.com/speech/v3/speechToText"
2    |
3    |  HTTP/1.1 400 Bad Request
4    |  Content-Length: 181
5    |  Content-Type: application/json
6    |
7    |  {"RequestError": {
8    |      "ServiceException": {
9    |          "MessageId": "SVC0001",
10   |         "Text": "stream format is not amr [no amr header and coding
        != amr]",
11   |                "Variables": ""
12   |     }
13   |  }}
```

Example 3-8: *cURL error response.*

4 Using Speech SDK for iOS

This section describes how to add speech recognition to iOS apps using version 2.0.1 of the Speech SDK for iOS.

- How the Speech SDK works on iOS
- Setting up your iOS Project
- Speech Recognition Tasks on iOS
- Testing and Deploying Speech Applications on iOS

iOS is the software platform for the Apple iPhone, iPod touch, and iPad. The Speech SDK supports applications developed with the iOS Software Development Kit and distributed by the iOS App Store. The iOS SDK is available for iOS Development Program members at the web site http://developer.apple.com, which provides the tools and APIs necessary to develop apps on the iOS platform using the Objective-C programming language. The Speech SDK supports applications that target iOS version 5.0 and later.

4.1 How the Speech SDK Works on iOS

The Speech SDK is packaged as a static library called ATTSpeechKit that links to your iOS application code. The library implements a client for the Speech API recognition web service. It exports the `ATTSpeechService` class, which provides the following functionality:

- Capturing audio from the device microphone.
- Showing feedback to the user.
- Streaming audio and custom data to the Speech API.
- Waiting for a response from the Speech API.

When your app wants to accept speech input from the user, it configures properties on an `ATTSpeechService` object and calls a method to start the interaction. The `ATTSpeechService` instance activates the device's microphone, accepts audio input from the user, and waits for the response from the server. When the recognition is complete, the `ATTSpeechService` class returns the result of recognition to your application.

Although the ATTSpeechKit library does not include classes that interface with the Text to Speech web service directly, the Speech SDK includes sample code that shows how to use standard iOS classes to access the Text to Speech service and play the audio it returns.

4.2 Speech SDK Prerequisites for iOS

To develop applications for iOS, you need a Mac OS X computer that runs the Xcode tools, and you should be familiar with programming iOS applications using Objective-C. To get started with the Speech SDK for iOS, prepare your development environment by completing the following steps.

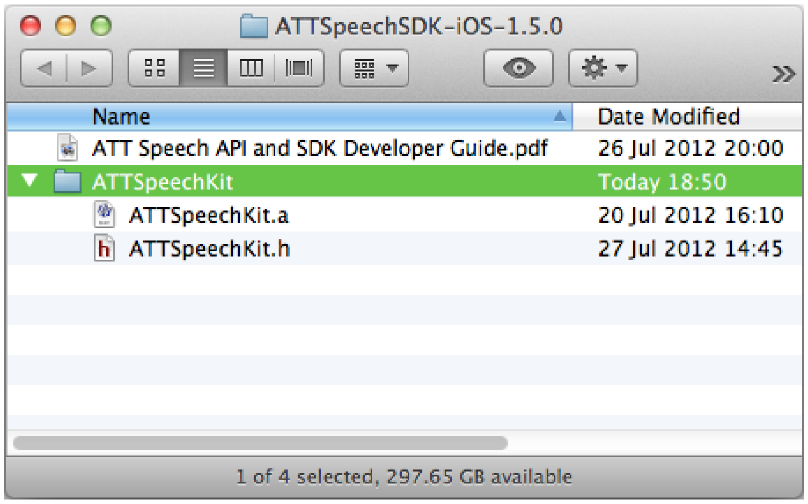
1. You must be a member of Apple's iOS Developer Program to run applications on a device and to distribute applications to customers.
2. Follow the instructions on the Apple Developer website to install Xcode and the iOS SDK.

4.3 Setting up your iOS Project

To use the Speech SDK in your application, you must add the ATTSpeechKit library to your Xcode project and link it to a set of system frameworks. The SimpleSpeech sample code on the AT&T Developer web site includes an Xcode project that is already configured to link with the ATTSpeechKit library. You can use that sample project as the basis for new apps.

To add the Speech SDK to an existing app, use the following procedure.

1. Open the Xcode project for your application.
2. Display the unzipped Speech SDK in the Finder and select the `ATTSpeechKit` subfolder (as shown in the following figure).



ATTSpeechSDK-iOS-1.5.0

Name	Date Modified
ATT Speech API and SDK Developer Guide.pdf	26 Jul 2012 20:00
ATTSpeechKit	Today 18:50
ATTSpeechKit.a	20 Jul 2012 16:10
ATTSpeechKit.h	27 Jul 2012 14:45

1 of 4 selected, 297.65 GB available

Figure 4-1: Selecting the ATTSpeechKit subfolder.

3. Drag the `ATTSpeechKit` folder from the Finder onto the project icon within your Xcode window.

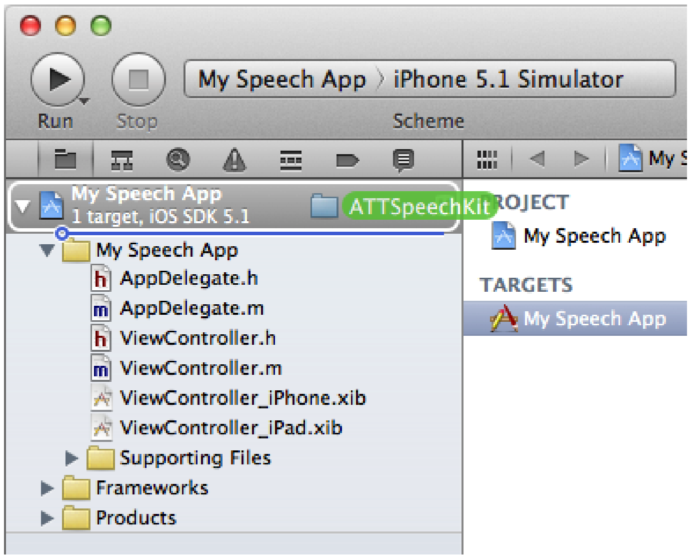
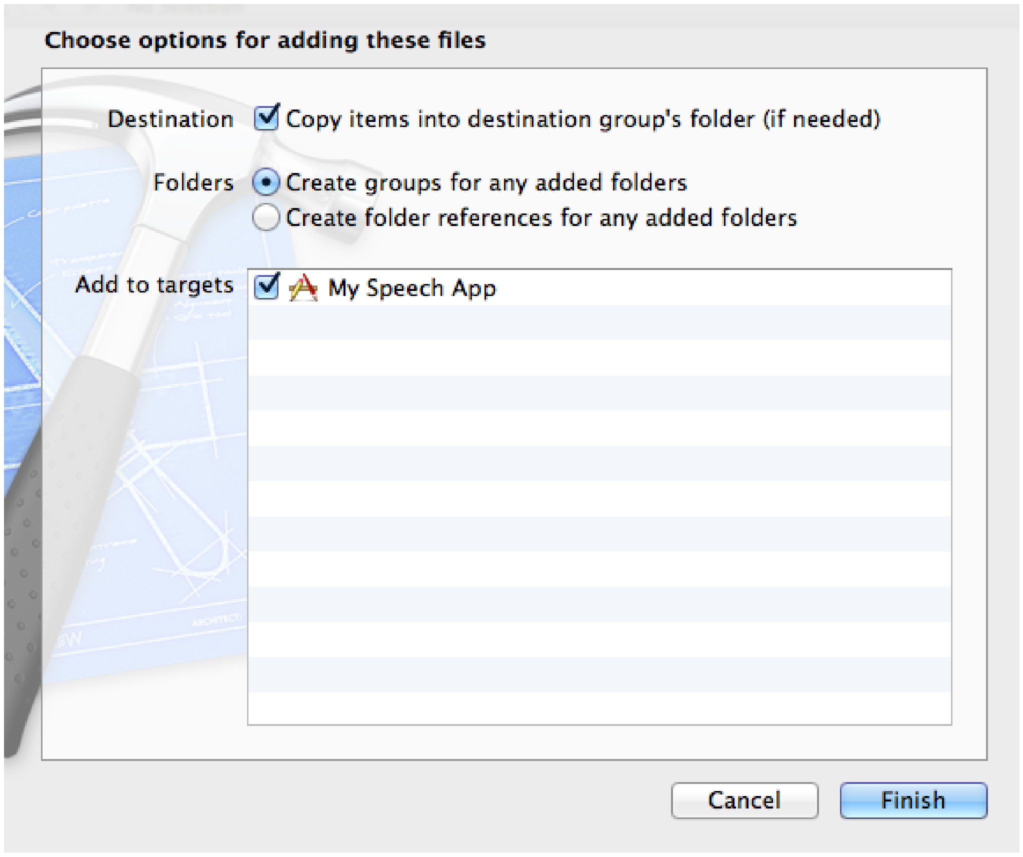


Figure 4-2: Dragging the ATTSpeechKit folder to your Xcode window.

4. In the dialog box that appears, select the following items, and then click the Finish button.



Choose options for adding these files

Destination [✓] Copy items into destination group's folder (if needed)

Folders [● Create groups for any added folders] ○ Create folder references for any added folders

Add to targets [✓] My Speech App

- **Destination:** Select the check box “Copy items into destination group’s folder (if needed)”.
- **Folders:** Select the radio button “Create groups for any added folders”.
- **Add to targets:** Ensure that your application target is selected in the “Add To Targets” list.

5. Link to the system frameworks via the following procedure.

   a. Display the Project Editor in Xcode by selecting your project in the Project Navigator.

   b. Select your target in the Project Editor.

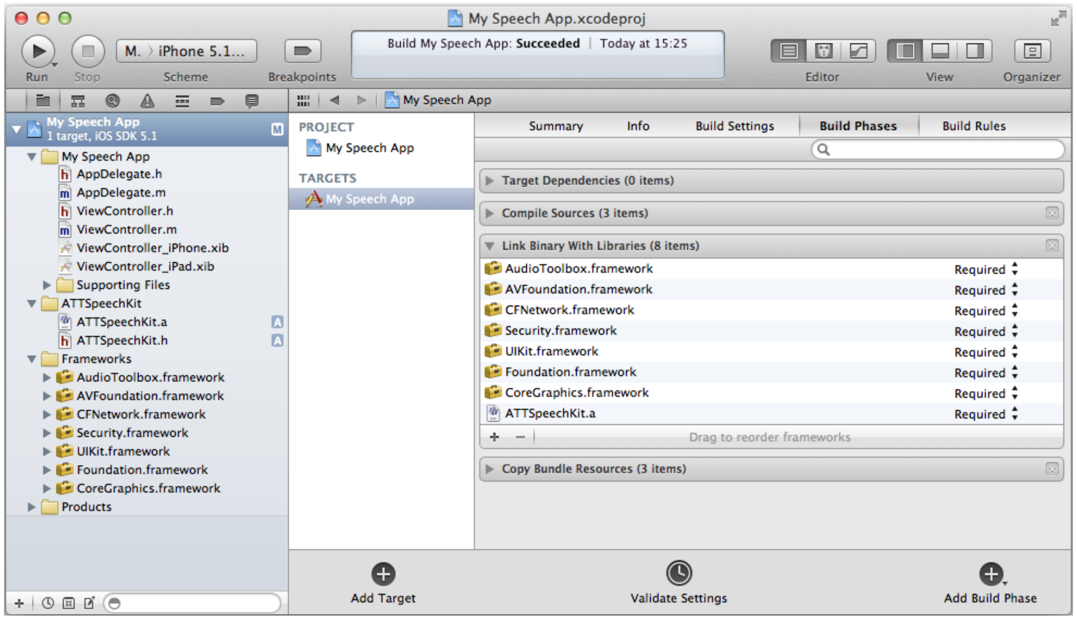
   c. In the Build Phases tab, expand the Link Binaries With Libraries section.

   d. Press the “+” button, and add the following system frameworks.

```
AudioToolbox.framework
AVFoundation.framework
CFNetwork.framework
Security.framework
```

e. (Optional) In the Project Navigator, move the newly added frameworks to the Frameworks group.

6. When complete, your Xcode target should look similar to the following.



```
My Speech App
1 target, iOS SDK 5.1
  My Speech App
    AppDelegate.h
    AppDelegate.m
    ViewController.h
    ViewController.m
    ViewController_iPhone.xib
    ViewController_iPad.xib
    Supporting Files
  ATTSpeechKit
    ATTSpeechKit.a
    ATTSpeechKit.h
  Frameworks
    AudioToolbox.framework
    AVFoundation.framework
    CFNetwork.framework
    Security.framework
    UIKit.framework
    Foundation.framework
    CoreGraphics.framework
  Products
```

```
Target Dependencies (0 items)
Compile Sources (3 items)
Link Binary With Libraries (8 items)
  AudioToolbox.framework      Required
  AVFoundation.framework      Required
  CFNetwork.framework         Required
  Security.framework          Required
  UIKit.framework             Required
  Foundation.framework        Required
  CoreGraphics.framework      Required
  ATTSpeechKit.a              Required
Copy Bundle Resources (3 items)
```

Figure 4-3. Completed Xcode target.

7. Follow the instructions in the following section to add code that invokes the `ATTSpeechService` class and receives delegate callbacks.

4.4 Speech Recognition Tasks on iOS

To add speech recognition to your application, you need to write code that performs the following actions.

- Configure `ATTSpeechService` runtime properties.
- Acquire an OAuth access token.
- Start speech interaction at the appropriate time (for example, in a button press handler)
- (Optional) Send inline grammar data.
- (Optional) Customize UI during speech interaction.
- (Optional) Customize speech endpointing.
- Handle speech recognition results and errors.

4.4.1 Configure ATTSpeechService Runtime Properties

The AT&T Speech SDK needs to be configured before an application uses it to call the Speech API. The best place to perform this configuration is in the application delegate’s `applicationDidBecomeActive:` method. Adding the configuration code to this method ensures that the Speech SDK is configured early during the application lifecycle, which minimizes latency during a speech interaction. It also ensures that the configuration occurs each time iOS brings the application to the foreground.

Your application will access the Speech SDK though the singleton `ATTSpeechService` object. This object manages the device’s microphone input and network connection to the Speech API on behalf of your application. It exposes a number of properties that you can use to specify the parameters of a speech recognition request. For the full list of `ATTSpeechService` request properties, see Section 5.2 ATTSpeechService Request Properties and Constants.

The `delegate` property is a mandatory property to set. It is an object supplied by your application that implements the `ATTSpeechServiceDelegate` protocol. For more information on the `ATTSpeechServiceDelegate` protocol, see Section 5.3 ATTSpeechService Delegate Callback Methods.

Another mandatory property is `recognitionURL`. This is the URL of the Speech API speech recognition service that your application uses. Get this URL from the on-boarding information for your application on the AT&T Developer web site.

A third mandatory property for the Speech API is `speechContext`. This property contains the name of the speech context that your application wants to use for speech recognition. For more information on the available contexts, see Section 2.1.2 Speech Contexts.

When your application configures the runtime properties of the Speech SDK, it can optionally call the `prepare` method. This tells `ATTSpeechService` to prepare the audio hardware of the device, which minimizes delays that can occur the first time the microphone is activated.

In addition to setting the Speech SDK properties, your application should also acquire an OAuth access token when the OS activates the application. For more details on acquiring the OAuth token, see Section 4.4.2 Acquire OAuth Access Token.

In the following example, code is added to the `applicationDidBecomeActive:` method to configure the Speech SDK and get an OAuth token.

```
applicationDidBecomeActive:
```

```
- (void) applicationDidBecomeActive: (UIApplication*) app
{
  ATTSpeechService* speechService = [ATTSpeechService
sharedSpeechService];
  speechService.delegate = self;
  speechService.recognitionURL = [NSURL URLWithString:
@"https://api.att.com/speech/v3/speechToText"]
  speechService.speechContext = @"Generic";
  [self myRenewSpeechOAuth];
  [speechService prepare];
}
```

Example 4-1: Configure `ATTSpeechService` in `applicationDidBecomeActive:`.

4.4.2 Acquire OAuth Access Token

Requests to the Speech API recognition service must include a valid OAuth access token. To avoid authentication failures during a speech interaction, your application should obtain the access token before the user tries to speak. Your application obtains an access token by making an OAuth request to the Speech API. The Speech SDK includes sample code that demonstrates how to obtain an OAuth access token. Your application can re-use the sample code or implement its own technique for obtaining the token.

When your code receives the access token, it can use the token in subsequent speech interactions by setting the `ATTSpeechService` `bearerAuthToken` property, as in the following lines:

```
ATTSpeechService* speechService = [ATTSpeechService sharedSpeechService];
speechService.bearerAuthToken = myAccessToken;
```

Since the network I/O to fetch the OAuth token can take a fraction of a second, it is important that the request be performed asynchronously or on a background thread. That prevents the UI of the application from locking up as it waits for a response over the network. Your application should also disable its speech UI briefly until it receives an OAuth access token.

It is best to request an access token every time your app comes to the foreground. This ensures that the subsequent speech request will have a valid token. The sample code in Example 4-1 includes the line `[self myRenewSpeechOAuth]`. You should replace that line with a call to the code in your application that begins the asynchronous token request. If the users of your application will keep it in the foreground longer than an hour, you need to add code to request a token at hourly intervals.

4.4.3 Start Speech Interaction

Once your application has configured the Speech SDK and obtained an OAuth access token, it can perform speech requests. In a typical application, the user can trigger speech input by pressing a button. The application code handling the button press calls the `startListening` method, and the `ATTSpeechService` object will begin a speech interaction. The Speech SDK will turn on the microphone and begin streaming data in compressed Speex format to the Speech API server. Your delegate object is called at various stages through the speech interaction until it receives the recognition result or an error.

In the following example, code is added to the application's action handler method to start a speech interaction.

ATTSpeechService

```
- (IBAction) listen: (id) sender
{
  ATTSpeechService* speechService = [ATTSpeechService
sharedSpeechService];
  [speechService startListening];
}
```

Example 4-2: Start speech interaction on iOS.

4.4.4 Send Inline Grammar Data

If your application is using a Speech to Text Custom context, you need to add the custom grammar data to the multipart document of the web service request. For more information on the multipart format, see Section 3.2.2 Inline Grammar Request Format.

There are two steps in adding grammar data to the request. First, your application must set the `isMultipart` and `contentType` properties when initializing the `ATTSpeechService` object as in the following lines. That tells the Speech SDK to send data in the proper multipart format.

```
speechService.isMultipart = YES;
speechService.contentType = @"multipart/x-srgs-audio";
```

Then your delegate object needs to implement the `speechServiceSendingPartsBeforeAudio:` callback method. `ATTSpeechService` calls this method as it is streaming data on the HTTP request to the web service. Your delegate implementation will insert the inline grammar data at this stage using the `addPart:contentType:disposition:` method. The following example shows adding both lexicon and grammar data to a request.

```
speechServiceSendingPartsBeforeAudio:
- (void) speechServiceSendingPartsBeforeAudio: (ATTSpeechService) speechService
{
  [speechService addPart: myLexiconData contentType: @"application/pls+xml" disposition: @"form-data; name=\"x-dictionary\""];
  [speechService addPart: myGrammarData contentType: @"application/srgs+xml" disposition: @"form-data; name=\"x-grammar\""];
}
```

Example 4-3: Send inline grammar on iOS.

4.4.5 Customize UI

An application should display a user interface that lets the user know that microphone input is being captured during a speech interaction. The Speech SDK can display a standard alert window while listening to the microphone, as shown in Figure 4-4. When listening is complete, the alert window changes to a progress display while the Speech SDK is waiting for a recognition result from the server. Your application can enable this UI by setting the `showUI` property of `ATTSpeechService` to `YES`. It can customize the labels in the alert through several `ATTSpeechService` properties. For more information on the UI properties, see the `showUI` property in Section 5.2 ATTSpeechService Request Properties and Constants.

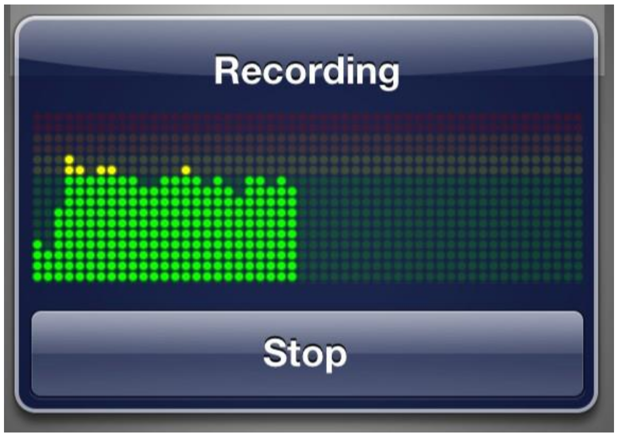


Figure 4-4: Speech SDK standard UI on iOS.

If your application displays its own UI during a speech interaction, it can register delegate callback methods to receive updates during the recording and processing. Implement the delegate method `speechService:audioLevel:` to receive an average input volume level roughly every 1/10 second. Other delegate methods receive notifications when listening starts and ends. For more information on delegate callbacks, see Section 5.3 ATTSpeechService Delegate Callback Methods.

The Speech SDK automatically ends recording when the user stops talking. However, your application can manually stop capturing audio by calling the `stopListening` method. The Speech SDK will wait for the Speech API services to perform recognition on the speech that was captured up to that point.

Your app can call the `cancel` method to abort the speech interaction. Note that your delegate will not get a callback from the service if `cancel` is called.

4.4.6 Speech Endpointing

Once the user has begun a speech interaction, the Speech SDK needs to determine when to stop listening, a process called *endpointing*. The Speech SDK supports the following techniques for endpointing on iOS:

- **Silence Detection:** When the user is quiet after doing some speaking, the Speech SDK can treat the silence as a signal to stop listening. An application can tune this value by setting the `endingSilence` property.
- **Timeout Expiration:** An application using the Speech SDK can set the maximum duration of speech input through the `maxRecordingTime` property.
- **Manual Endpointing:** The user may press a button in the app's user interface to signal that they are done talking. The standard UI presented by the Speech SDK displays a "Stop" button for the user to perform manual endpointing. An application that presents its own UI while listening can call the `stopListening` method to do the same thing.

The Speech SDK handles all three techniques during speech interactions. Timeouts and manual endpointing serve to override silence detection. For example, if an application sets the `maxRecordingTime` property to 5 seconds and the user is still talking at the 5 second mark, then the Speech SDK will stop listening, even though the user was not silent.

In addition to using silence to detect that the user has finished talking, the Speech SDK uses silence to determine whether the user spoke at all during an interaction. If the user has not spoken after `maxInitialSilence` seconds, then the Speech SDK can cancel the speech interaction. An application customizes this behavior through the `cancelWhenSilent` property.

4.4.7 Handle Speech Recognition Results and Errors

When speech recognition is complete, the Speech SDK calls the delegate's `speechServiceSucceeded:` method. The `ATTSpeechService` argument exposes properties for the application to get the recognition response data. An array of transcription hypothesis strings is available in the `responseStrings` property. The parsed JSON data from the service is in the `responseDictionary` property, and the raw binary data is in the `responseData` property. For the full list of response properties, see Section 5.4 ATTSpeechService Response Properties.

The following example shows a minimal implementation of the `speechServiceSucceeded:` method that checks the hypothesis array.

`speechServiceSucceeded:`

```
-(void) speechServiceSucceeded: (ATTSpeechService*) speechService
{
  [NSArray* nbest = speechService.responseStrings;
  NSString* recognizedText = @"";
  if (nbest != nil && nbest.count > 0)
    recognizedText = [nbest objectAtIndex: 0];
  if (recognizedText.length)
    NSLog(@"Recognition result: %@", recognizedText);
  else
    NSLog(@"Speech was not recognized.");
}
```

Example 4-4: Successful Transaction delegate method.

If an error occurs doing the speech recognition process, the Speech SDK calls the delegate's `speechService:failedWithError:` method. Possible errors include the user canceling, the user not speaking in time, problems with the network connection, and errors reported by the server. The `NSError` argument includes properties that describe the type of error. For more information on interpreting error callbacks, see Section 5.5 ATTSpeechService Error Codes.

The following example shows a minimal implementation of the `speechService:failedWithError:` method that checks the error status.

`speechService:failedWithError:`

```
-(void) speechService: (SpeechService*) speechService
failedWithError: (NSError*) error
{
  if ([error.domain isEqualToString: ATTSpeechServiceErrorDomain]
&& (error.code == ATTSpeechServiceErrorCodeCanceledByUser))
    NSLog(@"User canceled.");
  else
    NSLog(@"Recognition failed with error: %@", [error
localizedDescription]);
}
```

Example 4-5: Failed transaction delegate method.

4.5 Testing Speech Applications on iOS

The Speech SDK implements audio capture and network I/O as asynchronous activities. However, iOS does not offer easy-to-use testing frameworks that support the event loops required for asynchronous operations. As a consequence, it is difficult to use automated unit testing tools to test speech functionality.

The best procedure for application developers is to verify Speech API functions (like such as valid credentials and connectivity) outside of their iOS application by using the cURL command-line tool. For more information on the testing with cURL, see Section 3.9 Speech Recognition Examples.

4.6 Deploying Speech Applications on iOS

Developers who create speech-enabled applications on iOS devices using the Speech SDK must link their applications to the ATTSpeechKit static library. There is nothing additional that must be added to distribute the application to users.

5 Speech SDK Reference for iOS

This section is a reference for the Speech SDK on iOS. It describes the methods, properties, callbacks, error codes, and state transitions in the `ATTSpeechService` class.

- ATTSpeechService Methods
- ATTSpeechService Request Properties
- ATTSpeechService Delegate Callbacks
- ATTSpeechService Response Properties
- ATTSpeechService Error Codes
- ATTSpeechService State Transitions

5.1 ATTSpeechService Methods

The following table describes the methods that `ATTSpeechService` implements.

Method	Required	Description
`+(ATTSpeechService)` **`sharedSpeechService`**	Yes	This class method returns the singleton speech service object.
`-(void)` **`prepare`**	No	Initializes the iOS audio subsystem. Calling this method is optional, but if it is called early during the application lifecycle, it can reduce the latency of the first speech interaction requested by the user.
`-(BOOL)` **`startListening`**	Yes	Starts a speech interaction using the microphone. Before calling this method, set the request properties on the `ATTSpeechService` object to configure the interaction. After this method is called, the Speech SDK invokes will invoke methods on the delegate object to report interaction status, errors, and the recognition result.

Method	Required	Description
`-(BOOL)` **`startWithAudioData:`** `(NSData*) audioData`	Yes	Sends audio data to the Speech to Text API recognition service for processing instead of using the microphone. Before calling this method, set the request properties on the `ATTSpeechService` object to configure the interaction. The method does not compress the audio data, so it should already be in a format supported by the Speech to Text service. After this method is called, the `ATTSpeechService` instance invokes will invoke methods on the delegate object to report interaction status, errors, and the recognition result.
`-(void)` **`stopListening`**	No	Manually ends speech input. This method is optional because the ATTSpeechKit library will automatically end speech input when the user stops talking or a timeout expires. After this method is called, the `ATTSpeechService` object will wait for the server to process the submitted speech, and then it will report the result or error to the delegate object.
`-(void)` **`extendTimeout:`** `(NSTimeInterval) extraInterval`	No	Keep acquiring data for the specified number of seconds. This method is optional in unimodal speech applications. Call this in multimodal applications to prevent `ATTSpeechService` object from ending audio input while the user is supplying input through other means.

Method	Required	Description
`-(void) addPart: (NSData*) partData contentType: (NSString*) contentType disposition: (NSString*) contentDisposition`	No	When the Speech SDK is posting multipart data to the Speech to Text Custom service, the application can call this method to add additional parts to the posted data. It may only be called when the delegate is handling a `speechServiceSendingPartsBeforeAudio:` or `speechServiceSendingPartsAfterAudio:` callback. It is an error to call this method in any other callbacks or outside of a speech interaction.
`-(void) cancel`	No	Cancels a speech interaction. After this method is called, the `ATTSpeechService` object will not attempt to perform speech recognition on the submitted data, and it will not make further delegate callbacks related to the canceled interaction.
`+(void) cleanupForTermination`	No	Call this class method when you are done with all speech interactions to release all resources used by the Speech SDK. This method is intended for non-production builds of applications when investigating memory issues.

Table 5-1: ATTSpeechService methods.

5.2 ATTSpeechService Request Properties and Constants

The following tables consists of the `ATTSpeechService` properties that control speech recognition. Set these properties before calling the `startListening` or `startWithAudioData:` methods.

Property	Default Value	Type	Required	Description
`delegate`		`ATTSpeechServiceDelegate`	Yes	An application-defined object that receives callbacks during speech recognition. For more information, see Section 5.3 ATTSpeechService Delegate Callback Methods.
`currentState`	`ATTSpeechServiceStateIdle`	`ATTSpeechServiceState`	n/a	Returns the current state of a speech interaction.
`recognitionURL`		`NSURL`	Yes	The URL of the Speech API service. For more information on this URL, see Section 3.1.1 Speech Recognition Service URL.
`isMultipart`	`NO`	`BOOL`	No	Specifies whether to wrap the audio data in a MIME multipart container. When `YES`, Speech SDK posts will post data in multipart format, and the delegate must implement the `speechServiceSendingPartsBeforeAudio:` or `speechServiceSendingPartsAfterAudio:` callbacks.
`contentType`	See description.	`NSString`		MIME type of the audio data for the server. It is used in the `Content-Type` HTTP header. This property is needed only when uploading an audio file. When using the microphone, `ATTSpeechService` uses the MIME type for the chosen audio format.
`speechContext`		`NSString`	Yes	The name of the Speech to Text recognition context to use for recognition. It is used in the `X-SpeechContext` HTTP header.

Property	Default Value	Type	Required	Description
`xArgs`	`nil`	`NSDictionary` mapping `NSString` to `NSString`	No	A collection of key-value argument pairs for a Speech API context. It is used in the `X-Arg` HTTP header. This property will be merged with a header specified in `requestHeaders` and the default values generated by Speech SDK.
`sendsDefaultXArgs`	`YES`	`BOOL`	No	Leaving this property `YES` enables the Speech SDK to automatically generate `X-Arg` values based on the device and application properties: `DeviceType`, `DeviceOS`, `DeviceTime`, `ClientSDK`, `CientApp`, and `ClientVersion`. When set to `NO`, the Speech SDK will not add the `X-Arg` values.
`audioFormat`	`ATTSKAudioFormatSpeex_WB`	`NSString`	No	Name of format in which to encode audio from the microphone. Defaults to Speex wideband format. Use only the constants defined in Table 5-3.
`bearerAuthToken`		`NSString`	Yes	An OAuth access token that validates speech requests from this application. It is used in the `Authentication: Bearer` HTTP header. For more information on OAuth access tokens, see Section 4.4.2 Acquire OAuth Access Token.
`requestHeaders`	`nil`	`NSDictionary` mapping `NSString` to `NSString`	No	A collection of HTTP headers to add to the request.
`httpMethod`	`nil`	`NSString`	No	The HTTP method to use for sending requests. Set to `nil` to use default method, which is "POST".

Property	Default Value	Type	Required	Description
`showUI`	`NO`	`BOOL`	No	Setting this property to `YES` enables the Speech SDK to display a progress meter and a button for canceling the interaction.
`recordingPromptMessage`	`"Listening"`	`NSString`	No	The text that the Speech SDK displays while capturing audio data from the microphone. Use this property only when the `showUI` property is set to `YES`.
`recordingStopButtonTitle`	`"Stop"`	`NSString`	No	The button label that the Speech SDK displays while capturing audio data from the microphone. The user can press the button to manually endpoint audio input. Use this property only when the `showUI` property is set to `YES`.
`processingPromptMessage`	`"Processing"`	`NSString`	No	The text that the Speech SDK displays while waiting for the server to return a recognition result. Use this property only when the `showUI` property is set to `YES`.
`processingCancelButtonTitle`	`"Cancel"`	`NSString`	No	The button label that the Speech SDK displays while waiting for the server to return a recognition result. The user can press the button to cancel the speech interaction. Use this property only when the `showUI` property is set to `YES`.
`connectionTimeout`	`10.0`	`float`	No	The maximum number of seconds that the Speech SDK waits for a connection to the server when initiating a recognition request.

Property	Default Value	Type	Required	Description
cancelWhenSilent	YES	BOOL	No	Controls how the Speech SDK handles silent audio input. When the value is NO, the Speech SDK sends will send silent audio to the server for processing. When the value is YES, the Speech SDK will automatically returns the an following error code of ATTSpeechServiceErrorCodeNoAudio when it detects silent audio.: ATTSpeechServiceErrorCodeNoAudio
maxInitialSilence	1.5	float	No	The maximum number of seconds that a user can remain silent before beginning to talk. To disable the timeout, set the property to the following constant: ATTSpeechServiceInfiniteInterval.
endingSilence	0.85	float	No	The number of seconds of silence after the user stops talking before the Speech SDK performs endpointing. To disable automatic endpointing, set the property to the following constant: ATTSpeechServiceInfiniteInterval.
minRecordingTime	1.0	float	No	The minimum number of seconds of audio to capture from the microphone. This limit is mainly useful for manual endpointing via stopListening. If the user does not meet this limit, Speech SDK returns will return an error code of ATTSpeechServiceErrorCodeAudioTooShort. To disable this limit, set the property to zero.

Property	Default Value	Type	Required	Description
`maxRecordingTime`	`25.0`	`float`	No	The maximum number of seconds of audio to capture from the microphone. When this limit is exceeded, Speech SDK stops listening to the microphone and waits for the Speech API to process the captured audio. To make this duration unlimited, set the property to the following constant: `ATTSpeechServiceInfiniteInterval`.
`serverResponseTimeout`	`30.0`	`float`	No	The maximum number of seconds that the Speech SDK waits for the server to complete a recognition response.

Table 5-2: ATTSpeechService request properties.

Some of the values for `ATTSpeechService` properties are special constants declared by the ATTSpeechKit library. The following table describes those constants.

Constant	Type	Description
`ATTSKAudioFormatAMR_NB`	`NSString`	Value for `ATTSpeechService.audioFormat` to specify AMR narrowband (8kHz sampling) format, which has a bitrate of 12.2kbps.
`ATTSKAudioFormatSpeex_WB`	`NSString`	Value for `ATTSpeechService.audioFormat` to specify Speex wideband (16kHz sampling) format, which has a variable bitrate averaging around 14kbps.
`ATTSKAudioFormatWAV_NB`	`NSString`	Value for `ATTSpeechService.audioFormat` to specify uncompressed streaming WAV narrowband (8kHz sampling) format, which has a bitrate of 128kbps.

Constant	Type	Description
`ATTSKAudioFormatWAV_WB`	`NSString`	Value for `ATTSpeechService.audioFormat` to specify uncompressed streaming WAV wideband (16kHz sampling) format, which has a bitrate of 256kbps.
`ATTSpeechServiceInfiniteInterval`	`float`	Represents an infinite duration of time for properties such as `ATTSpeechService.maxRecordingTime`.

Table 5-3: ATTSpeechService request property constants.

5.3 ATTSpeechService Delegate Callback Methods

While the `ATTSpeechService` object is performing a speech recognition interaction, it calls methods on its delegate object. The application is required to implement the two delegate methods that communicate the recognition result to the application. The remaining delegate methods are optional, giving the application a chance to customize the behavior of the speech interaction.

The following table describes the Speech SDK delegate callback methods.

Callback Method	Return	Required	Description
`speechServiceSucceeded:`	`void`	Yes	The Speech SDK calls this method when it returns a recognition result. The `ATTSpeechService` object argument will contain properties that include the response data and recognized text. For more information on how to interpret the response data, see Section 5.4 ATTSpeechService Response Properties.
`speechService: failedWithError:`	`void`	Yes	The Speech SDK calls this method when speech recognition fails. The reasons for failure can include the user canceling, network errors, or server errors. For more information on the `NSError` argument, see Section 5.5 ATTSpeechService Error Codes.
`speechServiceWillStartListening:`	`void`	No	The Speech SDK calls this method immediately before capturing audio to give the application a chance to perform a task. For example, the application can display a custom recording UI.
`speechServiceIsListening:`	`void`	No	The Speech SDK calls this method when it begins has begun to capture audio from the microphone.

Callback Method	Return	Required	Description
`speechServiceShouldEndRecording:`	`BOOL`	No	The Speech SDK calls this method when it is ready to end audio capture because the user is silent. The delegate method should return `NO` if listening should continue.
`speechServiceHasStoppedListening:`	`void`	No	The Speech SDK calls this method when it has finished capturing audio from the microphone.
`speechService: audioLevel:`	`void`	No	The Speech SDK calls this method repeatedly as it captures audio. The callback rate is roughly 1/10 second. An application can use the audio level data to update its UI.
`speechService: willEnterState:`	`void`	No	The Speech SDK calls this method when it transitions among states in a recording interaction, for example, from capturing to processing. The method argument contains the state the service will enter, and `speechService.currentState` contains the state the service is leaving. Upon exiting this delegate callback, the service will be in the new state.

Callback Method	Return	Required	Description
`speechServiceSendingPartsBeforeAudio`	`void`	No	When Speech SDK is posting data in multipart format, it calls this delegate method so the application can add parts to the request. The application should call `addPart:contentType:disposition:` to add a part. Parts added during the handling of this delegate callback will be received by the speech recognition service before the audio from the microphone, so this is the appropriate place to add inline grammars and hints to a request.
`speechServiceSendingPartsAfterAudio`	`void`	No	When Speech SDK is posting data in multipart format, it calls this delegate method so the application can add parts to the request. The application should call `addPart:contentType:disposition:` to add a part. Parts added during the handling of this delegate callback will be received by the speech recognition service after the audio from the microphone.

Table 5-4: ATTSpeechService delegate callback methods.

5.4 ATTSpeechService Response Properties

When a speech interaction is complete, the `ATTSpeechService` object exposes data about the response in its object properties. These response properties are valid during the execution of the `speechServiceSucceeded:` callback.

Property	Type	Description
`responseStrings`	`NSArray` of `NSString`	Contains the recognition response as an array of hypothesis strings. It is roughly equivalent to the JSON path `Rrecognition.nbestNBest[*].Hhypothesis`. Each string in the array is a possible way of interpreting the submitted speech. When the speech cannot be recognized, this property will be `nil`or have zero elements.
`responseDictionary`	`NSDictionary`	Contains the parsed JSON tree of the speech recognition response. For more information on how to interpret the data, see Section 3.7 Speech Recognition Response Formats. This property is `nil` if the speech service does did not return JSON.
`responseData`	`NSData`	Contains the raw bytes of the speech recognition response.
`statusCode`	`int`	Contains the HTTP status code of the speech recognition response.
`responseHeaders`	`NSDictionary` mapping `NSString` to `NSString`	Contains the HTTP headers that were included in the speech recognition response. Headers whose keys appear multiple times in a response (such as `Set-Cookie`) have their values concatenated together.

Table 5-5: ATTSpeechService response properties

5.5 ATTSpeechService Error Codes

When the Speech SDK calls the delegate's `speechService:failedWithError:` callback, the application can determine what happened by examining the properties of the `NSError` argument. For more information on the callback, see Section 5.3 ATTSpeechService Delegate Callback Methods.

The Speech SDK reports errors from a variety of sources, such as network errors, HTTP error results, and errors reported by the operating system. It follows the standard practices for `NSError` objects, so that each source of error corresponds to a value of the `NSError domain` property, and each distinct error from a source corresponds to values of the `NSError code` property. The following table describes the error domains and codes reported by the Speech SDK.

NSError Domain	NSError Code	Description
`ATTSpeechServiceErrorDomain`	`ATTSpeechServiceErrorCodeCanceledByUser`	The user canceled the speech interaction. This occurs, for instance, when the user presses the "Cancel" button or puts the application in the background.
`ATTSpeechServiceErrorDomain`	`ATTSpeechServiceErrorCodeAttemptAtReentrancy`	The application called `[the ATTSpeechService startListening]` or `[ATTSpeechService startWithAudioData:]` method while a speech interaction was already in process.
`ATTSpeechServiceErrorDomain`	`ATTSpeechServiceErrorCodeInvalidParameter`	One of the properties on the `ATTSpeechService`object was not valid.
`ATTSpeechServiceErrorDomain`	`ATTSpeechServiceErrorCodeInvalidURL`	The URL that was passed to the `recognitionURL` property is not valid.
`ATTSpeechServiceErrorDomain`	`ATTSpeechServiceErrorCodeNoDelegate`	The application did not set the delegate of the `ATTSpeechService` object.
`ATTSpeechServiceErrorDomain`	`ATTSpeechServiceErrorCodeConnectionFailure`	The Speech SDK was unable to make a network connection to the Speech API service.

NSError Domain	NSError Code	Description
`ATTSpeechService ErrorDomain`	`ATTSpeechServiceErrorCode NoResponseFromServer`	The server did not send a full response because it timed out or the network connection was dropped.
`ATTSpeechService ErrorDomain`	`ATTSpeechServiceErrorCode AudioTooShort`	`[ATTSpeechService stopListening]` was called before enough audio input was collected. Returned only when the `minRecordingTime` property is nonzero.
`ATTSpeechService ErrorDomain`	`ATTSpeechServiceErrorCode NoAudio`	The user did not speak loud enough for the speech to be recognized. Returned only when the `cancelWhenSilent` property is true.
`ATTSpeechService ErrorDomain`	`ATTSpeechServiceErrorCode NoMicrophone`	The device does not have a microphone.
`ATTSpeechService HTTPErrorDomain`	HTTP error code, such as 400, 404, and 500.	The server returned an HTTP error code.
Other domain values	n/a	There was an error propagated from the operating system.

Table 5-6: ATTSpeechService error codes.

5.6 ATTSpeechService State Transitions

An application can test the current stage of a speech interaction by accessing the `currentState` property of the `ATTSpeechService` instance. The delegate can also be notified automatically about the transitions among the stages by implementing the `speechService:willEnterState:` method.

The following table describes the `ATTSpeechService` states that occur during speech interaction.

State	Description
`ATTSpeechServiceStateIdle`	Nothing occurring. `ATTSpeechService`is in this state before beginning a speech interaction and after delegate callbacks have returned.
`ATTSpeechServiceStateConnecting`	Attempting to connect to the host before audio capture begins.
`ATTSpeechServiceStateRecording`	Audio capture is taking place.
`ATTSpeechServiceStateSendingAudioData`	Audio data are being sent to the server.
`ATTSpeechServiceStateProcessing`	The server is processing audio data.
`ATTSpeechServiceStateError`	`ATTSpeechService` is handling an error condition.

Table 5-7: ATTSpeechService state transitions.