AT&T™ Speech SDK Developer Guide for Android

Publication Date: July 1 2013

Legal Disclaimer

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**You**" and "**Your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to You because AT&T believes the Information may be useful to You. The Information is provided to You solely on the basis that You will be responsible for making Your own assessments of the Information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from Your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

Table of Contents

Contents

Table of Contents

Table of Contents

Table of Figures

Table of Tables

Table of Examples

1 Introduction

This guide is for software developers on the Android platform who intend to build applications using the AT&T Speech API and Speech SDK. It explains the configuration, development, and testing of applications that use the AT&T Speech SDK for Android. This guide should provide developers with a thorough understanding of all AT&T Speech API and Speech SDK data formats, operations, and parameters.

1.1 Accessing 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 web site at the following location:
https://developer.att.com/developer/forward.jsp?passedItemId=12500023

1.2 Related Documentation

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

- ***AT&T Speech API Technical Documentation:*** Contains detailed information on the REST interface of the AT&T Speech API, including request parameters and the format of the response data. Find a link to the technical documentation at the following location:
https://developer.att.com/developer/forward.jsp?passedItemId=12500023
- ***AT&T Speech SDK Release Notes:*** The distributions of the AT&T Speech SDK include a text file with release notes, describing the changes in that release of the SDK.

1.3 Acronym and Terms

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

Acronym	Term
AMR	Adaptive Multi-Rate audio format
API	Application Programming Interface
APK	Application Package
JSON	JavaScript Object Notation

Acronym	Term
MIME	Multipurpose Internet Mail Extensions
OS	Operating System
REST	Representational State Transfer
SDK	Software Development Kit
SMS	Short Message Service
SSL	Secure Sockets Layer
UI	User Interface
WAV	Waveform audio file format

Table 1-1: Acronyms and Terms

2 Overview of the AT&T 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 Overview of the AT&T Speech API

The AT&T Speech API services perform speech recognition and generation for third-party applications using a client-server REST architecture. The services support HTTP 1.1 clients and are not tied to any wireless carrier. The Speech API includes three web services:

- **Speech to Text:** This service 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:** This service is 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:** This service 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 web services. The phrase “speech recognition service” refers to the features that both web services have in common.

2.1.1 REST Web Service

The AT&T speech recognition services convert speech to text in a REST 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 performs 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 application might want the user to speak short keyword phrases like “find restaurants near me”, whereas a dictation program might want the user to speak full natural sentences. The AT&T Speech API recognition services supports these various scenarios with a set of speech contexts. Each speech context is tuned for a different scenario of user vocalization. The Speech to Text contexts include the following:

- **Voice Mail (`VoiceMail`):** This context transcribes long-form speech, such as voice mail messages. The audio comes from the client application; the service does not provide access to voice mail systems.
- **Business Search (`BusinessSearch`):** This context 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 application to parse the transcribed text and perform actual searches for the user.
- **Question and Answer (`QuestionAndAnswer`)**: This context 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 application to parse the transcribed text and perform actual queries for the user.
- **Web Search (`WebSearch`):** This context 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 application to parse the transcribed text and perform actual searches for the user.
- **Short Message Service (`Sms`):** This context 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 application to submit the text to any external messaging systems.
- **Generic (`Generic`):** This context recognizes a broad range of full sentences.
- **Television (TV):** This context 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 Text to Speech Custom service also allows applications to use custom recognition grammars. This is geared towards use cases with a specialized vocabulary. The Speech to Text Custom contexts include the following:

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

Note: The Speech SDK for Android does not currently support the inline grammar format used by the Text to Speech Custom service.

2.1.3 AT&T Developer Program On-Boarding

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

Once you have an account in the program, you will register your application on the AT&T Developer Program web site for access to Speech API services. For each application 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.

Each account in the Speech API is subject to traffic thresholds. Apps that exceed those thresholds may see subsequent requests throttled or shut down. You can contact the AT&T Developer Program to make arrangements for high-traffic applications.

It is very important that you keep the values for your app key and secret private. The AT&T Speech API assumes that all requests with your OAuth credentials are authorized by you. If your app key and secret are revealed, other parties could make Speech API requests that count against your thresholds, which could in turn result in the web service throttling or cutting off your application's Speech API traffic. If that occurs, you should be prepared to issue new versions of your application with new credentials.

2.1.4 Security

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

2.1.5 OAuth Client Credentials

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. Visit http://oauth.net/ for more information on the OAuth standard.

The Speech API uses the OAuth 2.0 Client Credentials flow for authentication. This is a two-party authentication in which the client application 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 application. The access token has a lifetime of two hours, after which a new access token must be acquired.
3. The client application 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 app and the service, not a three-party flow involving the user. There is no need for the end user to log in for an application to use the Speech API.

2.2 Overview of the AT&T Speech SDK

The AT&T Speech API, as a REST web service, works with any internet-connected device. It is straightforward for developers to send audio data to the Speech API recognition service 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 REST web service interface and make direct HTTP connections to the web services of the Speech API.

2.2.1 Platforms

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

For each supported platform, the Speech SDK is distributed as a ZIP archive that contains a code 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's device and generates data in AMR 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 end-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 that an application 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 user interface also provides a way for the end-user to cancel the request. Displaying the standard interface is optional; applications can implement a custom UI if desired.

3 Speech Recognition Web Service

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

3.1 Speech Recognition Web Service Requests

The Speech API recognition service 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 developers' credentials, 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 service requests connect to a URL which varies based on the actual service that the application has been on-boarded to. The URL for the speech recognition service has the template:

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

The fields of the URL are as follows:

- ***<hostname>:*** The address of the Speech API server. The production host name is `api.att.com`. Applications on-boarded to other versions of the service use different host names.
- ***<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 API recognition service use a common set of HTTP headers. The following table describes the HTTP headers that a client application can add to the request.

Header Name	Required/ Optional	Description
`X-SpeechContext`	Required	This header contains the name of the speech context, such as VoiceMail or BusinessSearch. Refer to the Speech API technical documentation for a full list of values supported in this header.
`Authorization: Bearer`	Required	This header contains the client's OAuth access token that authenticates this request. For more information about generating this header, see Section 3.5 Using OAuth Credentials.
`Content-Type`	Required	This header is the MIME type of the data that the client application sends in the POST request. For information on the supported content types, see Section 3.7 Speech Recognition Response Formats.
`Transfer-Encoding: chunked` or `Content-Length`	Required	These headers indicate a streaming or non-streaming request. For more information, see Section 3.3 Request Streaming.
`Accept`	Optional	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`	Optional	This header (which can appear multiple times) contains a set of extra parameters for the web service. For more information on the supported parameters, see Section 3.4 Extra Arguments.

Table 3-1: Request headers.

The following example illustrates the HTTP headers of the request.

HTTP headers of a request

```
1   |   X-SpeechContext: BusinessSearch
2   |   Authorization: Bearer ZmFrZWVudHJ5
3   |   Content-Type: audio/amr
4   |   Transfer-Encoding: chunked
```

Example 3-1: HTTP headers of a request

3.2 Speech Recognition Request Formats

The Speech API recognition service accepts 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 also submit grammar and 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 refers to the multipart format as the *inline grammars* format. Notice that the Speech SDK for Android does not currently support the inline grammars format.

3.2.1 Audio Request Format

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

- AMR narrowband format, 12.2 kbit/sec, 8 kHz sampling, MIME type `audio/amr`.
- WAV format, 16 bit PCM data, single channel, 8 kHz sampling, 128 kbit/sec, MIME type `audio/wav`. It sends a special streaming version of WAV in which the RIFF header does not 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 `multipart/form-data` format. This section describes the key features of the inline grammars format for Speech SDK users. Refer to the AT&T Speech API technical documentation for a full description of this format, and to the standard RFC 2388 at http://tools.ietf.org/html/rfc2388 for details on the underlying `multipart/form-data` format.

The multipart data format consists of several sections (or parts), and each part contains a part body and a collection of part headers. There are two part headers required of 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/ Optional	Content Disposition	Content Type	Description
Dictionary	Optional	`form-data; name="x-dictionary"`	`application/pls+xml`	Pronunciation lexicon of words in grammar. Data are in standard PLS format
Grammar	Required	`form-data; name="x-grammar"`	`application/srgs+xml`	Context-free grammar describing the recognizable phrases. Data are in standard SRGS format.
Speech	Required	`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, please refer to the Speech API technical documentation. Note that the Speech SDK for Android does not currently support the inline grammar format.

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 at the following location: 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 web service 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

```
1   | 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 Speech SDK can add to a Speech API recognition request. The Speech SDK will add most of these automatically.

Name	Description
`ClientSdk`	The name and version of the Speech SDK software capturing audio for the Speech API service. 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 device's operating system. 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 application making the request. On Android, use the package identifier; on iOS use the bundle identifier.
`ClientVersion`	The version of the application 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 Android Activity or Fragment or 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 web service 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 invalid, the web service request will fail with an HTTP 401 `Unauthorized` error.

3.5.1 OAuth Access Token Request

To get a new access token, the client application 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 developer's API key and API secret. The request body uses standard form encoding (application/x-www-form-urlencoded) and has the following template:

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

The fields of the request body are:

- ***client_id***: The client ID (API key) for the developer's application.
- ***client_secret:*** The client secret (API secret) for the developer's application.
- ***scope:*** A parameter that describes the set of services that the application will use. For Speech API services, use a scope of SPEECH.

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

3.5.2 OAuth Access Token Response

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

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

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.
- ***expires_in:*** The number of seconds this token is valid. When this value is 0, the token has an unspecified duration. It is recommended that applications request a new access token every two hours.
- ***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 API key and secret in an OAuth token request. The refresh token has a lifetime of about one day. After the refresh token expires, a client application has to use the API key and secret to renew an access token. As a consequence, it's probably easier for a client application to forego using refresh tokens and simply use the API key and secret for authentication.

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

Unauthorized Credentials Response

```
1   | {
2   |     "error":"invalid_client"
3   | }
```

Example 3-3: Unauthorized credentials response

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 Speech API services. 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 Web Service Responses

The AT&T 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 recognition service:

Status Code	Meaning	Description

Status 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 invalid.
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.

3.6.2 Response Headers

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

Header Name	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-Size`	The server may either send the response incrementally by using a chunked encoding or the server may send the response all at once by specifying a content size.
`WWW-Authenticate: Bearer`	This header is included in the error responses when the request omits the credentials of the developer.

Table 3-5: Response headers

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.

Refer to the Speech API technical documentation for complete reference information on the three formats.

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 applications. Refer to the Speech API technical documentation for full reference information.

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

JSON Format Hierarchy

```
1   | {
2   | :   "Recognition":
3   | :   {
4   | :   :   "ResponseId":"e6bf3236ad938ca19c28f95a9065c813",
5   | :   :   "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: JSON format hierarchy

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 class of the object.
- **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 are inappropriate for machine interpretation.

3.8.3 JSON Objects and Fields

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

3.8.3.1 Top-Level Field

The following field is the sole field at the top-level.

- **`Recognition`:** Object (required). The top-level `Recognition` object contains all of the useful data of a transcription. For more information, 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 not common: `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: The current Speech API release has 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 optional 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 hypothesis should be independently confirmed 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, but 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”
- TextMessage:
  - `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 only contains words that are in the `ResultText` string. The `WordScores` array contains the confidence values for each of these words.
- **`WordScores`:** Array of numbers (required array of numbers). 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 AT&T Speech API recognition service, it is 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 is a summary of the cURL options used in the subsequent examples.

- The `--request` parameter tells cURL to issue an HTTP POST.

- The `–H` or `--header` flag adds headers to the HTTP request. The Speech API requires the `Content-Type`, `Authorization`, and `X-SpeechContext` headers on speech requests.
- The `–v` or `--verbose` flag 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.
- The `–i` or `--include` flag tells cURL to echo the headers of the HTTP response.

3.9.2 Example of 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 `$API_KEY` and `$API_SECRET` to the values obtained from the AT&T Developer Program web site for your application.

cURL OAuth request

```
1   | curl --include --data
      "client_id=$API_KEY&client_secret=$API_SECRET&grant_type=client_cr
      edentials&scope=SPEECH"  "https://api.att.com/oauth/token"
2   | HTTP/1.1 200 OK
3   | Content-Length: 131
4   | Content-Type: application/json
5   |
6   | {
7   | "access_token":"0123456789abdcdef0123456789abcdef",
8   | "expires_in":"0",
9   | "refresh_token":"0123456789abcdef0123456789abcdef01234567"
10  | }
```

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 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 applications must check both the status code and the contents of the response data.

To test the Speech API recognition service with cURL, you need an audio file in 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

```
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/rest/2/SpeechToText"
2   |
3   | HTTP/1.1 200 OK
4   | Content-Length: 417
5   | Content-Type: application/json
6   |
7   | {"Recognition":{"ResponseId":"3cbb3879c89227b3e536c5a827e54832","N
      Best":[{
      "WordScores":[0.36,0.15,0.039,0.039,0.059,0.079,0.09,0.1,0.129,0.0
      79],
      "Confidence":0.113000001,"Grade":"accept","ResultText":"Hey ... I
      will see you there unless it's urgent.",
      "Words":["Hey","...","I","will","see","you","there","unless","it's
      ","urgent."],"LanguageId":"en-us","Hypothesis":"Hey ... I will see
      you there unless it's urgent."}]}}
```

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 had invalid credentials: either the request lacked an `Authorization` header or the token in the header was invalid. When a client application 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 that has invalid credentials. The key line is line 3 which contains the `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/rest/1/SpeechToText"
2   |
3   | HTTP/1.1 401 Unauthorized
4   | Content-Length: 123
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  |     }
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 will return a status code of 400 when the request was malformed. This can occur if the client application 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 application 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 application 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 and the JSON body with a `RequestError` element.

Error 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/rest/1/SpeechToText"
2   |
3   | HTTP/1.1 400 Bad Request
4   | Content-Length: 113
5   | Content-Type: application/json
6   |
7   | {"RequestError": {
9   |     "serviceException": {
10  |         "MessageId": "SVC0001",
11  |         "Text": "Invalid value for part %1"
12  |     }
13  |   }
14  | }
```

Example 3-8: Error cURL response.

4 Speech SDK for Android

This section describes how to add speech recognition to Android applications using version 1.6 of the AT&T Speech SDK for Android.

Android is a software platform for mobile devices and tablets. The AT&T Speech SDK supports applications developed using the Android Software Development Kit and distributed by Google Play. The Android SDK is available for download from http://developer.android.com, which provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. The AT&T Speech SDK supports applications that target version 2.2 and higher of the Android platform.

4.1 How the Speech SDK Works on Android

The Speech SDK is packaged as a library called `ATTSpeechKit` that links to your Android application code. The library implements a client for the Speech API recognition web service. It provides the following classes to perform speech recognition on the Android platform:

- **`ATTSpeechActivity`**: A high-level API that uses a child Activity to perform a speech interaction. It is straightforward to convert code that uses Android's `RecognizerIntent` API to use the `ATTSpeechActivity` API.
- **`ATTSpeechService`**: A lower-level API that gives an application more control over the speech interaction.

Both classes in the `ATTSpeechKit` library perform the following tasks:

- Capturing audio from the device microphone.
- Showing feedback to the user.
- Streaming audio to the AT&T Speech API recognition web service.
- Waiting for a response from the service.

When your application wants to accept speech input from the user, it configures properties on instances of the `ATTSpeechKit` classes and calls a method to start the interaction. The `ATTSpeechKit` library 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 `ATTSpeechKit` library returns the result of recognition to your application.

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

4.2 Speech SDK Prerequisites for Android

Before using the Speech SDK for Android, you should be familiar with how to program Android activities using Java. To get started with the Speech SDK for Android, prepare your development environment by completing the following steps:

1. Install the Android SDK. Follow the instructions on the Android Developer website to install the SDK.
2. Install the Android Developer Tools plug-in for Eclipse (optional). Android has close integration with Eclipse for building and running applications. Follow the instructions on the Android Developer website if you want to use Eclipse.

4.3 Setting up your Android Project

Complete the following steps to add the Speech SDK to your Eclipse project and configure it for speech recognition:

1. Unzip the Speech SDK ZIP file. The resulting folder contains the `ATTSpeechKit.jar` file, which is the code library that you will link to your application.

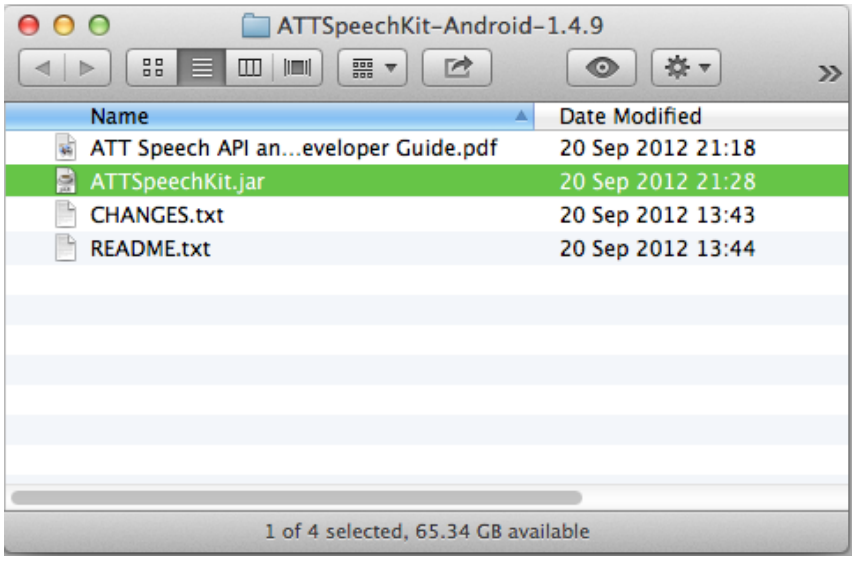

Figure 4-1: Speech SDK ZIP file.

2. Add a new folder to your Android project directory called `libs` (if you do not already have one). Copy the `ATTSpeechKit.jar` file into the `libs` folder.

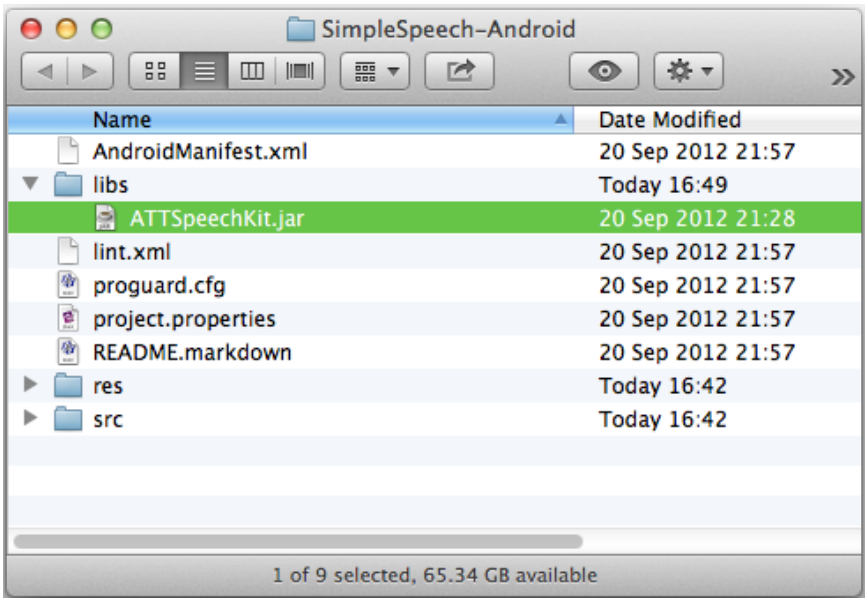

Figure 4-2: New libs folder to your Android project.

3. Make sure the `ATTSpeechKit.jar` file is on your project's build path. In Eclipse, right-click on the `libs/ATTSpeechKit.jar` file to bring up a context menu. Choose the menu item Build Path > Configure Build Path. Look under Android Dependencies in the Libraries tab. `ATTSpeechKit.jar` should appear in that list. If not, choose Add JARs and add `ATTSpeechKit.jar`.

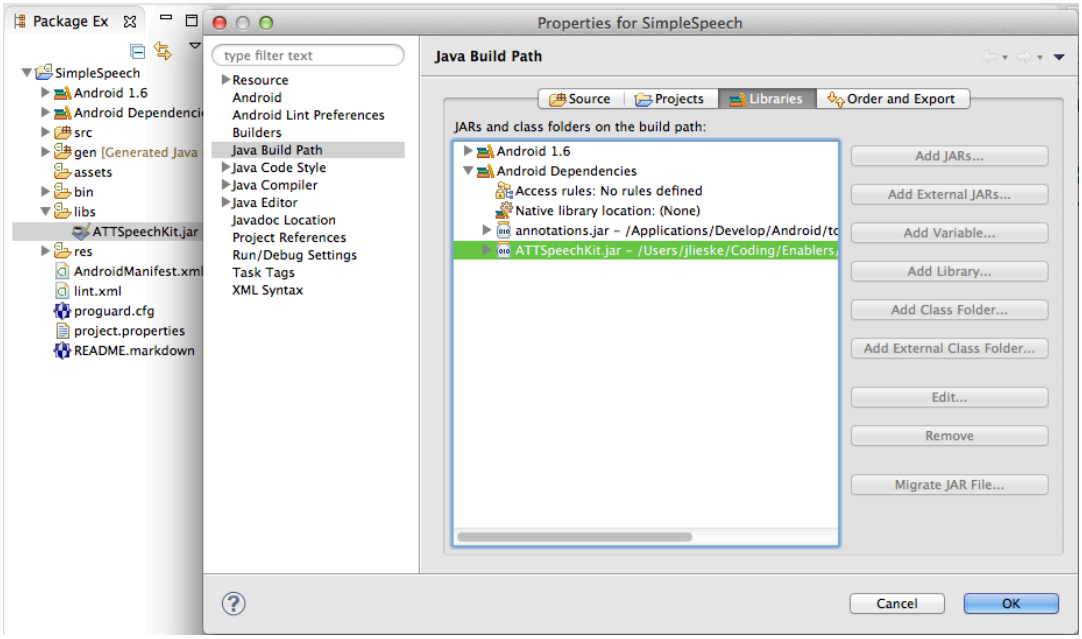

Figure 4-3: Java build path.

4. Configure your project's Android manifest to enable microphone and network access. In the `AndroidManifest.xml` file, add the following uses-permission items before the `<application>` tag (or at the same place where the other `uses-permission` items are located).

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

5. If your application uses the `ATTSpeechActivity` API, configure your project's Android manifest to refer to the `ATTSpeechActivity` class. Add the following activity between the `<application ...>` and `</application>` tags In your `AndroidManifest.xml` file:

```
<activity android:name="com.att.android.speech.ATTSpeechActivity"
android:theme="@android:style/Theme.Translucent.NoTitleBar"
android:configChanges="mcc|mnc|locale|touchscreen|keyboard|keyboardHi
dden|navigation|orientation|fontScale" />
```

4.4 Speech Recognition Tasks on Android

Android applications are composed of activities, which are objects that derive from the `android.activity.Activity` class. In a typical application, each screen of the application consists of a distinct activity object. The Speech SDK provides the `ATTSpeechActivity` class to let developers leverage this standard Android design pattern. `ATTSpeechActivity` has a programming model similar to Android's built-in `RecognizerIntent` interface.

If an application needs more control over the speech interaction, then it can use the `ATTSpeechService` API in the Speech SDK. This API lets the developer supply listener objects to the Speech SDK, which calls the listeners at various stages of the speech interaction. For example, an application that wants to display a custom audio feedback UI can install an `ATTSpeechAudioLevelListener` to get audio level values while the user speaks.

To add speech recognition to your application using either `ATTSpeechActivity` or `ATTSpeechService`, you need to write code that performs the following functions:

- (`ATTSpeechActivity` only) Configure the Android manifest to refer to the `ATTSpeechActivity` class.
- Configure the Speech SDK runtime properties.
- Acquire an OAuth access token.
- Start speech interaction at the appropriate time (for example, in a button press handler).
- (Optional) Customize UI during speech interaction.
- (Optional) Customize speech endpointing.
- Handle speech recognition results and errors.

4.4.1 Configure the Android Manifest

Every Android application's `AndroidManifest.xml` file has a list of Activity classes contained in the application. When your application links the AT&T Speech SDK and calls the `ATTSpeechActivity` API, Android will use the information in the manifest to run the child activity. To use `ATTSpeechActivity`, ensure that the following appears in the Android manifest:

ATTSpeechActivity in AndriodManifes.xml file

```
1   |  <activity android:name="com.att.android.speech.ATTSpeechActivity"
2   |  android:theme="@android:style/Theme.Translucent.NoTitleBar"
3   |  android:configChanges="mcc|mnc|locale|touchscreen|keyboard|keyboar
       dHidden|navigation|orientation|fontScale" />
```

Example 4-1: ATTSpeechActivity in AndriodManifes.xml file

4.4.2 Configure Speech SDK Runtime Properties

The AT&T Speech SDK needs to be configured before an application uses it to call the Speech API. The mechanism is different between `ATTSpeechActivity`

and ATTSpeechService, but the configuration data is largely the same. For a full list of the configuration properties, see Table 5-2 Android Request Properties.

To perform speech recognition with the ATTSpeechActivity API, an application uses the standard Android techniques to start a child activity. Your activity creates an Intent object and sets properties (called "extras") on it to configure a speech recognition request.

When using the ATTSpeechService API, instead of using an Intent object, your application will call methods on the ATTSpeechService singleton object. Obtain a reference to the ATTSpeechService singleton object by calling the getSpeechService() method.

Note: The ATTSpeechService singleton is per-Activity; do not save the singleton in a global variable across multiple Activities.

A mandatory configuration property to set is the URL of the Speech API recognition service that your application uses. Set this via the ATTSpeechActivity.EXTRA_RECOGNITION_URL Intent extra or the method ATTSpeechService.setRecognitionURL(). Get this URL from your application's on-boarding information on the AT&T Developer Program website. For more information on the URL template, see Section 3.1.1 Speech Recognition Service URL.

A required property for the Speech API is the name of the speech context that your application wants to use for recognition. Set this via the ATTSpeechActivity.EXTRA_SPEECH_CONTEXT Intent extra or the ATTSpeechService.setSpeechContext() method. For more information on the available speech contexts, see Section 2.1.2 Speech Contexts.

In addition to setting the Speech API configuration properties, your application must also acquire an OAuth token as described below. It will add the token string to the configuration via the ATTSpeechActivity.EXTRA_BEARER_AUTH_TOKEN Intent extra or by calling the ATTSpeechService.setBearerAuthToken() method.

Programs using ATTSpeechService must also set the listener objects that the Speech SDK will call during a speech interaction. It is required to set the result and error listeners via setSpeechResultListener() and setSpeechErrorListener(). The other listener objects are optional.

4.4.3 Acquire OAuth Access Token

Requests to AT&T Speech API must include a valid OAuth access token. To avoid authentication failures during a speech interaction, your application should acquire 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 should save the token in an instance variable for use in speech interactions.

Since the network I/O required to fetch the OAuth token can take a fraction of a second, it's important that the request be done 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. If the users of your application will keep it in the foreground for longer than an hour, you need to add code to request a token at hourly intervals.

4.4.4 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 Speech SDK will turn on the microphone and begin streaming data in compressed AMR format to the Speech API server. The application code handling the button press makes different calls depending on whether it uses the `ATTSpeechActivity` or `ATTSpeechService` API.

4.4.4.1 Starting Speech using ATTSpeechActivity

An application using the `ATTSpeechActivity` API will use Android system calls to start a child activity: it passes an Intent object to the `startActivityForResult()` method. Android will call your activity's `onActivityResult()` method when speech processing is complete or if there is an error. For more information on the `onActivityResult()` method, see Example 4-4 Minimal `onActivityResult` Method.

The following example illustrates the sequence of calls to construct an `Intent` and initiate a speech interaction. Your application could make these calls, for example, in an action handler that is called when a user taps the Record button in the UI of your application.

Starting ATTSpeechActivity

```
1   |  public void myStartSpeechRecognition()
2   |  {
3   |      Intent request = new Intent(this, ATTSpeechActivity.class);
4   |      request.putExtra(ATTSpeechActivity.EXTRA_RECOGNITION_URL,
        MY_SPEECH_URL);
5   |      request.putExtra(ATTSpeechActivity.EXTRA_SPEECH_CONTEXT,
        "Generic");
6   |      request.putExtra(ATTSpeechActivity.EXTRA_BEARER_AUTH_TOKEN,
        myOAuthToken);
7   |      startActivityForResult(request, MY_SPEECH_INTERACTION);
8   |  }
```

Example 4-2: Starting ATTSpeechActivity

4.4.4.2 Starting Speech using ATTSpeechService

An application using `ATTSpeechService` calls the `startListening()` method on the singleton object to begin a speech interaction with microphone input. Your callback interface instance will be called at various stages through the speech interaction until it receives the recognition result or an error.

The following example illustrates the sequence of calls to configure the `ATTSpeechService` singleton and initiate a speech interaction. Your application could make these calls, for example, in an action handler that is called when a user taps the Record button in the UI of your application.

Starting ATTSpeechService

```
1   |  public void myStartSpeechRecognition()
2   |  {
3   |      ATTSpeechService speechSvc =
        ATTSpeechService.getSpeechService(this);
4   |      speechSvc.setSpeechResultListener(mySpeechResultListener);
5   |      speechSvc.setSpeechErrorListener(mySpeechErrorListener);
6   |      speechSvc.setRecognitionURL(MY_SPEECH_URL);
7   |      speechSvc.setSpeechContext("Generic");
8   |      speechSvc.setBearerAuthToken(myOAuthToken);
9   |      speechSvc.startListening();
10  |  }
```

Example 4-3: Starting ATTSpeechService

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. 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. Speech SDK will display this UI by default; your application can disable this UI by setting the `ATTSpeechActivity.EXTRA_SHOW_UI` Intent extra to false or calling `ATTSpeechService.setShowUI(false)`. When displaying the standard UI, your application can customize the labels in the alert through several properties. For more information on the UI properties, see Table 5-2 Android Request Properties.

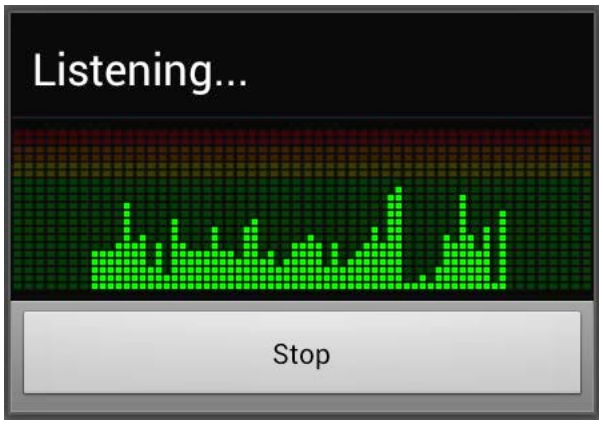

Figure 4-4: Speech SDK Standard UI on Android

An application using the `ATTSpeechService` API has the option of displaying its own UI during a speech interaction. It can register listener objects that receive updates during the recording and processing. Implement the `ATTSpeechAudioLevelListener` interface to receive an average input volume level roughly every 1/10 second. `ATTSpeechStateListener` receives notifications when microphone input starts and ends. For more information on listener callbacks, see Section 5.3 Android `ATTSpeechService` Callbacks.

The Speech SDK automatically ends recording when the user stops talking. Additionally, your application can manually stop capturing audio by calling the `ATTSpeechService.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 `ATTSpeechService.cancel()` method to abort the speech interaction. Notice that your `ATTSpeechErrorListener` will not get a callback from the service if the `cancel` method 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 Android:

- **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 `ATTSpeechActivity.EXTRA_ENDING_SILENCE` Intent extra or calling the `ATTSpeechService.setEndingSilence()` method.
- **Timeout Expiration:** An application using the Speech SDK can set the maximum duration of speech input through the `ATTSpeechActivity.EXTRA_MAX_RECORDING_TIME` Intent extra or the `ATTSpeechService.setMaxRecordingTime()`method.
- **Manual Endpointing:** The user may press a button in the application'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 `ATTSpeechService.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 back to your application. The exact mechanism is different for `ATTSpeechActivity` and `ATTSpeechService`, but similar information is provided by both classes. Your callback methods can obtain an array of transcription hypotheses or the raw JSON response from the speech service. For the full list of response properties, see Table 5-4 Android result properties.

If an error occurs during a speech interaction, the Speech SDK will call back to your application with the error information. Possible errors include the user canceling, the user not speaking in time, problems with the network connection,

and errors reported by the server. For more information on interpreting error callbacks, see Section 5.5 Android Error Properties.

4.4.7.1 Handling ATTSpeechActivity Results

When your app uses the `ATTSpeechActivity` API for a speech interaction, the Speech SDK calls your activity's `onActivityResult(requestCode, resultCode, resultIntent)` method when processing is complete or an error occurs. The interpretations of the `resultCode` argument are as follows:

- **RESULT_OK:** Indicates that the processing was successful. The `resultIntent` argument contains the "extras" properties with the recognition response. An array of transcription hypothesis strings is available in the EXTRA_RESULT_TEXT_LIST extra property, and the raw binary data is in the EXTRA_RESULT_RAW_BYTES property. For the full list of response properties, see Table 5-4 Android result properties.
- **RESULT_CANCELED:** Indicates that the user canceled the interaction.
- **Other values:** Indicate that an error occurred. The `resultCode` value describes the type of error, and the `resultIntent` argument includes "extras" properties with diagnostic information. For a description of the error codes and properties, see Section 5.5 Android Error Properties.

The following example shows a minimal implementation of the `onActivityResult()` method.

onActivityResult method

```
1   | public void onActivityResult(int requestCode, int resultCode,
      Intent resultIntent)
2   | {
3   |     if (requestCode == MY_SPEECH_INTERACTION) {
4   |         if (resultCode == RESULT_OK) {
5   |             // Retrieve text recognized from speech
6   |             ArrayList<String> nbest =
      resultIntent.getStringArrayListExtra(ATTSpeechActivity.EXTRA_RESUL
      T_TEXT_LIST);
7   |             if ((nbest != null) && (nbest.size() > 0)) {
8   |                 String text = nbest.get(0);
9   |                 Log.v(MY_ACTIVITY, "Recognition result: "+text);
10  |             }
11  |             else
12  |                 Log.v(MY_ACTIVITY, "Speech was silent or not
      recognized.");
13  |         }
14  |         else if (resultCode == RESULT_CANCELED)
15  |             Log.v(MY_ACTIVITY, "User canceled.");
16  |         else {
```

```
17  |             String error =
    resultIntent.getStringExtra(ATTSpeechActivity.EXTRA_RESULT_ERROR_M
    ESSAGE);
18  |             Log.v(MY_ACTIVITY, "Recognition failed with error:
    "+error);
19  |         }
20  |     }
21  | }
```

Example 4-4: Minimal `onActivityResult` Method

4.4.7.2 Handling ATTSpeechService Results

When your app uses ATTSpeechService for a speech interaction, the Speech SDK calls the `ATTSpeechServiceResultListener` interface when processing is complete. It passes an `ATTSpeechResult` argument that exposes properties for the application to get the recognition response data. An array of transcription hypothesis strings is available in the `getTextStrings()` method. The parsed JSON data from the service is in the `getJSON()` method, and the raw binary data is in the `getRawData()` method. For the full list of response properties, see Table 5-4 Android result properties.

The following example shows a minimal implementation of `ATTSpeechServiceResultListener` that examines the array of transcription hypotheses.

Successful Transaction Listener

```
1   | class ResultListener implements ATTSpeechResultListener {
2   |     public void onResult(ATTSpeechResult result)
3   |     {
4   |         List<String> nbest = result.getTextStrings();
5   |         if (nbest != null && nbest.size() > 0)
6   |             Log.v("MyApp", "Recognition result: "+nbest.get(0));
7   |         else
8   |             Log.v("MyApp", "Speech was silent or not
    recognized.");
9   |     }
10  | }
```

Example 4-5: Successful Transaction Listener

If an error occurs doing the speech recognition process, the Speech SDK calls the `ATTSpeechErrorListener` callback interface. Possible errors include the user canceling, the user not speaking in time, problems with the network connection, and errors reported by the server. The `ATTSpeechError` argument exposes properties for the application to get the information about the error. For more information on interpreting error callbacks, see Section 5.5 Android Error Properties.

The following example shows a minimal implementation of the `ATTSpeechErrorListener` interface that checks the error status.

Failed Transaction Listener

```
1   | class ErrorListener implements ATTSpeechErrorListener {
2   |     public void onError(ATTSpeechError error)
3   |     {
4   |         ErrorType resultCode = error.getType();
5   |         if (resultCode == ErrorType.USER_CANCELED)
6   |             Log.v("MyApp", "User canceled.");
7   |         else
8   |             Log.v("MyApp", "Recognition failed with error
      "+resultCode+": "+error.getMessage());
9   |     }
10  | }
```

Example 4-6: Failed Transaction Listener

5 Speech SDK Reference for Android

This section is a reference for the ATTSpeechKit library on Android. It describes the classes, methods, properties, callback listeners, error codes, and state transitions in both the `ATTSpeechActivity` and `ATTSpeechService` APIs.

- Android `ATTSpeechService` Methods
- Android Request Properties
- Android `ATTSpeechService` Callbacks
- Android Result Properties
- Android Error Properties
- Android State Transitions

5.1 Android ATTSpeechService Methods

The following table describes the methods that the `ATTSpeechService` class implements. (The `ATTSpeechActivity` API does not support these methods. It supports the standard Android `startActivityForResult()` method.)

Method	Required/ Optional	Description
`getSpeechService(Activity activity)`	Required	This static method returns the singleton speech service object for the activity.
`startListening()`	Required	Starts a speech interaction using the microphone. Before calling this method, set the request properties on the `ATTSpeechService` object to configure the interaction. After the method is called, the Speech SDK will invoke methods on the delegate object to report interaction status, errors, and the recognition result.

Method	Required/ Optional	Description
`startWithAudioData(byte[] audioData)`	Required	Sends audio data to the Speech 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 no compression of the data, so the file should already be in a format supported by the Speech API recognition service. After this method is called, the `ATTSpeechService` instance will invoke methods on the callback objects to report interaction status, errors, and the recognition result.
`stopListening()`	Optional	Manually ends speech input. This method is optional because the `ATTSpeechKit` library automatically ends 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 report the result or error to the listener objects.
`cancel()`	Optional	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 listener callbacks related to the canceled interaction.

Table 5-1: Android ATTSpeechService Methods

5.2 Android Request Properties

The following table describes the Android Speech SDK properties that control a speech interaction.

For developers using the `ATTSpeechActivity` class, the table lists the `Intent` extras that control speech recognition in an `ATTSpeechActivity` request. Your application sets these extra properties on the `Intent` that it passes to the `startActivityForResult()` method. The property names are all static constants on the `ATTSpeechActivity` class.

For developers using the `ATTSpeechService` class, the table describes the methods on that class that control speech requests.

ATTSpeechActivity Extra/ ATTSpeechService Method	Default Value	Type	Required/ Optional	Description

`EXTRA_RECOGNITION_URL`
 	
`setRecognitionURL()`
		`String`	Required	The URL of the Speech API recognition service. For more information, see Section 3.1.1 Speech Recognition Service URL.

`EXTRA_AUDIO_DATA`		`byte[]`	Optional	The audio data to submit to the Speech API recognition service for processing. When this property is omitted, the `ATTSpeechActivity` gets input from the microphone. (`ATTSpeechActivity` only. When using `ATTSpeechService`, use `startWithAudioData()`.)

`EXTRA_CONTENT_TYPE`
 	
`setContentType()`
	`"audio/amr"`	`String`	Optional	MIME type of the audio data for the server. The value must be a MIME type such as `"audio/amr"`. It is used in the `Content-Type` HTTP header. This property is needed only when uploading audio data.

`EXTRA_SPEECH_CONTEXT`
 	
`setSpeechContext()`
		`String`	Required	The name of the Speech API recognition context to use for recognition. It is used in the `X-SpeechContext` HTTP header.

`EXTRA_XARGS`		`String`	Optional	A set of extra arguments for the Speech API recognition service. The parameter is a string of comma-separated *name*=*value* pairs, with the values percent-encoded. It is used in the `X-Arg` HTTP header. (ATTSpeechActivity only.)

ATTSpeechActivity Extra/ ATTSpeechService Method	Default Value	Type	Required/ Optional	Description
setXArgs()		Map <String, String>	Optional	A set of extra arguments for the Speech API recognition service. The parameter is a mapping of name-value pairs, with the values not encoded. It is used in the `X-Arg` HTTP header. (`ATTSpeechService` only.)
`EXTRA_SENDS_DEFAULT_XARGS` `sendsDefaultXArgs()`	`true`	`boolean`	Optional	Leaving this property true 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 the property is set to false, Speech SDK will not add the `X-Arg` values.
`EXTRA_BEARER_AUTH_TOKEN` `setBearerAuthToken()`		`String`	Required	The OAuth access token that validates speech requests from this application. It is used in the `Authentication: Bearer` HTTP header.
`setRequestHeaders()`		`Map <String, String>`	Optional	A collection of HTTP headers to add to the request. (`ATTSpeechService` only.)
`EXTRA_REQUEST_HEADER_NAMES`		`String[]`	Optional	An array of HTTP header names to add to the request. The `EXTRA_REQUEST_HEADER_VALUES` property must also be set when this property is set. (`ATTSpeechActivity` only.)

ATTSpeechActivity Extra/ ATTSpeechService Method	Default Value	Type	Required/ Optional	Description
`EXTRA_REQUEST_HEADER_VALUES`		`String[]`	Optional	An array of HTTP header values to add to the request. It is a parallel array to the `EXTRA_REQUEST_HEADER_NAMES` property array. (`ATTSpeechActivity` only.)
`EXTRA_SHOW_UI` `setShowUI()`	`true`	`boolean`	Optional	Controls the display of a progress meter and a button for canceling the interaction. Set this property to false to hide the UI.
`EXTRA_RECORDING_PROMPT` `setRecordingPrompt()`	`"Listening"`	`String`	Optional	The text that the Speech SDK displays while capturing audio data by the microphone. Use this property only when the `showUI` property is true.
`EXTRA_RECORDING_STOP_BUTTON` `setRecordingStopButton()`	`"Stop"`	`String`	Optional	The button label that the Speech SDK displays while capturing audio data by the microphone. The user can press the button to manually endpoint audio input. Use this property only when the `showUI` property is true.
`EXTRA_PROCESSING_PROMPT` `setProcessingPrompt()`	`"Processing"`	`String`	Optional	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 true.
`EXTRA_PROCESSING_CANCEL_BUTTON` `setProcessingCancelButton()`	`"Cancel"`	`String`	Optional	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 true.

ATTSpeechActivity Extra/ ATTSpeechService Method	Default Value	Type	Required/ Optional	Description
`EXTRA_CANCEL_WHEN_SILENT` `setCancelWhenSilent()`	`true`	`boolean`	Optional	Specifies how Speech SDK handles silent audio input. When the value is false, Speech SDK will send silent audio to the server for processing. When true, the Speech SDK will automatically cancel the processing when it detects silent audio.
`EXTRA_MAX_INITIAL_SILENCE` `setMaxInitialSilence()`	1500	`int`	Optional	The maximum amount of time, in milliseconds, that a user can remain silent before the Speech SDK aborts the speech interaction. To disable aborting, set the value to `Integer.MAX_VALUE.`
`EXTRA_ENDING_SILENCE` `setEndingSilence()`	`1000`	`int`	Optional	The maximum amount of time, in milliseconds, of silence after the user stops talking when the Speech SDK performs endpointing. To disable endpointing, set the value to `Integer.MAX_VALUE`.
`EXTRA_MIN_RECORDING_TIME` `setMinRecordingTime()`	`500`	`int`	Optional	The minimum amount of time, in milliseconds, to capture audio from the microphone. To disable aborting, set the value to 0.
`EXTRA_MAX_RECORDING_TIME` setMaxRecordingTime `()`	`25000`	`int`	Optional	The maximum amount of time, in milliseconds, that the user can speak into the microphone. When the maximum time is reached, the Speech SDK automatically stops the microphone input and submits the audio for processing. To disable the timeout, set the value to `Integer.MAX_VALUE`.

ATTSpeechActivity Extra/ ATTSpeechService Method	Default Value	Type	Required/ Optional	Description

`EXTRA_CONNECTION_TIMEOUT`
 	
`setConnectionTimeout()`
	`10000`	`int`	Optional	The maximum amount of time, in milliseconds, that the Speech SDK waits for a successful connection to the server when initiating a recognition request. To disable the timeout, set the value to `Integer.MAX_VALUE`.

`EXTRA_SERVER_RESPONSE_TIMEOUT`
 	
`setServerResponseTimeout()`
	`30000`	`int`	Optional	The maximum amount of time, in milliseconds, that the Speech SDK waits for the server to complete a recognition response. To disable the timeout, set the value to `Integer.MAX_VALUE`.

Table 5-2: Android Request Properties

5.3 Android ATTSpeechService Callbacks

While the ATTSpeechService object is performing a speech recognition interaction, it calls methods on the listener objects set by the application. The application is required to set the ATTSpeechResultListener and ATTSpeechErrorListener callbacks that communicate the recognition result to the application. The remaining listeners are optional, giving the application a chance to customize the behavior of the speech interaction.

The following table describes the Speech SDK callback interfaces. (The ATTSpeechActivity API does not support these interfaces. It supports the standard Android onActivityResult() method.)

Listener Interface	Listener Method	Required/ Optional	**Description**
ATTSpeechResultListener	onResult()	Required	The Speech SDK calls this interface when it returns a recognition result. The ATTSpeechResult argument has properties that contain the response data and recognized text. For more information on how to interpret the response data, see Section 5.4 Android Result Properties.
ATTSpeechErrorListener	onError()	Required	The Speech SDK calls this interface when speech recognition fails. The reasons for failure can include the user canceling, network errors, or server errors. For more information on the ATTSpeechError argument, see Section 5.5 Android Error Properties.

Listener Interface	Listener Method	Required/ Optional	**Description**
`ATTSpeechStateListener`	`onStateChanged()`	`Optional`	The Speech SDK calls this interface when it transitions among states in a recording interaction, for example, from capturing to processing.
`ATTSpeechAudioLevelListener`	`onAudioLevel(int)`	`Optional`	The Speech SDK calls this interface repeatedly as it captures audio. An application can use the audio level data to update its UI.

Table 5-3: Android ATTSpeechService Callback Interfaces

5.4 Android Result Properties

When a speech interaction completes successfully, the Speech SDK returns the data described in this section.

For applications that use the `ATTSpeechActivity` API, Android will call the `onActivityResult()` method with a result code of `RESULT_OK` to indicate a successful interaction. The `ATTSpeechActivity` API attaches "extras" properties to the `Intent` object that is passed to the `onActivityResult()` method. The following table describes the available extras; the property names are all static constants on the `ATTSpeechActivity` class.

Applications that use the `ATTSpeechService` API will have their `ATTSpeechResultListener` interface called on a successful interaction. The following table describes the data available on the `ATTSpeechResult` object that is passed to the `onResult()` method.

ATTSpeechActivity Extra/ ATTSpeechService Class Method	Type	Required/ Optional	Description

`EXTRA_RESULT_TEXT_STRINGS`
 	
`getTextStrings()`
	`ArrayList <String>`	Optional	Contains the recognized text as a collection of hypothesis strings. The collection can be null or zero length when the speech was silent or the service did not recognize the speech. This property is present only when the service returns the standard Speech API JSON format.

`EXTRA_RESULT_RAW_DATA`
 	
`getRawData()`
	`byte[]`	Required	The full binary data that is returned by the server.

`EXTRA_RESULT_STATUS_CODE`
 	
`getStatusCode()`
	`int`	Required	The HTTP status code that the speech service returned.

`getResponseHeaders()`	`Map <String, String>`	Required	The HTTP headers that the speech service returned. Header names are case-insensitive via `get()` and preserve the case returned by the service. (`ATTSpeechService` only.)

ATTSpeechActivity Extra/ ATTSpeechService Class Method	Type	Required/ Optional	Description
`EXTRA_RESULT_HEADER_NAMES`	`String[]`	Required	The names of the HTTP headers that the speech service returned. The `EXTRA_RESULT_HTTP_HEADER_VALUES` property contains the corresponding values. (`ATTSpeechActivity` only.)
`EXTRA_RESULT_HEADER_VALUES`	`String[]`	Required	The values of the HTTP headers that the speech service returned. These correspond to the header names in the `EXTRA_RESULT_HTTP_HEADER_NAMES` property. (`ATTSpeechActivity` only.)
`getJSON()`	`JSONObject`	Optional	The parsed JSON data that the speech service returned. This property is present only when the service returns JSON data. (`ATTSpeechService` only.)

Table 5-4: Android result properties

5.5 Android Error Properties

When a speech interaction does not complete successfully, the Speech SDK calls back to your application with information about the error. Possible errors include the user canceling, the user not speaking in time, problems with the network connection, and when the server returns a non-200 HTTP response.

The following table describes the error codes that an `ATTSpeechActivity` or `ATTSpeechService` request returns. When using the `ATTSpeechActivity` API, the result codes are passed in the `resultCode` parameter of the `onActivityResult()` method. When using the `ATTSpeechService` API, the error codes are `ATTSpeechError.ErrorType` enumeration constants accessed by calling the `ATTSpeechError.getType()` method on the `ATTSpeechError` object passed to `ATTSpeechErrorListener`.

ATTSpeechError getType() Value	ATTSpeechActivity Result Code	Description
`ErrorType.USER_CANCELED`	`RESULT_CANCELED`	The user or the application canceled the speech interaction.
`ErrorType.PARAMETER_ERROR`	`RESULT_PARAMETER_ ERROR`	One of the properties set on `ATTSpeechService` or on the `Intent` passed to the `ATTSpeechActivity` class was not valid.
`ErrorType.CAPTURE_FAILED`	`RESULT_AUDIO_ERROR`	There was a problem capturing data from the microphone.
`ErrorType.BELOW_MINIMUM_LENGTH`	`RESULT_AUDIO_LENGTH_ERROR`	The user's audio was too short.
`ErrorType.INAUDIBLE`	`RESULT_INAUDIBLE_ ERROR`	The user was silent and `cancelWhenSilent` was set to true.
`ErrorType.CONNECTION_ERROR`	`RESULT_CONNECTION_ ERROR`	A problem occurred while connecting to the server.
`ErrorType.RESPONSE_ERROR`	`RESULT_RESPONSE_ ERROR`	A problem occurred while reading the response over the network.

ATTSpeechError getType() Value	ATTSpeechActivity Result Code	Description
`ErrorType.SERVER_ERROR`	`RESULT_SERVER_ERROR`	The server reported an HTTP error. For the data returned by the server, see Table 5-6 Android Error Properties.
`ErrorType.OTHER_ERROR`	`RESULT_CLIENT_ERROR`	Some other error occurred.

Table 5-5: Android Error Codes

Table 5-6 Android Error Properties shows the additional data that is returned with an error result. This additional data, known as “extras” properties, are attached to the `Intent` object that is passed to the `onActivityResult()` method when using the `ATTSpeechActivity` API. Note that some of the properties are present only for HTTP errors reported with the result code `ATTSpeechActivity.RESULT_SERVER_ERROR`. This table lists the available extras; the property names are all static constants on the `ATTSpeechActivity` class.

Applications that use the `ATTSpeechService` API will have their `ATTSpeechErrorListener` callback called on an error result. The Speech SDK will pass an `ATTSpeechError` object to the `onError()` method. Notice that some of the properties are present only for HTTP errors reported with the result code `ATTSpeechError.ErrorType.SERVER_ERROR`; call the `ATTSpeechError.getResult()` method to access the server’s HTTP response data for such errors. The following table describes the data available on the `ATTSpeechError` and `ATTSpeechResult` objects passed to the `onError()` method.

ATTSpeechActivity Extra/ ATTSpeechService Class Method	Type	Required/ Optional	Description

`EXTRA_RESULT_ERROR_MESSAGE`
 	
`ATTSpeechError.getMessage()`
	`String`	Required	A string from the Speech SDK that contains diagnostic information for network or server errors. It is unsuitable for display to the user.

ATTSpeechActivity Extra/ ATTSpeechService Class Method	Type	Required/ Optional	Description

`EXTRA_RESULT_STATUS_CODE`
 	
`ATTSpeechResult.getStatusCode()`
	`int`	Optional	The HTTP status code returned by the speech server. It is included only for errors of type `SERVER_ERROR`.

`ATTSpeechResult.getResponseHeaders()`	`Map <String, String>`	Optional	The HTTP headers that the speech service returned. Header names are case-insensitive via `get()` and preserve the case returned by the service. It is included only for errors of type `SERVER_ERROR`. (`ATTSpeechService` only.)
`EXTRA_RESULT_HEADER_NAMES`	`String[]`	Optional	The names of the HTTP headers that the speech service returned. `EXTRA_RESULT_HEADER_VALUES` contains the corresponding values. It is included only for `errors` of type `SERVER_ERROR`. (`ATTSpeechActivity` only.)

ATTSpeechActivity Extra/ ATTSpeechService Class Method	Type	Required/ Optional	Description
`EXTRA_RESULT_HEADER_VALUES`	`String[]`	Optional	The values of the HTTP headers that the speech service returned. These correspond to the header names in `EXTRA_RESULT_HEADER_NAMES`. It is included only for errors of type `SERVER_ERROR`. (`ATTSpeechActivity` only.)
`EXTRA_RESULT_RAW_DATA` `ATTSpeechResult.getRawData()`	`byte[]`	Optional	The full binary data that is returned by the. It is included only for errors of type `SERVER_ERROR`.

Table 5-6: Android Error Properties

5.6 Android State Transitions

An application that wants to be notified about the progress of a speech interaction can provide an `ATTSpeechStateListener` object to `ATTSpeechService`. Speech SDK will call the listener with values of the `ATTSpeechStateListener.SpeechState` enumeration when it enters a new stage of the interaction. The following table describes the stages that occur during speech interaction.

State	Description
`SpeechState.IDLE`	Nothing occurring. `ATTSpeechService` is in this state before beginning a speech interaction and after delegate callbacks have returned.
`SpeechState.INITIALIZING`	`ATTSpeechService` is beginning a speech interaction, but it is not ready to accept audio yet. This stage can be skipped in some situations.
`SpeechState.RECORDING`	Audio capture is taking place.
`SpeechState.PROCESSING`	The server is processing audio data.
`SpeechState.ERROR`	ATTSpeechService is handling an error condition.

Table 5-7: ATTSpeechStateListener.SpeechState Values

6 Android Testing

The Speech SDK supports the tools of the Android platform for running automated tests. For more information on testing the Android application, refer to the topic *Testing* at the following location:

http://developer.android.com/guide/topics/testing/index.html

Note: The Android Emulator has limited support for audio input. The emulator's software codecs tend to distort the audio, which causes speech recognition to fail. For best results, use actual Android devices instead of the emulator for testing speech applications.

7 Android Deployment

Developers who create speech-enabled applications on Android devices must link their applications to the `ATTSpeechKit` library JAR, so that the library code is embedded in the application's application package (APK) file. There is nothing additional that must be added to distribute the application to end-users.

8 Appendix A: Index