Hewlett Packard Enterprise

IDOL Speech Server

Software Version: 11.5

Administration Guide

Document Release Date: October 2017

Software Release Date: October 2017

Legal notices

Warranty

The only warranties for Hewlett Packard Enterprise Development LP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted rights legend

Confidential computer software. Valid license from HPE required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright notice

© Copyright 2015-2017 Hewlett Packard Enterprise Development LP

Trademark notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent software updates, go to https://downloads.autonomy.com/productDownloads.jsp.

To verify that you are using the most recent edition of a document, go to https://softwaresupport.hpe.com/group/softwaresupport/search-result?doctype=online help.

This site requires that you register for an HPE Passport and sign in. To register for an HPE Passport ID, go to https://hpp12.passport.hpe.com/hppcf/login.do.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

Visit the HPE Software Support Online web site at https://softwaresupport.hpe.com.

This web site provides contact information and details about the products, services, and support that HPE Software offers.

HPE Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Access product documentation
- Manage support contracts
- Look up HPE support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HPE Passport user and sign in. Many also require a support contract.

To register for an HPE Passport ID, go to https://hpp12.passport.hpe.com/hppcf/login.do.

To find more information about access levels, go to https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

To check for recent software updates, go to https://downloads.autonomy.com/productDownloads.jsp.

About this PDF version of online Help

This document is a PDF version of the online Help.

This PDF file is provided so you can easily print multiple topics or read the online Help.

Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online Help.

Contents

Part 1: Getting Started 15
Chapter 1: Introduction to IDOL Speech Server 17
IDOL Speech Server 17
OEM Certification 17
The IDOL Platform 17
Introduction to Speech Processing 17
Speech-to-Text 18
Phonetic Phrase Search 19
Speaker Identification 19
Spoken Language Identification 19
Transcript Alignment 19
Audio Fingerprint Identification 20
Audio Security 20
IDOL Speech Server System Architecture 20
Chapter 2: Speech Processing Overview 21
Understand Audio 21
Audio Sources and Capture 21
Audio Capture 22
Clipping 22
Background Noise 22
Audio Compression 22
Audio Sampling Rate 23
Audio Bandwidth 24
Speech-to-Text 25
Measure Speech-to-Text Success Rates 26
Improve Speech-to-Text Success Rates 26
Language Models 26
Measure the Effectiveness of Language Models 27
Source Text for Building Custom Language Models 28
Build Custom Language Models 28
Acoustic Models 28
Phonetic Phrase Search 29
Speaker Identification 30
Spoken Language Identification 31
Audio Fingerprint Identification 33
Transcript Alignment 33
Transcript Language Models 33
Chapter 3: Install IDOL Speech Server 35
System Requirements 35
Supported Operating Systems 35

Windows ... 35
Linux ... 35
Antivirus Recommendations ... 35
Recommended Hardware Specifications ... 36
Memory Requirements ... 36
Other Requirements ... 37
Install IDOL Speech Server ... 37
Install IDOL Speech Server on Windows ... 37
Install an IDOL Component as a Service on Windows ... 38
Install IDOL Speech Server on UNIX ... 40
Install an IDOL Component as a Service on Linux ... 41
Install a Component as a Service for a systemd Boot System ... 42
Install a Component as a Service for a System V Boot System ... 43
Licenses ... 43
Display License Information ... 44
Configure the License Server Host and Port ... 45
Revoke a Client License ... 45
Troubleshoot License Errors ... 46
Supported Resources ... 47
Resource Packs ... 48
Sentence Breaking Plug-In ... 51
Chapter 4: Configure IDOL Speech Server ... 53
IDOL Speech Server Configuration Files ... 53
Modify Configuration Parameter Values ... 54
Configuration File Sections ... 54
[Server] Section ... 55
[Service] Section ... 55
[Paths] Section ... 55
[License] Section ... 55
[Logging] Section ... 55
[ActionName] Sections ... 56
[MyEventHandler] Sections ... 56
[ACIEncryption] ... 57
[AuthorizationRoles] ... 57
Tasks Configuration File Sections ... 57
[TaskTypes] Section ... 57
[MyTask] Sections ... 58
[ModuleName] Sections ... 59
[Resources] Section ... 59
[MyLanguage] Sections ... 60
[MyFPDB] Sections ... 61
[MySidBase] Section ... 61
Configure Language Packs ... 62
Configure Task Queues ... 63
Enable Queues ... 63
View Queue Status ... 64

Include an External Configuration File ... 65
Include the Whole External Configuration File ... 65
Include Sections of an External Configuration File ... 66
Include a Parameter from an External Configuration File ... 66
Merge a Section from an External Configuration File ... 67
Configure Client Authorization ... 67
Customize Logging ... 69
IDOL Speech Server Logs and Error Reports ... 70
Event Handlers ... 70
Configure an Event Handler ... 70
Configure Input and Output Directories ... 71
Use XSL Templates to Transform Action Responses ... 72
Chapter 5: Run IDOL Speech Server ... 75
Start and Stop IDOL Speech Server ... 75
Start IDOL Speech Server ... 75
Stop IDOL Speech Server ... 76
Send Actions to IDOL Speech Server ... 76
Send Actions by Using a GET Method ... 77
Send Data by Using a POST Method ... 78
Application/x-www-form-urlencoded ... 78
Multipart/form-data ... 78
Verify IDOL Speech Server Runs Correctly ... 79
GetRequestLog ... 79
GetLicenseInfo ... 79
GetLicenseCounts ... 80
GetStatus ... 82
Display Online Help ... 84
Audio Requirements ... 84
Audio and Video Files ... 85
Streamed Audio ... 85
Start and Stop Tasks ... 85
Start a Task ... 85
Run Tasks Across Multiple Cores ... 86
Check the Status of a Task ... 87
End a Task ... 87
Queue Tasks ... 88
Schedule Tasks ... 88
Configure Schedules ... 88
Create a Schedule File ... 89
Enable the Schedule in the Configuration File ... 90
Schedule a Task by Sending an Action ... 90
Standard Tasks ... 91
Deprecated Tasks ... 94
Display Configured Tasks ... 98
Display Task Details ... 99
Display Configured Resources ... 101

Check Available Resources 102
Find Recommended Language Resources to Use in a Task 103
Get Task Results 105
Results Format 106
Continuation Tokens 106
Delete Output Files 107
Use ClearTmpResults Action 107
Use DeleteResults Action 108
Use DeleteResult Action Parameter with GetResults Action 108
Monitor the Status of the Output File Manager 109
Restrictions on File Deletion 109
Create and Manage Lists 109
Create and Edit Lists 109
View Lists 111
Use Lists in Actions 111
Back Up and Restore IDOL Speech Server 111
Troubleshooting 112

Part 2: IDOL Speech Server Operations 113

Chapter 6: Manage Language Packs and Other Resources 115
Overview 115
Load Language Resources 115
Unload Language Resources 116
Unload Manually Loaded Resources 117
Monitor Resource Usage 117

Chapter 7: Speech-to-Text 119
Prepare the Audio 119
Select the Language Pack 119
Run the Task 120
Interpret the Results 121
CTM Transcript Output 121
XML Transcript Output 122
Pauses in Recognition Output 123
Use a Lattice File 123
Control Speech-to-Text Process Speed 123
Enable Word Confidence Scores 124
Tune Parameters 125
Noisy Data 125
Missing Words in Transcript 125
Extra Words in Transcript 125
Use a Custom Language Model 125
Troubleshooting 126
Generate Diagnostics 126
Score Speech Recognition 127
Prepare the Truth Transcript Text 127

Run the Scorer ... 127
Format of Scorer Results ... 128
Common Problems Related to Scoring Outputs ... 128
Chapter 8: Create Custom Language Models ... 129
Overview ... 129
Standard and Custom Language Models ... 130
Calculate Perplexity ... 130
Format of Perplexity Results ... 131
Look Up Vocabulary ... 132
Look Up Pronunciations ... 133
Training Text for Custom Language Models ... 135
Select Appropriate Text ... 135
Prepare Text ... 135
Normalize Text ... 136
Build the Language Model ... 136
Evaluate the Custom Language Model ... 138
Troubleshooting ... 138
Chapter 9: Normalize Text ... 139
Overview ... 139
Supported Languages ... 139
Segment Text ... 140
Metadata Tag Syntax ... 140
Run Text Normalization ... 142
Troubleshooting ... 143
Chapter 9: Align Transcripts ... 143
Overview ... 143
Common Problems ... 144
Normalize the Transcript ... 145
Build the Transcript Language Model ... 145
Run Speech-to-Text ... 145
Check Transcript ... 145
Format of Diagnostic File ... 146
Align the Transcript ... 147
Two Pass Alignment ... 148
Troubleshooting ... 148
Chapter 10: Phonetic Phrase Search ... 149
Overview ... 149
Create the Phoneme Time Track File ... 149
Search the Phoneme Time Track File ... 150
Phonetic Phrase Search in a Single Step ... 151
Chapter 11: Use Speaker Clustering ... 153
The SplitSpeech Process ... 153
Control the Number of Speakers ... 153
Processing Time ... 154

Chapter 12: Identify Speakers in Audio 155
Overview 155
Create Speaker ID Feature Files 155
Train Speaker Templates 156
Create a Speaker Template from a Single Audio File 157
Create a Speaker Template from Multiple Feature Files 157
Estimate Speaker Score Thresholds 158
Package Templates 162
Identify Speakers in Audio 165
Format of Speaker Identification Results 166
Troubleshooting 167
Generate Diagnostics 168
Warning Messages 168
Improve Performance 169
Error Messages 170
Chapter 13: Identify Languages in Audio 171
Overview 171
Language Identification Modes 171
Open Set Language Identification 172
Configure Language Identification Tasks 172
Run Language Identification 173
Create Your Own Language Classifiers 175
Create Language Identification Feature Files 176
Performance Considerations 176
Create the Language Classifier 177
Combine Classifiers into a Language Identification Set 178
Optimize the Language Identification Set 179
Review the Optimization Log File 181
Troubleshooting 185
Error Messages 186
Chapter 14: Audio Fingerprint Identification 187
Overview 187
Define the AFP Database 187
Define the AFP Database in the Tasks Configuration File 188
Manage AFP Databases 189
Add Clips to a Database 189
Remove Clips from a Database 190
Optimize a Database 191
View the Database Contents 191
Detect Indexed Clips in Audio 193
Detect Repeated Audio Sections 196
Troubleshooting 196
Chapter 15: Audio Security 199
Overview 199
Configure Audio Security Tasks 199

Create the Alarm Database ... 199
Run Audio Security ... 200
Chapter 16: Stream Live Audio ... 203
Run Speech-to-Text on Live Audio ... 203
Remove Areas Categorized as Music or Noise ... 204
Use Live Mode for Streaming ... 204
Chapter 17: Preprocess Audio ... 207
Audio Preprocessor Modes ... 207
Audio Categorization ... 207
Audio Quality Assessment ... 208
Clipping Detection ... 208
Signal-to-Noise Ratio Calculation ... 208
DTMF Identification ... 208
Configure Audio Preprocessing Tasks ... 208
Run an Audio Preprocessing Task ... 209
Example Results ... 210
Audio Categorization Mode ... 210
Clipping Detection Mode ... 210
SNR Calculation Mode ... 211
DTMF Mode ... 211
Run Audio Analysis ... 212
Chapter 18: Postprocess Results ... 213
Postprocessing Operations ... 213
Filter Vocabulary ... 213
Recombine Word Fragments ... 214
Configure Audio Postprocessing Tasks ... 215

Part 3: Create Custom Tasks ... 217
Chapter 19: Understanding Tasks ... 219
Overview ... 219
Modules and Data Streams ... 219
Check Module License Information ... 220
Syntax for Module and Stream Variables ... 221
Name a Data Stream Instance ... 221
Name a Module Instance ... 222
Specify Module Inputs and Outputs ... 225
Schemas for Standard Tasks ... 225
Speech-to-Text ... 225
Language Configuration ... 227
Phonetic Phrase Search ... 229
Speaker Identification ... 230
Spoken Language Identification ... 230
Audio Fingerprint Identification ... 232
Transcript Alignment ... 232

Chapter 20: Configure Custom Tasks 235
General Task Configuration 235
Convert Data Streams Between Permitted Types 235
Disable Time Slicing 235
General Module Configuration 236
Configure Default Outputs 236
Run the Configured Task 236
Configure Variable Parameters 237
Generate Output File Names 237
Configure Action Parameters 238
Set a Default Value 239
Configure Simple References 239
Configure Complex References 239
Load a Language Pack Manually 240
Unload a Language Pack 242
Chapter 21: Sample Task Schemas 245
Stereo Speech-to-Text Conversion 245
Audio Speech-to-Text Conversion 246
Speech-to-Text with Word Filtering 247
Language Identification 247
Language Identification Feature Extraction 248
Language Identification Optimization 249
Transcript Alignment 249
Build Language Models 250
Cluster Speech 250
Train Templates for Speaker Identification 251
Perform Speaker Identification Using Templates 251
Audio Fingerprint Identification 252
Add a Track to an Audio Fingerprint Database 252
Remove a Track from an Audio Fingerprint Database 253
Audio Security 253
Preprocess Audio Files for Phonetic Phrase Search 253
Combine Phonetic Phrase Search Data Files 254
Run a Phonetic Phrase Search 254
Combine Output from Multiple Channels 254
Categorize Audio 255
Assess Audio Quality 255
Identify DTMF Dial Tones 256
Word Recompounding 256

Part 4: Appendixes 259

Appendix A: Input and Output Files 261
Specify Files 261
Input Files 261
Output Files 262

Configuration File 262
Logging Sections 262
[Server] Section 262
Tasks Configuration File 262
[LangPack] Section 262
[afpedit] Section 263
[afpfile] Section 263
[afpout] Section 263
[align] Section 263
[amadaptadddata] Section 264
[amadaptend] Section 264
[audio] Section 264
[audiotemplatedevel] Section 265
[audiotemplateedit] Section 265
[audiotemplatescore] Section 266
[audiotemplatetrain] Section 266
[audiopreproc] Section 266
[audiosec] Section 266
[ctm] Section 267
[filter] Section 267
[fmdcombiner] Section 267
[ivdevel] Section 267
[ivedit] Section 268
[ivfile] Section 268
[ivscore] Section 268
[langid] Section 269
[lbout] Section 269
[lfout] Section 269
[lidoptimizer] Section 269
[lidout] Section 270
[lidtrain] Section 270
[lmbuild] Section 270
[lmtool] Section 270
[normalizer] Section 271
[phraseprematch] Section 271
[plh] Section 271
[postproc] Section 271
[segment] Section 272
[sidout] Section 272
[speakerid] Section 272
[stt] Section 272
[textnorm] Section 272
[textsegment] Section 273
[transcheck] Section 273
[wav] Section 273
[wout] Section 274

Appendix B: IDOL Speech Server File Types 275
File Types 275
Appendix C: Audio Quality Guidelines 279
Appendix D: Audio Transcript Requirements 281
Manually Normalize Text 281
Appendix E: Compatibility Notes 283

Glossary 285

Send documentation feedback 289

Part 1: Getting Started

This section describes how to install and configure IDOL Speech Server.

- Introduction to IDOL Speech Server
- Speech Processing Overview
- Install IDOL Speech Server
- Configure IDOL Speech Server
- Run IDOL Speech Server

Chapter 1: Introduction to IDOL Speech Server

This section introduces IDOL Speech Server and its functions.

- IDOL Speech Server 17
- The IDOL Platform 17
- Introduction to Speech Processing 17
- IDOL Speech Server System Architecture 20

IDOL Speech Server

IDOL Speech Server can perform a variety of speech processing functions, including speech-to-text and language recognition. IDOL Speech Server processes audio data and passes the extracted data into IDOL Server.

OEM Certification

IDOL Speech Server works in OEM licensed environments.

The IDOL Platform

At the core of IDOL Speech Server is the *Intelligent Data Operating Layer* (IDOL).

IDOL gathers and processes unstructured, semi-structured, and structured information in any format from multiple repositories using IDOL connectors and a global relational index. It can automatically form a contextual understanding of the information in real time, linking disparate data sources together based on the concepts contained within them. For example, IDOL can automatically link concepts contained in an email message to a recorded phone conversation, that can be associated with a stock trade. This information is then imported into a format that is easily searchable, adding advanced retrieval, collaboration, and personalization to an application that integrates the technology.

For more information on IDOL, see the *IDOL Getting Started Guide*.

Introduction to Speech Processing

IDOL Speech Server encompasses several speech processing functions in a single ACI server. IDOL Speech Server can perform:

- Speech-to-text
- Phonetic phrase search
- Speaker identification
- Spoken language identification
- Transcript alignment

- Audio fingerprint identification
- Audio security

All speech operations are asynchronous because this approach is more suited to speech processing, especially for live speech. You can send separate requests to the server to access results from the processing operations. This feature allows IDOL Speech Server to report and flag relevant events immediately, so that you do not have to wait until the entire file is processed.

IDOL Speech Server supports multiple languages. A single instance of IDOL Speech Server can process several languages simultaneously.

In addition to audio files, IDOL Speech Server can process audio as binary data sent as data blocks or streams. Sending data as a binary block is useful for processing a local file on a remote server that cannot view the local file system. Audio streaming makes real-time data processing possible–for example, converting incoming audio to text as it occurs.

IDOL Speech Server lets you put together combinations of speech processing functions to create custom operations, allowing you to perform several processes simultaneously on audio data.

The following sections introduce the speech processing functions.

Speech-to-Text

Speech-to-text is the process of translating spoken words into text. It is used in many contexts to analyze, search, and process audio content, including:

- The command and control of mobile devices
- Interactive voice systems for automated call handling
- Dictation of letters, memoranda, and other electronic text documents
- Audio and video search, where the search for specific terms or concepts is performed on the transcript
- Subtitles or closed captions for video
- High-level analysis of phone calls in contact centres

Speech-to-text is an imperfect process and the global research community is still trying to perfect it. Hence all automated systems that use speech-to-text are prone to errors.

There are several factors that affect the success rates of speech-to-text. These include:

- Audio quality metrics, such as signal bandwidth, background noise, and distortions introduced as part of the recording and storing processes
- Overlapping speech from different speakers
- The articulation clarity of the speaker and the speed of their speech
- In some cases, the specific dialect as well as any non-native characteristic of the speaker

There are several methods for measuring the success of speech-to-text:

- Word error rate (or the complementary word accuracy rate)
- Command success rate
- General or specific word recall and precision rates

Speech-to-text is language dependent.

Phonetic Phrase Search

Phonetic search is the process of searching for words and phrases by their pronunciation.

Phonemes are the fundamental units of sounds that make up a spoken language. For example, the word *catch* contains the three phonemes, or sounds, *k–a–tch*. For more information on phonemes, see http://www.sil.org/linguistics/glossaryOfLinguisticTerms/WhatIsAPhoneme.htm.

The phoneme identification engine first processes an audio file to create a time track of phonemes, which reports the time at which each phoneme occurs in the file. This is a one-time process. IDOL Speech Server then searches the phoneme time track data for the specified words or phrases. On an average desktop computer, the search process can operate at a few hundred times faster than real-time.

HPE recommends that you use phonetic phrase search for audio files where the speech-to-text accuracy is not high enough for search purposes.

Phonetic phrase search is language dependent.

Speaker Identification

Speaker identification is the process of identifying a speaker based on their voice characteristics. This is not to be confused with speaker verification or speaker diarization:

- *Speaker verification* is the process of verifying that the speech data matches the identity of a particular speaker.
- *Speaker diarization* is the process of partitioning speech data into homogeneous segments according to speaker identity. The segmentation is often done without identifying who the speaker is.

Speaker identification is text independent and requires speech samples from each speaker to create speaker templates.

Spoken Language Identification

Spoken language identification is the process of determining which natural language is being spoken. It is not necessary to identify the spoken words in the content to determine the language. IDOL Speech Server first tries to identify the phonemes in the audio data and then chooses a language that has the closest distribution of phonemes.

For best results, train the spoken language identification system using sample audio files from each of the languages to be identified.

Spoken language identification is text independent.

Transcript Alignment

Transcript alignment assigns time codes to all the words in the transcript for an audio file. The transcript alignment function can process most transcripts even if they contain noise and missing sections, up to a point. The generated time codes are normally accurate to within half a second.

Transcript alignment is useful for:

- Generating subtitles for videos from manual transcripts.
- Creating time indexes for words in the transcript so that the audio can be searched and positioned.

Speech-to-text is used in the process of generating the time codes.

Audio Fingerprint Identification

Also known as acoustic fingerprinting, audio fingerprint identification generates a digital summary of an audio sample, to identify it quickly or to locate similar samples in a database. You can use audio fingerprinting to:

- Identify songs, melodies, and jingles
- Identify advertisements
- Identify media tracks, where the media track can consist of human voices, music, or any other discernible sounds.

The audio sample to be identified does not need to be an exact copy of the original, but it does need to be based on the same original audio content (for example, you cannot use a cover version of an original song to carry out audio fingerprinting).

Audio Security

Audio security detects and labels segments of audio that contain security-related sounds, including:

- Various alarms, including car alarms
- Breaking glass
- Screams
- Gunshots

IDOL Speech Server System Architecture

IDOL Speech Server uses the Autonomy Content Infrastructure (ACI) Client API to communicate with custom applications that use HTTP commands to retrieve data. This communication is implemented over HTTP by using Extensible Markup Language (XML), and can adhere to Simple Object Access Protocol (SOAP).

IDOL Speech Servercan accept HTTP requests sent from a Web browser. The ACI server architecture also allows IDOL Speech Server to communicate with other ACI servers, such as IDOL server.

Chapter 2: Speech Processing Overview

This section describes the concepts behind the IDOL Speech Server speech processing functions.

- Understand Audio 21
- Speech-to-Text 25
- Phonetic Phrase Search 29
- Speaker Identification 30
- Spoken Language Identification 31
- Audio Fingerprint Identification 33
- Transcript Alignment 33

Understand Audio

To get the best results from IDOL Speech Server, it helps to understand the properties of audio. Most humans cannot detect subtle differences in audio quality. The human brain is excellent at understanding meaning and tends to ignore defects in audio if it can pick out the meaning. Because automatic speech processing technologies use statistical methods, varying attributes and imperfections in the audio alter the patterns computed in the analysis, which can lead to poor results. This can lead to complaints, such as “I can hear the speech well but the speech-to-text output has many errors”. The key to ensuring good results is to avoid imperfections and distortions in the audio.

Audio Sources and Capture

IDOL Speech Server processes both stored and live audio. Audio for processing can be acquired from the following sources:

- Audio devices, when processing live audio
- Various types of live media streams
- Media files, including audio and video files

IDOL Speech Server cannot directly capture audio from an audio device or handle media streams. To present audio data to IDOL Speech Server, you must either:

- Stream the audio data through a binary port
- Send the audio as a binary data block using either a base64-encoded string or a multipart/form-data POST request
- Provide video or audio files

> **NOTE:**
> IDOL Speech Server supports nearly all audio and video file formats if you set the `FfmpegDirectory` parameter in the `speechserver.cfg` configuration file. (For more information about this parameter, see the *IDOL Speech Server Reference*.) If you do not set this parameter, IDOL Speech Server accepts only 16-bit, linear Pulse Code Modulation (PCM) format WAV

files.

For best results, an audio file should:

- Be captured using an optimally positioned microphone. See Audio Capture, below.
- Be free from clipping. See Clipping, below.
- Contain as little background noise as possible. See Background Noise, below.

An audio stream must also:

- Be decompressed into 16-bit linear, little-endian PCM data. See Audio Compression, below.
- Have a sampling rate of either 8 kHz (8,000 Hz) or 16 kHz (16,000 Hz). See Audio Sampling Rate, on the next page.

The following sections provide more detail about each aspect of audio quality.

Audio Capture

The positioning of the microphone during audio capture has significant influence on audio quality. A microphone positioned very close to the mouth, such as a headset microphone, picks up speech very clearly but also any other noises in the aural cavity, such as lip smacking. A microphone placed a couple of feet away from the speaker picks up speech that has arrived at the microphone from multiple reflected paths, in addition to the direct path. The quality of these two signals is substantially different, leading to varying speech-to-text success rates.

Clipping

Clipping is a form of waveform distortion that occurs when the amplification of a signal is too high to deliver a sample value that can be represented within 16 bits. Clipping has a severe impact on the frequency properties of the audio signal.

To identify clipping in audio files, display them with a waveform display tool.

There is no known method for recovering the original signal from clipped audio for speech-to-text requirements.

Background Noise

Background noise, including music, can significantly impact speech processing.

The SNR (signal-to-noise ratio) measures how much background noise is present in audio data. Typically, recordings with noticeable background noise can have an SNR as low as 10-15 dB. Good-quality recordings have an SNR above 25 dB.

IDOL Speech Server provides an audio preprocessing module to measure various audio quality related metrics, including SNR. For more information, see Preprocess Audio, on page 207.

Audio Compression

For purposes of efficient storage, media files are often stored in a compressed form. Audio compression often results in a small amount of distortion. The compression algorithms, used in codecs

(*codec* stands for compression–decompression), have been developed over many years to minimize the distortion from a perceptual point of view. In general, when the compression rate increases, so does the distortion. But the newer codecs generally offer less distortion for the same compression rate, compared to the older codecs.

Popular audio codecs used in video are:

- MP2, MP3 (multiple sampling rates)
- WMA (multiple sampling rates)

This table summarizes the telephony codecs and compressed data rates (all sampled at 8 kHz).

Codec	Description	Data rate (Kbps)	Data rate (MB per hour)
Original	Linear 16 bit	128	57.6
G.711 ulaw	Ulaw 8 bit	64	28.8
G.711 linear	Linear 8 bit	64	28.8
G.726	ADPCM	32	14.4
G.729	CS-ACELP	8	3.6
GSM 6.10	CELP	13	6.5
DSP Group Truespeech	ADPCM	8.5	3.8
MP3	MDCT	Variable	Variable
WMA	MDCT	Variable	Variable

IDOL Speech Server does not support compressed streamed audio, but can support compressed audio files (you must set the `FFmpegDirectory` configuration parameter).

Audio Sampling Rate

The sampling rate is the rate at which an audio signal is sampled and digitized. In general, the higher the sampling rate, the more information is preserved in the audio signal. This table lists commonly used sampling rates.

Sampling rate (Hz)	Use
8,000	Telephone; adequate for human speech but without sibilance; ‘ess’ sounds like ‘eff’ (/s/, /f/).
11,025	One quarter the sampling rate of audio CDs; used for lower-quality PCM and MPEG audio, and for audio analysis of subwoofer bandpasses.
16,000	Wideband frequency extension over standard telephone narrowband 8,000 Hz. Used in most modern VoIP and VVoIP communication products.
22,050	One half the sampling rate of audio CDs; used for lower-quality PCM and MPEG audio.

Sampling rate (Hz)	Use
32,000	MiniDV digital video camcorder, video tapes with extra channels of audio (for example, DVCAM with 4 Channels of audio), DAT (LP mode), NICAM digital audio, used alongside analog television sound in some countries. Suitable for digitizing FM radio.
44,056	Used by digital audio locked to NTSC color video signals (245 lines by 3 samples by 59.94 fields per second = 29.97 frames per second).
44,100	Audio CD, also most commonly used with MPEG-1 audio (VCD, SVCD, MP3). Much professional audio equipment uses (or is able to select) 44,100 Hz sampling, including mixers, EQs, compressors, reverb, crossovers, recording devices.

Streamed audio must have a sampling rate of either 8,000 Hz (8 kHz) or 16,000 Hz (16 kHz).

If FFmpeg is enabled, IDOL Speech Server accepts audio files with a range of sampling rates. HPE recommends that the sampling rates are at least the sampling rate required by the processing task. Audio with sampling frequencies below this are upsampled, which causes severe quality issues. The minimum sampling frequencies are 8 kHz for processing telephony audio and 16 kHz for processing broadcast audio.

The audio bandwidth can restrict whether you can sample an audio file at 8 kHz or 16 kHz. For more information, see Audio Bandwidth, below.

Audio Bandwidth

The bandwidth is a property of the audio signal and represents the frequency up to which the signal holds information. The bandwidth of a signal is often equal to, but never higher than, half the sampling rate (this principle is known as the Nyquist theorem). For example, an audio stream with a sampling rate of 32 kHz has a maximum bandwidth of 16 kHz. The bandwidth can also be lower than half the sampling rate.

You need to find the bandwidth of an audio stream to decide what sampling rate to choose. For each language, you have a choice of two language packs–one pack can process audio at 8 kHz and the other pack can process audio at 16 kHz. Audio streams sampled at 16 kHz contain more information than streams sampled at 8 kHz, therefore HPE recommends that you choose the higher sampling rate where possible. However, an 8 kHz language pack expects the audio bandwidth to be close to 4 kHz, whereas the 16 kHz language pack expects the audio bandwidth to be close to 8 kHz. If you discover that an audio stream has a bandwidth much lower than expected for the 16 kHz pack, HPE recommends that you downsample it to 8 kHz before processing. The bandwidth of an audio signal can be lower than half the sampling rate for many reasons, including low pass filtering and upsampling (increasing the sampling rate).

You can use waveform analysis tools, such as wavesurfer (http://sourceforge.net/projects/wavesurfer/) and Adobe Audition, to check the bandwidth of an audio stream.

Speech-to-Text

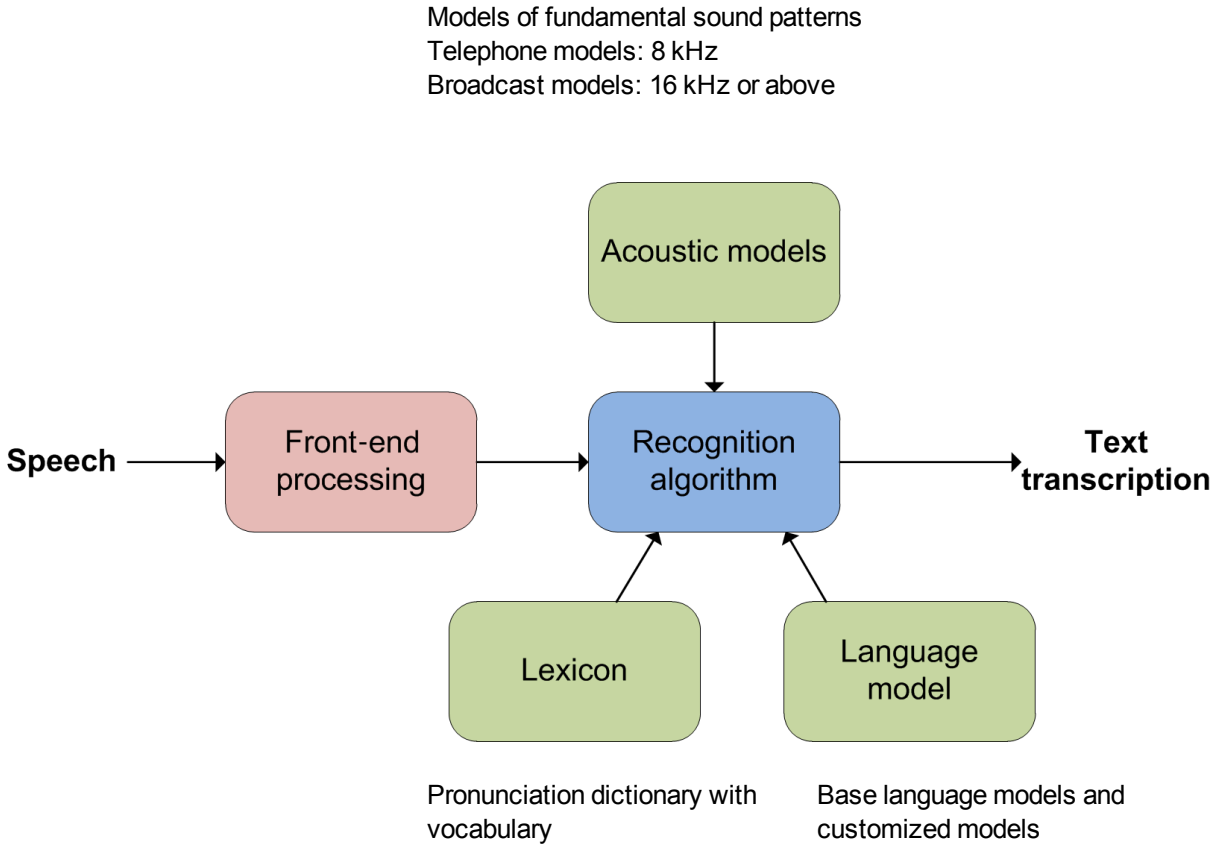
IDOL Speech Server's approach to performing speech-to-text is motivated by an information breakdown point of view of speech. This state-of-the-art approach is used by most leading speech technologists.

- Sentences are composed of words (the basis for the language models).
- Words are composed of phonemes and allophones (for detailed definitions of both terms, see http://www.sil.org/linguistics/glossaryOfLinguisticTerms). This is the basis for the pronunciation dictionaries.
- Each phoneme and allophone is described in terms of frequency spectrum-based features (the basis for the acoustic models).
- Signal processing analysis (performed by the front-end algorithms) converts an audio signal into frequency spectrum-based features.

This approach means that the speech-to-text engine requires a *language pack* that contains:

- A *language model* that contains information about how sentences are composed of words, as well as the word pronunciation dictionary
- An *acoustic model* that describes feature patterns for a complete set of allophones used by the particular natural language.

The following diagram shows the inputs and resources that the speech-to-text engine receives.



Measure Speech-to-Text Success Rates

To measure the success rate of speech-to-text for typical application data, you need:

- A collection of audio files for testing
- For each audio file, a corresponding text file that contains a verbatim record of the words spoken in the audio. These are the 'truth' files.

> **NOTE:**
> To ensure the accuracy of the resulting success rates, the text file must be a completely verbatim record of the audio file.

IDOL Speech Server provides a scoring tool that reports the general word precision and recall rates measured across the test files. The procedures for measuring success rates for speech-to-text are well established and standardized. For more information, see http://www.itl.nist.gov/iad/mig/tools/.

The general steps for measuring success rates are:

1. Run the speech-to-text function on the test data.
2. Normalize the text in the truth files. Normalization is important because it avoids representation mismatch between recognized words and truth text, such as 'one' and '1'.
3. Align the recognition output text against the normalized truth text and calculate precision and recall rates based on aligned entities. For details, see Score Speech Recognition, on page 127.

Improve Speech-to-Text Success Rates

You can improve the speech-to-text success rate by:

- Building a custom language model to better align the vocabulary and word flows.
- Tuning the parameters of the speech-to-text engine and the feature extraction process. For more information, see Tune Parameters, on page 125.

Language Models

IDOL Speech Server uses N-gram statistical language models when performing speech-to-text. An N-gram language model works out the probability of particular sub-sequences of words occurring in a longer word sequence (usually a sentence). The language model is estimated from text that represents the spoken or natural language.

As an example of how the language model works, consider the sentence "The color of the car is red". For an N-gram model with a value of n=3, the sequence is split into every possible sub-sequence that contains up to three words. So for this sentence, the n(=3)-gram fragments are:

"The"

"The color"

"The color of"

“color of the”

“of the car”

“the car is”

“car is red”

“is red.”

The probability of observing the entire sentence is the probability of observing all these individual fragments.

IDOL Speech Server allows you to build new language models. However, the language models included in the language packs are trained with billions of text samples across a wide range of topics, and should be sufficient for most deployments.

> **NOTE:**
> Any language model has a finite vocabulary. As a result, some of the spoken words might not be in the vocabulary for the speech-to-text. For most deployments, the out-of-vocabulary rates are low enough not to cause any concern.

IDOL Speech Server lets you supplement the standard language model with smaller, focused language models that are customized for the specific speech-to-text. HPE recommends that you build a custom language model if you want to improve speech-to-text success rates.

The following scenarios might require a custom language model:

- The vocabulary used in the audio data is very specialized (such as product names), and many of them are not present in the base language model.
- The topic of speech in the audio data is specialized (for example, a lecture on throat cancer).
- You have access to many transcripts that are good representations of typical conversations, such as call center conversations.

Measure the Effectiveness of Language Models

Before you build a custom language model, HPE recommends that you work out whether the base language model provided with the language pack is sufficient for your purpose. There are two ways to measure effectiveness:

- Vocabulary coverage–whether most or all the important words that you want to be recognized in the speech-to-text are included in the vocabulary.
- Perplexity–this relates to the estimated branching factor for the language in speech. Different audio data require models with different levels of perplexity; for example, call center conversations often closely follow a set pattern and therefore require a lower perplexity value than broadcast audio, which usually covers a wide variety of topics. The perplexity value allows you to decide if the language model is suitable for typical conversations in the audio data.

To measure the vocabulary coverage of a language model

1. Prepare a list of the words that you want to check are present in the language model.
2. Normalize the list (see Run Text Normalization, on page 142).
3. Check that the words are in the vocabulary (see Look Up Vocabulary, on page 132).

To measure the perplexity of a language model

1. Prepare text transcript files or text files that contain sentences from typical conversations that you expect in the audio data.
2. Normalize the text files (see Run Text Normalization, on page 142).
3. Use the IDOL Speech Server tool for calculating perplexity, which also reports out-of-vocabulary words in the text (see Calculate Perplexity, on page 130).

Perplexity values around or below 100 are typical and acceptable for call center-like conversations. Aim for this value when you process telephone data.

Perplexity values around or below 250 are typical and acceptable for broad coverage content, such as news. Aim for this value when you process such audio data.

Source Text for Building Custom Language Models

To create an effective custom language model, you must use sample text that is strongly representative of the speech data that you want to process. For example, you could source text from:

- Transcripts of call conversations or lectures
- Literature that describes the product or company
- Slides used in delivering the lecture
- Articles written by the same speaker that delivered the lecture
- A web article on the particular topic
- Company websites that contain natural language descriptions (rather than images or advertisements)

You might have to clean up the text to meet the requirements for presentation to IDOL Speech Server:

- Ensure that it is encoded in UTF-8
- Avoid duplicated text
- Ensure that sentence breaks are present in appropriate places
- Remove any material that does not represent natural language, such as HTML tags

Build Custom Language Models

You can build custom language models in IDOL Speech Server (see Create Custom Language Models, on page 129). In general, the more text that you use to build a language model, the better the performance of that model. However, the use of inappropriate text can diminish performance.

After you build a custom language model, you can check its suitability and performance. For instructions, see Measure the Effectiveness of Language Models, on the previous page.

Acoustic Models

Audio quality is affected by many factors, including the properties and position of the audio capture device, channel distortions (in the case of phone calls), and speaker population factors, such as dialect, accent, and timbre. When the recognized words poorly resemble the spoken words, it usually indicates that the acoustic models are a poor match for the audio data being processed.

> **NOTE:**
> In the 10.7 release of IDOL Speech Server, you could use acoustic adaptation to adapt the Gaussian Mixture Model (GMM) acoustic models to match an audio domain. To improve speech-to-text accuracy, IDOL Speech Server now includes Deep Neural Network (DNN) acoustic modeling. DNNs are not currently adaptable, but typically outperform even adapted GMM acoustic models. As a result, HPE does not generally recommend acoustic adaptation. However, in certain scenarios (for example, in cases where the language packs do not have a DNN, or where you are working with a very specific domain and believe that DNN recognition could be improved upon), acoustic adaptation can still be useful. Use the following instructions to perform this process.

IDOL Speech Server provides tools for adapting acoustic models. However, adapting models is not easy and must be done with great care.

To adapt acoustic models, you need:

- Sample audio data recorded under the same conditions as the audio that is to be processed
- For each recorded audio data file, a verbatim transcription text file

HPE recommends that you use 5-20 hours of audio data to adapt an acoustic model, although there is no upper limit to the amount of data you can use.

You must prepare the transcript text according to the transcription guidelines (see Audio Transcript Requirements, on page 281).

The underlying algorithm used in adapting acoustic models is iterative; as a result, adaptation requires multiple processing passes. The steps involved are:

1. Create front-end feature vector data files.
2. Normalize and prepare the transcript text files. You need to perform steps 1 and 2 only once.
3. Run the adaptation algorithm and save the crunched data.
4. Finalize the new adapted acoustic model.
5. Measure the speech-to-text success rates (see Measure Speech-to-Text Success Rates, on page 26).
6. Repeat steps 3 and 4 as required, depending on the speech-to-text success rates.

Adapting the acoustic model can also improve other operations that involve acoustic models, such as phonetic phrase search.

Phonetic Phrase Search

Phonetic phrase search is a fast and approximate method for finding sections in audio data that sound similar to the phrase being searched for. Typically, phonetic phrase searches can run many times faster than performing speech-to-text and then searching the transcript.

IDOL Speech Server first processes audio data to extract phonemes, and produces a time track charting the position of phonemes in the audio file. The time tracks for different phonemes overlap; as a result, each time position can appear in more than one phoneme time track.

IDOL Speech Server can then search through the time tracks to match selected words or phrases. It identifies possible time positions for the item being searched, together with a hit score.

Lowering the hit score threshold increases the number of identification hits but also increases the number of false positive results, leading to an increased recall rate but decreased precision. Raising the hit score threshold has the opposite effect. You can use the hit scores to trade off false positives against false negatives.

You can also perform the search in two stages, where an initial action processes the audio file to create the time tracks and a second, separate action searches these time tracks. The details of these are covered in Phonetic Phrase Search, on page 149.

This table compares phonetic phrase search and speech-to-text based search.

Phonetic search	Speech-to-text and search
Search usually occurs in two stages: the first stage creates the phoneme time track, the second stage searches the time track for phoneme sequences that resemble the specified phrase or phrases.	Search usually occurs in two stages: the first stage is speech-to-text, the second stage searches the transcript.
The first stage runs many times faster than speech-to-text. The second stage runs at about 200-600 times faster than real time.	Speech-to-text runs at real time, or a few times faster than real time. The second stage is instantaneous.
The first stage is regarded as the 'ingestion' stage, and occurs only once for each set of audio data. It is independent of the search stage.	The first stage is regarded as the 'ingestion' stage, and occurs only once for each set of audio data. It is independent of the search stage.
Can generate many false positive results, meaning that the search returns phoneme sequences that do not match the searched-for phrase. However, you can alter the hit score threshold to alter the rate of false positives and false negatives, to an extent.	Can generate many false negative results, meaning that the search can miss some instances of the searched-for phrase that are present in the data. There is little scope for altering this.
The non-specificity of phonemes contributes to the false positives obtained. The search does not use linguistic information.	The language model used in speech-to-text means that linguistic information is used in the search process.
Generally chosen when there is a large quantity of data to search, or when audio quality is too poor to obtain good speech-to-text search results.	Generally preferred when audio quality is reasonable and there is hardware available to process the data.

Phonetic phrase search is language dependent and requires an appropriate language pack.

Adapting the acoustic model (see Acoustic Models, on page 28) can improve phonetic phrase search performance.

Speaker Identification

IDOL Speech Server can perform speaker identification, as well as providing some processing toward speaker diarization. It does not perform speaker verification.

The speaker identification system also includes a rejection methodology for *open-set* identification, in the form of score thresholds for each speaker. Only when a template scores above this threshold for a

section can a hit be considered genuine. These thresholds use both speech data from the speaker (positive examples), as well as from a selection of a range of 'other' speakers (negative examples). HPE recommends that the amount of 'other' speaker training data totals at least 60 minutes.

Speaker identification involves:

- Creating a template for each speaker to identify
- Estimating score thresholds for each template based on positive and negative examples (this step is required for open-set speaker identification, but is optional for closed-set identification)
- Using one or more templates to identify known speakers in an audio recording

The base speaker ID pack that is included in the IDOL Speech Server installation contains a Universal Background Model (UBM). IDOL Speech Server uses the UBM file as the base to train speaker templates. Because the template models are trained from an established base model, the template is reliable even when training data is sparse. The UBM is also used in calculating the speaker scores, leading to more consistent score ranges.

Although you can train a speaker template from relatively little data, HPE recommends that you train with at least five minutes of sample audio for each speaker, and more if available. Use good quality audio samples that contain only the speaker's voice and no significant background noise. The spoken content can contain any vocabulary.

HPE recommends that you create no more than 100 speaker templates. Although there is no fixed limit to the number of speakers, larger speaker sets are likely to lead to an increase in classification errors.

Speaker identification is text independent, meaning that it can identify speakers speaking any content.

In some aspects, speaker identification is opposite in nature to speaker independent speech-to-text. In the former, IDOL Speech Server tries to identify differences in speaker voices. In the latter, IDOL Speech Server attempts to remove individual speaker differences so that the speech-to-text is robust across multiple speakers.

Speaker identification is impacted by:

- Background noise
- Audio distortion, weak pickup, and echo
- Speech overlap from multiple speakers talking at the same time

Spoken Language Identification

IDOL Speech Server can analyze audio data for numerous languages across the world. IDOL Speech Server contains a universal phoneme decoder that detects a wide range of phonemes across multiple languages. After IDOL Speech Server extracts phoneme information from an audio file, it performs a statistical analysis on the general distribution of the phonemes to estimate the identity of the spoken language.

IDOL Speech Server requires a language classifier file for each language that you want to identify. IDOL Speech Server provides a classifier pack that covers the following 20 languages for broadband (16 kHz) audio. To identify any other languages or dialects, or to process telephony (8 kHz) audio, you must create your own classifiers.

> **NOTE:**
> Telephony data quality varies hugely, therefore IDOL Speech Server requires classifiers that

are trained on representative data.

Arabic	Hebrew	Portuguese
Danish	Italian	Romanian
Dutch	Japanese	Russian
English	Korean	Slovak
French	Mandarin	Spanish
German	Persian	Swedish
Greek	Polish	

Although you can perform language identification tasks out of the box, HPE recommends that you optimize the classifiers on your own audio data. For instructions, see Optimize the Language Identification Set, on page 179.

For best results, train the language identification system to detect only the list of languages expected in the audio file. To train the language identification system, you need samples of typical audio that contain only the labeled spoken language. HPE recommends that you remove any non-speech sections in the audio, such as music. However, there is no need to remove sections of silence.

NOTE:
You do not require audio transcriptions to train the language identification system.

The training process involves performing a phoneme analysis on the training data and then using the analysis results to build a statistical model.

To train the language identifier

1. Decide which languages you want to detect.
2. Obtain the audio data. You need:
   - A minimum of one hour of audio for each language. For better results, HPE recommends that you use five hours of audio. The audio must be representative of the data you intend to process. For each language, you also require around 15 minutes of audio to optimize the combined language classifier.
   - Data from other languages, if the audio is likely to contain speech in other languages
3. Train the individual language identification models (see Create Your Own Language Classifiers, on page 175).
4. If required, create an 'other' language identification model that uses data from the 'other' languages.
5. Create the combined language identification system, putting together all the individual language identifiers.

Spoken language identification is affected by the same factors that affect speech-to-text and other speech processing.

Spoken language identification is text independent.

The data processing flow to perform language identification is:

1. Process the audio file to identify the phoneme sequences from the universal phoneme set.
2. Use that information to identify the spoken language.

Audio Fingerprint Identification

Audio fingerprint identification offers a robust method for identifying complete or partial portions of audio that are exact or slightly degraded copies of a known audio track.

Each master audio track is processed to create a fingerprint consisting of a 'significant' audio event in the audio track. These audio events are generally expected to be detectable in true or slightly degraded copies of the master audio track. IDOL Speech Server provides tools that you can use to create a database of audio fingerprints for each audio track. After the database is created, IDOL Speech Server is able to identify complete or partial samples of the audio tracks.

Transcript Alignment

When a text transcript corresponding to an audio file is available, transcript alignment can match the two files and generate the time positions of each word in a final transcript. The text transcript does not have to match the audio file exactly.

Transcript alignment involves the following steps:

1. Performing speech-to-text on the audio data to be aligned.
2. Normalizing the transcript text.
3. Performing transcript alignment.

The transcript alignment module compares the speech-to-text output and the normalized transcript text to generate the alignment and produce the time codes.

You might need to configure the speech-to-text for best results. HPE recommends that you build a 'transcript' custom language model to use in the speech-to-text (see Transcript Language Models, below).

Transcript alignment uses the same operation as the process of scoring speech-to-text results. The alignment function also reports the precision and recall rates with respect to the alignment. Measuring transcription success rates in this way is consistent with the industry standard practice of evaluating speech-to-text systems.

Transcript Language Models

Transcript language models are language models that are built using the exact or approximate audio transcripts that are to be processed, and therefore contain all the vocabulary used in the audio. Furthermore, the word probabilities are also heavily biased to produce the same word sequences present in the transcripts. Often the result is a very accurate speech-to-text output.

Transcript language models are useful for:

- Transcript alignment
- Confirmation that the transcript text matches the audio well, and a measure of the extent of this match.

Chapter 3: Install IDOL Speech Server

This section describes the software and hardware requirements and installation instructions for IDOL Speech Server.

- System Requirements 35
- Install IDOL Speech Server 37
- Licenses 43
- Supported Resources 47

System Requirements

This section describes the software and hardware requirements to run IDOL Speech Server.

Supported Operating Systems

IDOL Speech Server runs on the following operating systems.

Windows

- Windows Server 2012
- Windows Server 2008
- Windows 7

Linux

- Linux kernel
  - Red Hat Enterprise: 2.6.9-11.EL or later
  - SUSE: 2.6.16.60-0.54.5 or later

Antivirus Recommendations

If you are running antivirus software on the IDOL Speech Server host machine, you must ensure for performance reasons that it does not monitor the IDOL Speech Server directories.

Some advanced antivirus software can scan the network and might block some IDOL Speech Server traffic, which can cause errors. Where possible, exempt the IDOL Speech Server processes from this kind of network traffic analysis.

Recommended Hardware Specifications

HPE recommends that you use the following machine specifications for a machine running a single IDOL Speech Server:

- 2 GHz single or multicore processor, 64 bit
- 150 MB disk space for software installation, plus space for each language pack (the size of each language pack can vary from 12 MB to 450 MB)
- Memory requirements depend on the number of language packs, the number of simultaneous decode tasks, and the operating system.
  - 600 MB for each language pack (shared across multiple channels).
  - 200 MB for each input.

HPE recommends that you assign each speech-to-text task to a single, 2 GHz core.

Memory Requirements

IDOL Speech Server has the following memory requirements:

- Each language pack requires approximately 500 MB of RAM to load.

  > **NOTE:**
  > Broadband and telephony versions of the same language pack count as separate language packs internally. Similarly, if you load the same language pack without a DNN, with a small DNN (for example, for real-time processing), and with the standard DNN, this counts as three separate language packs.

- Each IDOL Speech Server task requires additional memory. HPE recommends the following approximate values. (If you run multiple tasks simultaneously, the amount of memory that is required increases; for example, to run two concurrent tasks that each use 250 MB of memory, 500 MB of memory is required.)

Task type	Memory
Speech-to-text	150 MB[1]
Language model building	300 MB[2]
Acoustic model adaptation	750 MB
Speaker identification	100 MB
Transcript alignment	250 MB
Language identification	250 MB
Language identification training	300 MB
Language identification optimization	10 MB

To ensure smooth operation, the `speechserver.cfg` configuration file allows you to limit the number of concurrent actions on the server. You must also be careful when you set the maximum number of language models that the server can load.

Other Requirements

IDOL Speech Server has the following other general requirements that you might need to take into account when setting up your system:

- For speech-to-text in live or relative mode, you can use Dynamic Neural Network (DNN) acoustic modelling only if your DNN files are smaller than a certain size. In addition, you must use Intel (or compatible) processors that support SIMD extensions SSSE3 and SSE4.1. For more information, see Use Live Mode for Streaming, on page 204.

[1]Stereo speech-to-text uses two `stt` modules and therefore uses twice the memory of a standard speech-to-text task.

[2]If the training texts contain unusually large vocabularies, the language model building tasks might require more memory.

Install IDOL Speech Server

To install IDOL Speech Server, use either the IDOL Server installer, which contains IDOL Speech Server as a component, or the IDOL Speech Server standalone zip package.

Install IDOL Speech Server on Windows

Use the following procedure to install IDOL Speech Server on Microsoft Windows operating systems, by using the IDOL Server installer.

The IDOL Server installer provides the major IDOL components. It also includes License Server, which IDOL Speech Server requires to run.

To install IDOL Speech Server

1. Double-click the appropriate installer package:

   `IDOLServer_`*`VersionNumber`*`_`*`Platform`*`.exe`

   where:

`VersionNumber`	is the product version.
`Platform`	is your software platform.

   The Setup dialog box opens.
2. Click **Next**.

   The License Agreement dialog box opens.
3. Read the license agreement. Select **I accept the agreement**, and then click **Next**.

   The Installation Directory dialog box opens.
4. Specify the directory to install IDOL Speech Server (and optionally other components such as

License Server) in. By default, the system installs on `C:\HewlettPackardEnterprise\IDOLServer-VersionNumber`. Click to choose another location. Click **Next**.

The Installation Mode dialog box opens.

5. Select **Custom**, and then click **Next**.

   The License Server dialog box opens. Choose whether you have an existing License Server.

   - To use an existing License Server, click **Yes**, and then click **Next**. Specify the host and ACI port of your License Server, and then click **Next**.
   - To install a new instance of License Server, click **No**, and then click **Next**. Specify the ports that you want License Server to listen on, and then type the path or click and navigate to the location of your IDOL license key file (`licensekey.dat`), which you obtained when you purchased IDOL Speech Server. Click **Next**.

   The Component Selection dialog box opens.

6. Click **Next**.
7. Select the check boxes for the components that you want to install, and specify the port information for each component, or leave the fields blank to accept the default port settings.

   For IDOL Speech Server, you must specify the following information:

ACI Port	The port that client machines use to send ACI actions to IDOL Speech Server.
Service Port	The port that IDOL Speech Server uses for License Server communication. This port must not be used by any other service.
Binary Data Port	The port to use for streaming audio data to IDOL Speech Server.

   Click **Next** or **Back** to move between components.

8. After you have specified your settings, the Summary dialog box opens. Verify the settings you made and click **Next**.

   The Ready to Install dialog box opens.

9. Click **Next**.

   The Installing dialog box opens, indicating the progress of the installation. If you want to end the installation process, click **Cancel**.

10. After installation is complete, click **Finish** to close the installation wizard.

Install an IDOL Component as a Service on Windows

On Microsoft Windows operating systems, you can install any IDOL component as a Windows service. Installing a component as a Windows service makes it easy to start and stop the component, and you can configure a component to start automatically when you start Windows.

Use the following procedure to install IDOL Speech Server as a Windows service from a command line.

To install a component as a Windows service

1. Open a command prompt with administrative privileges (right-click the icon and select **Run as administrator**).
2. Navigate to the directory that contains the component that you want to install as a service.
3. Send the following command:

   ```
   Component.exe -install
   ```

   where *Component*.exe is the executable file of the component that you want to install as a service.

   The -install command has the following optional arguments:

-start {[auto] | [manual] | [disable]}	The startup mode for the component. **Auto** means that Windows services automatically starts the component. **Manual** means that you must start the service manually. **Disable** means that you cannot start the service. The default option is **Auto**.
-username *UserName*	The user name that the service runs under. By default, it uses a local system account.
-password *Password*	The password for the service user.
-servicename *ServiceName*	The name to use for the service. If your service name contains spaces, use quotation marks (") around the name. By default, it uses the executable name.
-displayname *DisplayName*	The name to display for the service in the Windows services manager. If your display name contains spaces, use quotation marks (") around the name. By default, it uses the service name.
-depend *Dependency1* [,*Dependency2* ...]	A comma-separated list of the names of Windows services that Windows must start before the new service. For example, you might want to add the License Server as a dependency.

   For example:

   ```
   Component.exe -install -servicename ServiceName -displayname "Component Display
   Name" -depend LicenseServer
   ```

After you have installed the service, you can start and stop the service from the Windows Services manager.

When you no longer require a service, you can uninstall it again.

To uninstall an IDOL Windows Service

1. Open a command prompt.
2. Navigate to the directory that contains the component service that you want to uninstall.
3. Send the following command:

   ```
   Component.exe -uninstall
   ```

where *Component*.exe is the executable file of the component service that you want to uninstall.

If you did not use the default service name when you installed the component, you must also add the -servicename argument. For example:

Component.exe -uninstall -servicename *ServiceName*

Install IDOL Speech Server on UNIX

Use the following procedure to install IDOL Speech Server in text mode on UNIX platforms.

To install IDOL Speech Server on UNIX

1. Open a terminal in the directory in which you have placed the installer, and enter the following command:

   ./IDOLServer_*VersionNumber_Platform*.exe --mode text

   where:

VersionNumber	is the product version
Platform	is the name of your UNIX platform

   > **NOTE:**
   > Ensure that you have execute permission for the installer file.

   The console installer starts and displays the Welcome screen.
2. Read the information and then press the Enter key.

   The license information is displayed.
3. Read the license information, pressing Enter to continue through the text. After you finish reading the text, type **Y** to accept the license terms.
4. Type the path to the location where you want to install the servers, or press Enter to accept the default path.

   The Installation Mode screen is displayed.
5. Press 2 to select the Custom installation mode.

   The License Server screen opens. Choose whether you have an existing License Server.

   - To use an existing License Server, type **Y**. Specify the host and port details for your License Server (or press Enter to accept the defaults), and then press Enter. Go to Step 7.
   - To install a new instance of License Server, type **N**.
6. If you want to install a new License Server, provide information for the ports that the License Server uses.

   a. Type the value for the ACI Port and press Enter (or press Enter to accept the default value).

      ACI Port The port that client machines use to send ACI actions to the License Server.

   b. Type the value for the Service Port and press Enter (or press Enter to accept the default value).

Service Port	The port by which you send service actions to the License Server. This port must not be used by any other service.

c. Type the location of your IDOL license key file (`licensekey.dat`), which you obtained when you purchased IDOL Speech Server. Press `Enter`.

7. The Component Selection screen is displayed. Press `Enter`. When prompted, type **`Y`** for the components that you want to install. Specify the port information for each component, and then press `Enter`. Alternatively, leave the fields blank and press `Enter` to accept the default port settings.

   For IDOL Speech Server, you must specify the following information:

ACI Port	The port that client machines use to send ACI actions to IDOL Speech Server.
Service Port	The port that IDOL Speech Server uses for License Server communication. This port must not be used by any other service.
Binary Data Port	The port to use for streaming audio data to IDOL Speech Server.

> **NOTE:**
> These ports must not be used by any other service.

   The Init Scripts screen is displayed.

8. Type the user that the server should run as, and then press `Enter`.

> **NOTE:**
> The installer does not create this user. It must exist already.

9. Type the group that the server should run under, and then press `Enter`.

> **NOTE:**
> If you do not want to generate init scripts for installed components, you can simply press `Enter` to move to the next stage of the installation process without specifying a user or group.

   The Summary screen is displayed.

10. Verify the settings that you made, then press `Enter` to begin installation.

    The Installing screen is displayed.

    This screen indicates the progress of the installation process.

    The Installation Complete screen is displayed.

11. Press `Enter` to finish the installation.

Install an IDOL Component as a Service on Linux

On Linux operating systems, you can configure a component as a service to allow you to easily start and stop it. You can also configure the service to run when the machine boots. The following procedures describe how to install IDOL Speech Server as a service on Linux.

> **NOTE:**
> To use these procedures, you must have `root` permissions.

> **NOTE:**
> When you install IDOL Speech Server on Linux, the installer prompts you to supply a user name to use to run the server. The installer populates the init scripts, but it does not create the user in your system (the user must already exist).

The procedure that you must use depends on the operating system and boot system type.

- For Linux operating system versions that use systemd (including CentOS 7, and Ubuntu version 15.04 and later), see Install a Component as a Service for a systemd Boot System, below.
- For Linux operating system versions that use System V, see Install a Component as a Service for a System V Boot System, on the next page.

Install a Component as a Service for a systemd Boot System

> **NOTE:**
> If your setup has an externally mounted drive that IDOL Speech Server uses, you might need to modify the init script. The installed init script contains examples for an NFS mount requirement.

To install an IDOL component as a service

1. Run the appropriate command for your Linux operating system environment to copy the init scripts to your `init.d` directory.
   - Red Hat Enterprise Linux (and CentOS)

     ```
     cp IDOLInstallDir/scripts/init/systemd/componentname
     /etc/systemd/system/componentname.service
     ```

   - Debian (including Ubuntu):

     ```
     cp IDOLInstallDir/scripts/init/systemd/componentname
     /lib/systemd/system/componentname.service
     ```

   where *componentname* is the name of the init script that you want to use, which is the name of the component executable (without the file extension).

   For other Linux environments, refer to the operating system documentation.
2. Run the following commands to set the appropriate access, owner, and group permissions for the component:
   - Red Hat Enterprise Linux (and CentOS)

     ```
     chmod 755 /etc/systemd/system/componentname
     chown root /etc/systemd/system/componentname
     chgrp root /etc/systemd/system/componentname
     ```

   - Debian (including Ubuntu):

     ```
     chmod 755 /lib/systemd/system/componentname
     chown root /lib/systemd/system/componentname
     chgrp root /lib/systemd/system/componentname
     ```

where *componentname* is the name of the component executable that you want to run (without the file extension).

For other Linux environments, refer to the operating system documentation.

3. (Optional) If you want to start the component when the machine boots, run the following command:

```
systemctl enable componentname
```

Install a Component as a Service for a System V Boot System

To install an IDOL component as a service

1. Run the following command to copy the init scripts to your `init.d` directory.

```
cp IDOLInstallDir/scripts/init/systemv/componentname /etc/init.d/
```

where *componentname* is the name of the init script that you want to use, which is the name of the component executable (without the file extension).

2. Run the following commands to set the appropriate access, owner, and group permissions for the component:

```
chmod 755 /etc/init.d/componentname
chown root /etc/init.d/componentname
chgrp root /etc/init.d/componentname
```

3. (Optional) If you want to start the component when the machine boots, run the appropriate command for your Linux operating system environment:
   - Red Hat Enterprise Linux (and CentOS):

     ```
     chkconfig --add componentname
     chkconfig componentname on
     ```

   - Debian (including Ubuntu):

     ```
     update-rc.d componentname defaults
     ```

   For other Linux environments, refer to the operating system documentation.

Licenses

To use IDOL solutions, you must have a running License Server, and a valid license key file for the products that you want to use. Contact HPE Big Data Support to request a license file for your installation.

License Server controls the IDOL licenses, and assigns them to running components. License Server must run on a machine with a static, known IP address, MAC address, or host name. The license key file is tied to the IP address and ACI port of your License Server and cannot be transferred between machines. For more information about installing License Server and managing licenses, see the *License Server Administration Guide*.

When you start IDOL Speech Server, it requests a license from the configured License Server. You must configure the host and port of your License Server in the IDOL Speech Server configuration file.

You can revoke the license from a product at any time, for example, if you want to change the client IP address or reallocate the license.

CAUTION:
Taking any of the following actions causes the licensed module to become inoperable.

You **must not**:

- Change the IP address of the machine on which a licensed module runs (if you use an IP address to lock your license).
- Change the service port of a module without first revoking the license.
- Replace the network card of a client without first revoking the license.
- Remove the contents of the `license` and `uid` directories.

All modules produce a `license.log` and a `service.log` file. If a product fails to start, check the contents of these files for common license errors. See Troubleshoot License Errors, on page 46.

Display License Information

You can verify which modules you have licensed either by using the IDOL Admin interface, or by sending the `LicenseInfo` action from a web browser.

To display license information in IDOL Admin

- In the **Control** menu of the IDOL Admin interface for your License Server, click **Licenses**.

  The **Summary** tab displays summary information for each licensed component, including:
  - The component name.
  - The number of seats that the component is using.
  - The total number of available seats for the component.
  - (Content component only) The number of documents that are currently used across all instances of the component.
  - (Content component only) The maximum number of documents that you can have across all instances of the component.

  The **Seats** tab displays details of individual licensed seats, and allows you to revoke licenses.

To display license information by sending the LicenseInfo action

- Send the following action from a web browser to the running License Server.

`http://`*`LicenseServerHost`*`:`*`Port`*`/action=LicenseInfo`

where:

`LicenseServerHost`	is the IP address of the machine where License Server resides.
`Port`	is the ACI port of License Server (specified by the `Port` parameter in the `[Server]` section of the License Server configuration file).

In response, License Server returns the requested license information. This example describes a license to run four instances of IDOL Server.

```
<?xml version="1.0" encoding="UTF-8" ?>
<autnresponse xmlns:autn="http://schemas.autonomy.com/aci/">
  <action>LICENSEINFO</action>
  <response>SUCCESS</response>
  <responsedata>
    <LicenseDiSH>
      <LICENSEINFO>
        <autn:Product>
          <autn:ProductType>IDOLSERVER</autn:ProductType>
          <autn:TotalSeats>4</autn:TotalSeats>
          <autn:SeatsInUse>0</autn:SeatsInUse>
        </autn:Product>
      </LICENSEINFO>
    </LicenseDiSH>
  </responsedata>
</autnresponse>
```

Configure the License Server Host and Port

IDOL Speech Server is licensed through License Server. In the IDOL Speech Server configuration file, specify the information required to connect to the License Server.

To specify the license server host and port

1. Open your configuration file in a text editor.
2. In the `[License]` section, modify the following parameters to point to your License Server.

`LicenseServerHost`	The host name or IP address of your License Server.
`LicenseServerACIPort`	The ACI port of your License Server.

   For example:

   ```
   [License]
   LicenseServerHost=licenses
   LicenseServerACIPort=20000
   ```

3. Save and close the configuration file.

Revoke a Client License

After you set up licensing, you can revoke licenses at any time, for example, if you want to change the client configuration or reallocate the license. The following procedure revokes the license from a component.

> **NOTE:**
> If you cannot contact the client (for example, because the machine is inaccessible), you can free the license for reallocation by sending an `AdminRevokeClient` action to the License Server. For more information, see the *License Server Administration Guide*.

To revoke a license

1. Stop the IDOL solution that uses the license.
2. At the command prompt, run the following command:

   ```
   InstallDir/ExecutableName[.exe] -revokelicense -configfile cfgFilename
   ```

This command returns the license to the License Server.

You can send the `LicenseInfo` action from a web browser to the running License Server to check for free licenses. In this sample output from the action, one IDOL Server license is available for allocation to a client.

```
<autn:Product>
   <autn:ProductType>IDOLSERVER</autn:ProductType>
   <autn:Client>
      <autn:IP>192.123.51.23</autn:IP>
      <autn:ServicePort>1823</autn:ServicePort>
      <autn:IssueDate>1063192283</autn:IssueDate>
      <autn:IssueDateText>10/09/2003 12:11:23</autn:IssueDateText>
   </autn:Client>
      <autn:TotalSeats>2</autn:TotalSeats>
      <autn:SeatsInUse>1</autn:SeatsInUse>
</autn:Product>
```

Troubleshoot License Errors

The table contains explanations for typical licensing-related error messages.

License-related error messages

Error message	Explanation
Error: Failed to update license from the license server. Your license cache details do not match the current service configuration. Shutting the service down.	The configuration of the service has been altered. Verify that the service port and IP address have not changed since the service started.
Error: License for *ProductName* is invalid. Exiting.	The license returned from the License Server is invalid. Ensure that the license has not expired.
Error: Failed to connect to license server using cached licensed details.	Cannot communicate with the License Server. The product still runs for a limited period; however, you should verify whether your License Server is still available.
Error: Failed to connect to license server. Error code is SERVICE: *ErrorCode*	Failed to retrieve a license from the License Server or from the backup cache. Ensure that your License Server can be contacted.
Error: Failed to decrypt license keys. Please contact Autonomy support. Error code is SERVICE:*ErrorCode*	Provide HPE Big Data Support with the exact error message and your license file.

License-related error messages, continued

Error message	Explanation
Error: Failed to update the license from the license server. Shutting down	Failed to retrieve a license from the License Server or from the backup cache. Ensure that your License Server can be contacted.
Error: Your license keys are invalid. Please contact Autonomy support. Error code is SERVICE:*ErrorCode*	Your license keys appear to be out of sync. Provide HPE Big Data Support with the exact error message and your license file.
Failed to revoke license: No license to revoke from server.	The License Server cannot find a license to revoke.
Failed to revoke license from server *LicenseServer Host*:*LicenseServerPort*. Error code is *ErrorCode*	Failed to revoke a license from the License Server. Provide HPE Big Data Support with the exact error message.
Failed to revoke license from server. An instance of this application is already running. Please stop the other instance first.	You cannot revoke a license from a running service. Stop the service and try again.
Failed to revoke license. Error code is SERVICE:*ErrorCode*	Failed to revoke a license from the License Server. Provide HPE Big Data Support with the exact error message.
Your license keys are invalid. Please contact Autonomy Support. Error code is ACISERVER:*ErrorCode*	Failed to retrieve a license from the License Server. Provide HPE Big Data Support with the exact error message and your license file.
Your product ID does not match the generated ID.	Your installation appears to be out of sync. Forcibly revoke the license from the License Server and rename the `license` and `uid` directories.
Your product ID does not match this configuration.	The service port for the module or the IP address for the machine appears to have changed. Check your configuration file.

Supported Resources

IDOL Speech Server requires supporting files, which are available in packs for different languages and operations. After you complete the installation process, download the relevant packs from the HPE Download Center and save them to the language pack root directory.

> **NOTE:**
> The `LangPackDir` parameter in the `[Paths]` section of the IDOL Speech Server configuration file sets the location of the language pack root directory. If you save the resource packs in a different location to the directory that you specified during installation, you must update this parameter. Alternatively, you can specify the location of individual packs as the value of the

`PackDir` parameter in each resource configuration section.

You must also configure some of the packs in the `speechserver-tasks.cfg` configuration file. For more information, see Configure Language Packs, on page 62.

IDOL Speech Server resources are classified into types.

`fpdb`	Audio fingerprint databases. These are not downloadable files but are files that are created by several audio fingerprinting tasks.
`lang`	Basic language packs, which are used for speech-to-text and text normalization.
`langvt`	Language packs that are configured for processes including phonetic phrase match and audio security.
`sidbase`	Base speaker identification packs, which are used for speaker identification tasks.

Resource Packs

The following resource packs are available for IDOL Speech Server operations.

Pack	Description	Configuration requirement
Audio Security Pack	Contains the `EVENTS` pack required by the `audiosec` module.	Configure in the `[Resources]` section of the `speechserver-tasks.cfg` file. In individual tasks that use the `audiosec` module, specify the `EVENTS` pack as the value of the `Model` configuration parameter.
Language Pack (various languages)	Contain the files necessary for IDOL Speech Server to process various languages. They are available for several languages, as well as regional variations of the same language (for example, U.S. English and British English). Available language packs, on the next page lists the available language packs. The language packs also contain files that are adapted for phonetic phrase search.	Configure in the `[Resources]` section of the `speechserver-tasks.cfg` file. The configuration file must contain separate entries for the standard and phonetic phrase search versions of a language pack. The standard version is a `lang` resource type; the phonetic phrase match version is a `langvt` resource type and adds a `-pm` suffix to the pack name. For example, `ENUS` is the standard language pack, and `langvt:ENUS-pm` is the phonetic phrase search version.
Telephony Pack (various languages)	Contain the same files as the standard language packs, except that these are adapted for processing 8 kHz audio instead of 16 kHz.	Configure in the `[Resources]` section of the `speechserver-tasks.cfg` file. The configuration file must contain separate entries for the standard and phonetic phrase search versions of a language pack. The standard version is a `lang` resource type; the phonetic phrase match version is a `langvt`

Pack	Description	Configuration requirement
		resource type and adds a `-pm` suffix to the pack name. For example, `ENUS-tel` is the standard language pack, and `langvt:ENUS-tel-pm` is the phonetic phrase search version.
Language ID Pack	Contains the base classifier pack, which provides language classifiers for 20 languages (for a list of included languages, see Create Your Own Language Classifiers, on page 175). Also contains the SYLS language pack required to create language identification feature files from 16 kHz audio data.	For the base classifier pack, set the `LangList` parameter in the `langid` and `lidoptimizer` modules to a subset of languages to use from the pack. For the SYLS language pack: • Configure in the `[Resources]` section of the `speechserver-tasks.cfg` file. • In individual tasks that use the `lidfeature` module, specify the pack as the value of the `Lang` configuration parameter.
Language ID Telephony	Contains the SYLS language pack required to create language identification feature files from 8 kHz audio data.	Configure in the `[Resources]` section. In individual tasks that use the `lidfeature` module, specify the pack as the value of the `Lang` configuration parameter.
Speaker ID Pack	Contains 8 kHz and 16 kHz versions of base `.atf`, `.ian`, `.sig` and `.ast` files. The `.atf` file is used as a base template when training, optimizing, and packaging new audio template files (such as speaker templates). The `.ian` file is used during audio feature frame normalization. The `.ast` and `.sig` files are used during speaker segmentation.	Configure in the `[Resources]` section. In individual tasks that use the `normalizer`, `speakerid`, `audiotemplatetrain`, `audiotemplatedevel`, `audiotemplatescore`, or `audiotemplateedit` modules, specify the speaker ID pack as the value of the `SpkIdBasePack` configuration parameter if required.

Available language packs

Language	Language pack
Arabic	• Gulf Dialect: `ARGU` • Modern Standard Arabic: `ARMSA`
Catalan	• `CAES`

Available language packs, continued

Language	Language pack
Czech	• CSCZ
Danish	• DADK
Dutch	• NLNL
English	• Australian English: ENAU • British English: ENUK • Canadian English: ENCA • Irish English: ENIE • Singaporean English: ENSG • U.S. English: ENUS
Flemish	• NLBE
French	• Canadian French: FRCA • French: FRFR
German	• DEDE
Greek	• ELGR
Hebrew	• HBIL
Hindi	• HIIN
Hungarian	• HUHU
Italian	• ITIT
Japanese	• JAJP
Korean	• KOKR
Mandarin	• ZHCN
Persian	• FAIR
Polish	• PLPL
Portuguese	• Portuguese (Portugal): PTPT • Brazilian Portuguese: PTBR
Romanian	• RORO
Russian	• RURU
Slovak	• SKSK
Spanish	• Castilian Spanish: ESES

Available language packs, continued

Language	Language pack
	• Latin American Spanish: ESLA • North American Spanish: ESUS
Swedish	• SVSE
Turkish	• TRTR
Welsh	• CYUK

> **NOTE:**
> For certain languages only the broadband or telephony pack is available, not both.

Sentence Breaking Plug-In

IDOL Speech Server supports the Basis Sentence Breaking plug-in, which is available from the HPE Download Center. The plug-in enables IDOL Speech Server to perform text segmentation on Japanese, Korean, Mandarin, and Taiwanese Mandarin languages. Text segmentation inserts whitespace between words.

> **NOTE:**
> The Basis Sentence Breaking plug-in is available for the following platforms only:
> - Linux 64-bit
> - Windows 32-bit
> - Windows 64-bit

You must install the sentence breaking plug-in to use the `SegmentText` task and the segmentation function in the `LanguageModelBuild` task.

To enable text segmentation

1. Download the 11.5 version of the Basis Sentence Breaking plug-in from the HPE Download Center.
2. Unzip the plug-in directory and save the files to your desired location.
3. Open the IDOL Speech Server configuration file (`speechserver.cfg`) in a text editor.
4. In the `[Paths]` section, set the `RlpDir` parameter to the path of the directory containing the plug-in files.
5. Save and close the configuration file.
6. Restart IDOL Speech Server.

Chapter 4: Configure IDOL Speech Server

This section describes how to configure IDOL Speech Server.

- IDOL Speech Server Configuration Files 53
- Configure Language Packs 62
- Configure Task Queues 63
- Include an External Configuration File 65
- Configure Client Authorization 67
- Customize Logging 69
- Event Handlers 70
- Configure Input and Output Directories 71
- Use XSL Templates to Transform Action Responses 72

IDOL Speech Server Configuration Files

IDOL Speech Server uses two configuration files:

- The `speechserver.cfg` configuration file contains settings that determine the basic details that the IDOL Speech Server needs to run. This file includes location and port details for the IDOL Speech Server, licensing details, and logging details.
- The `speechserver-tasks.cfg` file contains settings for audio analysis tasks and resources that tasks require. In addition to specifying the mandatory fields, the tasks configuration file allows you to set predetermined values for some of the HTTP action parameters.

> **NOTE:**
> If you installed IDOL Speech Server from the Chef package, these files are named `softsoundserver.cfg` and `softsoundserver-tasks.cfg`.

Both configuration files are stored in the IDOL Speech Server installation directory. On Windows, the default directory is `C:\Program Files\HewlettPackardEnterprise\IDOLSpeech`. You can move the `speechserver-tasks.cfg` file to another location, but you must update the directory that is specified by the `TaskConfigs` parameter in the `speechserver.cfg` file. For more information about this parameter, see the *IDOL Speech Server Reference*.

You can modify the settings in the configuration files to customize IDOL Speech Server according to your requirements.

> **NOTE:**
> The action parameters and file names stored on disk, for example in the configuration file or in file name lists supplied to training modules, must be encoded using UTF-8.

Modify Configuration Parameter Values

You modify IDOL Speech Server configuration parameters by directly editing the parameters in the configuration file. When you set configuration parameter values, you must use UTF-8.

> **CAUTION:**
> You must stop and restart IDOL Speech Server for new configuration settings to take effect.

This section describes how to enter parameter values in the configuration file.

Enter Boolean Values

The following settings for Boolean parameters are interchangeable:

```
TRUE = true = ON = on = Y = y = 1
```

```
FALSE = false = OFF = off = N = n = 0
```

Enter String Values

To enter a comma-separated list of strings when one of the strings contains a comma, you can indicate the start and the end of the string with quotation marks, for example:

ParameterName=**cat,dog,bird,"wing,beak",turtle**

Alternatively, you can escape the comma with a backslash:

ParameterName=**cat,dog,bird,wing\,beak,turtle**

If any string in a comma-separated list contains quotation marks, you must put this string into quotation marks and escape each quotation mark in the string by inserting a backslash before it. For example:

ParameterName=**"<font face=\"arial\" size=\"+1\"><b>","<p>"**

Here, quotation marks indicate the beginning and end of the string. All quotation marks that are contained in the string are escaped.

Configuration File Sections

The IDOL Speech Server configuration file (`speechserver.cfg`) contains the following sections representing configurable areas.

[Server]	[Logging]
[Service]	[*MyLogging*]
[Paths]	[*ActionName*]
[License]	[*MyEventHandler*]

For details of these sections and the parameters for each section, see the *IDOL Speech Server Reference*. The following sections describe the general configuration sections.

[Server] Section

The [Server] configuration section contains general settings for IDOL Speech Server. For example:

```
[Server]
Port=13000
NumberOfTaskManagers=5
TasksPerTaskManager=1
MaxLangResources=2
BinaryDataPort=16000
TasksConfig=./speechserver-tasks.cfg
TaskHistorySize=5000
LegacyTaskStates=False
```

[Service] Section

The [Service] section contains parameters that determine which machines are permitted to use and control the IDOL Speech Server service. For example:

```
[Service]
ServicePort=40010
```

[Paths] Section

The [Paths] section contains parameters that specify directories that contain files that IDOL Speech Server requires. For example:

```
[Paths]
FFmpegDirectory=C:\HewlettPackardEnterprise\SpeechServer\libraries
CustomLMDir=T:\customLM
TempDir=T:\temp
TrainedAmDir=T:\trainedAM
```

[License] Section

The [License] section contains licensing details, which you must not change. For example:

```
[License]
LicenseServerHost=127.0.0.1
LicenseServerACIPort=20000
LicenseServerTimeout=600000
LicenseServerRetries=1
```

[Logging] Section

The [Logging] section lists the logging streams to set up to create separate log files for different log message types (query, index, and application). It also contains a subsection for each of the listed logging streams, in which you can configure the parameters that determine how each stream is logged. For example:

```
[Logging]
LogDirectory=$USER_INSTALL_DIR$\logs
LogTime=True
LogEcho=True
LogLevel=FULL

0=ApplicationLogStream
1=ActionLogStream
2=SoftsoundLogStream

[ApplicationLogStream]
LogFile=Application.log
LogTypeCSVs=application

[ActionLogStream]
LogFile=Action.log
LogTypeCSVs=action

[SoftsoundLogStream]
LogFile=Softsound.log
LogTypeCSVs=softsound
```

[*ActionName*] Sections

The [*>ActionName*] sections contain parameters that specify event handlers to use when an action starts, finishes, or returns an error. You configure the event handlers themselves in individual [*MyEventHandler*] sections.

> **NOTE:**
> Only the AddTask and LoadLanguage actions support event handlers.

```
[AddTask]
OnError=ErrorHandler
OnStart=StartHandler
OnFinish=FinishHandler
```

[*MyEventHandler*] Sections

The [*MyEventHandler*] sections contain parameters that control how IDOL Speech Server processes actions when they start, finish, or return an error. You must create a [*MyEventHandler*] section for each handler configuration section that you specify in the [*ActionName*] sections.

```
[ErrorHandler]
LibraryName=HTTPHandler
URL=http://handlers:8080/lo-proxy/callback.htm?

[StartHandler]
LibraryName=TextFileHandler
FilePath=./EventData/
```

[ACIEncryption]

You can use the [ACIEncryption] section to encrypt communications between ACI servers and any applications that use the HPE ACI API. For example:

```
[ACIEncryption]
CommsEncryptionType=GSS
ServiceName=Kerberos
```

[AuthorizationRoles]

The [AuthorizationRoles] defines roles that enable a particular set of actions for particular clients, SSL identities, and GSS principals. You must create a subsection for each authorization role that you define in the [AuthorizationRoles] configuration section.

For example:

```
[AuthorizationRoles]
0=AdminRole
1=UserRole

[AdminRole]
StandardRoles=Admin,Index,ServiceControl
Clients=localhost
SSLIdentities=admin.example.com

[UserRole]
StandardRoles=Query,ServiceStatus
SSLIdentities=admin.example.com,userserver.example.com
```

Tasks Configuration File Sections

The IDOL Speech Server tasks configuration file (speechserver-tasks.cfg) contains the following sections.

[TaskTypes]	[Resources]
[*MyTask*]	[*MyLanguage*]
[*ModuleName*]	[*MyFPDB*]

For details of these sections and the parameters for each section, see the *IDOL Speech Server Reference*. The following sections describe the general configuration sections.

[TaskTypes] Section

The [TaskTypes] section lists the tasks that are configured in the IDOL Speech Server. You must create a [*MyTask*] configuration section for each task type listed in the [TaskTypes] section.

```
[TaskTypes]

// Speech to text
0=speechToText
1=speechToTextFilter
2=speechToTextTelephony
3=punctuateCtm

// Speaker cluster processing
8=ClusterSpeech
9=ClusterSpeechTel
10=ClusterSpeechToTextTel

// Transcript analysis
11=TranscriptAlign
12=TranscriptCheck
13=Scorer

// Language model building
14=LanguageModelBuild
15=TextNorm

// Speaker identification
16=ivSpkId
17=ivSpkIdFeature
18=ivSpkIdTrain
19=ivSpkIdTrainAudio
20=ivSpkIdDevel
21=ivSpkIdDevelAudio
22=ivSpkIdDevelFinal
23=ivSpkIdSetAdd
24=ivSpkIdSetDelete
25=ivSpkIdEditThresh
26=ivSpkIdInfo
```

Related Topics

- Configure Custom Tasks, on page 235

[*MyTask*] Sections

The [*MyTask*] sections define configuration options for each IDOL Speech Server audio processing task. You must create a [*MyTask*] section for each task you have listed in the [TaskTypes] section.

Each section contains details of the schema you use as well as any other parameters required for the task.

```
[speechToText]
0 = a,ts <- audio(MONO, input)
1 = f  <- frontend(_, a)
2 = nf <- normalizer(_, f)
```

```
3 = w1  <- stt(_, nf)
4 = w2  <- postproc(_, w1)
5 = output <- wout(_, w2, ts)
defaultResults = out
```

```
[TranscriptAlign]
0 = w <- ctm2(READ, input)
1 = w2 <- align2(ALIGN, w)
2 = output <- wout2(_, w2)
DefaultResults=Out
```

Related Topics

- Configure Custom Tasks, on page 235

[*ModuleName*] Sections

The [*ModuleName*] configuration sections contain settings for the modules. Create a configuration section for each module that you use in the [*MyTask*] configuration sections. Each configuration section must have the same name as the module referenced in the task schemas. If you use more than one configuration of a module, create a section for each configuration, including any numerical suffixes.

You can set configuration parameters in the individual module configuration sections to variable values. You can use these values to create action parameters that allow you to specify the value of the configuration parameter when you create a task. You can refer the values of all similar configuration parameters to a single configuration parameter where you set a standard value. For details, see Configure Variable Parameters, on page 237.

```
[audio]
inputType = $params.inputType
file = $params.file
streamMode = $stt.mode
sampleFrequency = $stt.lang.sampleFrequency
sugdInputFrequency = $params.sugdInputFrequency
sugdInputChannels = $params.sugdInputChannels
startTime = $params.startTime
endTime = $params.endTime
```

[Resources] Section

The [Resources] section lists the resources that IDOL Speech Server requires, including language packs and AFP databases. You must create a [*MyLanguage*] configuration section for each language pack, and a [*MyFPDB*] configuration section for each Audio Fingerprint database listed in the [Resources] section.

```
[Resources]
0=ENUK
1=ENUS
2=fpdb:AFP
3=fpdb:ADVERTS
4=FRFR
```

```
5=DEDE
6=ARMSA
7=fpdb:AFP
```

Related Topics

- Configure Language Packs, on page 62
- Define the AFP Database, on page 187

[*MyLanguage*] Sections

The [*MyLanguage*] sections contains settings for language packs that you have defined in the [Resources] section. You must create a [*MyLanguage*] section for each language that you have listed in the [Resources] section.

```
[ENUK]
PackDir=ENUK
Pack=ENUK-6.3
SampleFrequency=16000
AmFile = T:\LP\ENUK\ver-ENUK-5.0-16k.am
CustomLM=$params.CustomLM
CustomDct=myDictionary.dct.sz
DNNFile = $params.DNNFile
ClassWordFile = $params.ClassWordFile
PronFile = $params.PronFile

[ENUS]
PackDir=ENUS
Pack=ENUS-6.3
SampleFrequency=16000
AmFile = T:\LP\ENUK\ver-ENUK-5.0-16k.am
CustomLM=$params.CustomLM
CustomDct=myDictionary.dct.sz
DNNFile = $params.DNNFile

[FRFR]
PackDir=FRFR
Pack=FRFR-6.3
SampleFrequency=16000
AmFile = T:\LP\ENUK\ver-ENUK-5.0-16k.am
CustomLM=$params.CustomLM
CustomDct=myDictionary.dct.sz
DNNFile = $params.DNNFile
```

Related Topics

- Configure Variable Parameters, on page 237
- Configure Language Packs, on page 62

[*MyFPDB*] Sections

The Audio Fingerprint database (`fpdb`) configuration sections contain settings for the databases used in IDOL Speech Server.

You can set configuration parameters in these sections to variable values. You can use these values to create action parameters that allow you to specify the value of the configuration parameter when you create a task. For example, the following database configuration allows you to specify which database (the directory it is in, and the base file name of the database) on the command line (using the `PackDir` and `Pack` parameters):

```
[AFPDatabase]
PackDir = $params.packdir
Pack = $params.pack
FxxCacheSize=2
TtxCacheSize=200
```

Alternatively you can explicitly set these values in the configuration file, and specify a particular database:

```
[ADVERTS]
PackDir = C:\databases
Pack = adverts
FxxCacheSize=2
TtxCacheSize=200
```

You must list all Audio Fingerprint database resources in the `[Resources]` section before you use them. In this list, prefix the resource name with **`fpdb:`**.

Related Topics

- Configure Variable Parameters, on page 237
- Audio Fingerprint Identification, on page 187

[*MySidBase*] Section

The speaker identification base pack (`sidbase`) configuration sections contain details of the sid base pack that you want to use for speaker identification. This resource contains details of all the speaker identification base files. If you configure a base pack and set the `SpkIdBasePack` configuration parameter in the speaker identification modules, IDOL Speech Server can automatically find the base files for the speaker identification tasks, and you do not have to specify the base files explicitly.

You must configure the directory and version number for the base pack. For example:

```
[SIDBASE]
PackDir = SpeakerIdPack
Pack = gen-1.8
```

In this case, the `PackDir` is relative to the `SpeakerID` global directory, which is configured in the `SpeakerIDDir` configuration parameter. If you have not configured a `SpeakerID` global directory, the directory is relative to the main server install directory.

You must list the speaker identification base pack resource in the `[Resources]` section before you use it. In this list, you must prefix the resource name with `sidbase:`.

Configure Language Packs

IDOL Speech Server can use language models for several languages at a time. The number of languages that you can configure depends on your license. You must configure each language in the `[Resources]` section of the tasks configuration file.

The `[Resources]` configuration section lists details for the configured language packs, primarily for use with speech-to-text. This section also contains subsections where you can configure resource packs for language identification, audio fingerprinting, phonetic phrase match, and audio security event detection.

To configure language packs

1. Open the IDOL Speech Server tasks configuration file (`speechserver-tasks.cfg`) in a text editor.
2. Find the `[Resources]` section or create one if it does not exist.
3. List each language that you want to configure in the form *N*`=`*Type*`:`*LanguagePack*, where:
   - *N* is the zero-based rank order for the language.
   - *Type* is the resource type, which is either `lang`, `langvt`, `fpdb`, or `sidbase` (see Supported Resources, on page 47). You can omit the type for `lang` resources, because it is the default type.
   - *LanguagePack* is the code for the language pack. For a list of available language packs, see Available language packs, on page 49.

   For example:

   ```
   [Resources]
   0=ENUK
   1=ENUS
   2=langvt:ENUS-pm
   ```
4. Create a configuration section for each language that you listed in the `[Resources]` section.
5. Set the `Pack` parameter to the name of the language pack. For example:

   ```
   [ENUK]
   Pack=ENUK-6.3
   ```
6. Set the `PackDir` parameter to the directory where the language model resides. For example:

   ```
   PackDir=ENUK
   ```

   If the parameter omits the directory path or specifies a relative path, IDOL Speech Server automatically prefixes the value with the default language pack directory path when it searches for the pack.
7. Set the `SampleFrequency` parameter to the sample frequency of the audio that the language pack is processing. For example:

   ```
   SampleFrequency=16000
   ```
8. If you configure the language pack to use a custom language model, set the `CustomLM` parameter

to the name and weight of the custom language model separated by a colon (:).

You must also set `CustomDct` parameter to the name of the custom dictionary, without the `.dct.sz` file name extension. For example:

```
CustomLM=news:0.6
CustomDct=news
```

In this example, IDOL Speech Server uses the file `news.tlm` in the custom language model directory with a weight of `0.6` and the dictionary file `news.dct.sz` in the same directory.

9. If you want to change the DNN file to use (for example, to use the smaller, faster DNN provided in the language pack instead of the standard one), set the value of the `DNNFile` parameter to the DNN file that you want to use.

   NOTE:
   HPE recommends (and for 7.0+ versions of language packs, it is compulsory) that you include the following lines in the configuration file for the `[frontend]` and `[normalizer]` modules, so that IDOL Speech Server can access the header to determine the quantity and nature of the extracted acoustic feature vectors:

   ```
   DNNFile = $stt.lang.DNNFile
   DNNFileStd = $stt.lang.DNNFileStd
   ```

   For more information, see the *IDOL Speech Server Reference*.

10. If the language pack is for use specifically with language identification tasks (for example, SYLS packs), set the `LangID` parameter to **`True`**.
11. Configure any other available parameters for your language model or language identification model.
12. Save and close the configuration file.
13. Restart IDOL Speech Server for your changes to take effect (see Start and Stop IDOL Speech Server, on page 75).

Configure Task Queues

By default, IDOL Speech Server refuses tasks if the active task count or active language pack count are at their maximum configured values. This behavior can be desirable if a task requires immediate processing, such as live audio streaming. It is also useful if several IDOL Speech Server instances are running, so that tasks can be resubmitted to an available server. However, if you process many audio files on a single server, you can queue the tasks until resources become available.

Enable Queues

To enable queuing, set the `EnableQueue` parameter to **`True`** in the `[Server]` section of the IDOL Speech Server configuration file. For example:

```
[Server]
EnableQueue=True
```

When queuing is enabled, the server maintains a queue of tasks awaiting processing. Initially the queue is empty.

When an AddTask action is sent to IDOL Speech Server, the task is assigned to a task manager and runs if sufficient resources are available and if no other tasks are queued. If insufficient resources exist or if other tasks are in the queue, the new task is added to the end of the queue.

When a task completes and the resources become available, IDOL Speech Server assigns the next task in the queue to a task manager and removes it from the queue.

> **NOTE:**
> If you enable queuing, the CheckResources, LoadLanguage, and UnloadLanguage actions are disabled.

If you use the AbortTask action to end a queued task, IDOL Speech Server removes the task from the queue and does not process it.

Related Topics

- Check Available Resources, on page 102
- Load a Language Pack Manually , on page 240
- Unload a Language Pack, on page 242
- End a Task, on page 87

View Queue Status

To view the status of the task queue, send a GetStatus action.

For example:

```
http://localhost:13000/action=GetStatus
```

This action returns the status information for the server, including queue information from the IDOL Speech Server located on the local machine using ACI port 13000.

If you want to omit queue information from GetStatus, add the ShowQueue parameter to the GetStatus action. For example:

```
http://localhost:13000/action=GetStatus&ShowQueue=False
```

This action returns the status information for the server, excluding queue information from the IDOL Speech Server located on the local machine using ACI port 13000.

When a task is sent to IDOL Speech Server, it is given a unique task *token*. If you add a task token to the GetStatus action to view the status of a specific task, you can view the position of that task in the queue. It does not show information on the rest of the queue.

> **NOTE:**
> By default, IDOL Speech Server supports a maximum of 5,000 tokens for completed tasks at any one time. If there are more than 5,000 completed tasks, IDOL Speech Server deletes the task in the list that has the earliest completion time. The GetStatus action does not recognize the tokens of tasks that have been removed from the history list.
>
> To increase the number of completed tasks for which the server retains information, edit the value of the TaskHistorySize parameter in the [Server] section of the configuration file. For more information, see the *IDOL Speech Server Reference*.

Related Topics

- GetStatus, on page 82
- Start and Stop Tasks, on page 85
- Check the Status of a Task, on page 87
- Customize Logging, on page 69

Include an External Configuration File

You can share configuration sections or parameters between ACI server configuration files. The following sections describe different ways to include content from an external configuration file.

You can include a configuration file in its entirety, specified configuration sections, or a single parameter.

When you include content from an external configuration file, the `GetConfig` and `ValidateConfig` actions operate on the combined configuration, after any external content is merged in.

In the procedures in the following sections, you can specify external configuration file locations by using absolute paths, relative paths, and network locations. For example:

```
../sharedconfig.cfg
K:\sharedconfig\sharedsettings.cfg
\\example.com\shared\idol.cfg
file://example.com/shared/idol.cfg
```

Relative paths are relative to the primary configuration file.

> **NOTE:**
> You can use nested inclusions, for example, you can refer to a shared configuration file that references a third file. However, the external configuration files must not refer back to your original configuration file. These circular references result in an error, and IDOL Speech Server does not start.
>
> Similarly, you cannot use any of these methods to refer to a different section in your primary configuration file.

Include the Whole External Configuration File

This method allows you to import the whole external configuration file at a specified point in your configuration file.

To include the whole external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the external configuration file.
3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. For example:

```
< "K:\sharedconfig\sharedsettings.cfg"
```

4. Save and close the configuration file.

Include Sections of an External Configuration File

This method allows you to import one or more configuration sections from an external configuration file at a specified point in your configuration file. You can include a whole configuration section in this way, but the configuration section name in the external file must exactly match what you want to use in your file. If you want to use a configuration section from the external file with a different name, see Merge a Section from an External Configuration File, on the next page.

To include sections of an external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the external configuration file section.
3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file name, add the configuration section name that you want to include. For example:

   ```
   < "K:\sharedconfig\extrasettings.cfg" [License]
   ```

   > **NOTE:**
   > You cannot include a section that already exists in your configuration file.

4. Save and close the configuration file.

Include a Parameter from an External Configuration File

This method allows you to import a parameter from an external configuration file at a specified point in your configuration file. You can include a section or a single parameter in this way, but the value in the external file must exactly match what you want to use in your file.

To include a parameter from an external configuration file

1. Open your configuration file in a text editor.
2. Find the place in the configuration file where you want to add the parameter from the external configuration file.
3. On a new line, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. After the configuration file name, add the name of the configuration section name that contains the parameter, followed by the parameter name. For example:

   ```
   < "license.cfg" [License] LicenseServerHost
   ```

   To specify a default value for the parameter, in case it does not exist in the external configuration file, specify the configuration section, parameter name, and then an equals sign (=) followed by the default value. For example:

```
< "license.cfg" [License] LicenseServerHost=localhost
```

4. Save and close the configuration file.

Merge a Section from an External Configuration File

This method allows you to include a configuration section from an external configuration file as part of your IDOL Speech Server configuration file. For example, you might want to specify a standard SSL configuration section in an external file and share it between several servers. You can use this method if the configuration section that you want to import has a different name to the one you want to use.

To merge a configuration section from an external configuration file

1. Open your configuration file in a text editor.
2. Find or create the configuration section that you want to include from an external file. For example:

   ```
   [SSLOptions1]
   ```

3. After the configuration section name, type a left angle bracket (<), followed by the path to and name of the external configuration file, in quotation marks (""). You can use relative paths and network locations. For example:

   ```
   [SSLOptions1] < "../sharedconfig/ssloptions.cfg"
   ```

   If the configuration section name in the external configuration file does not match the name that you want to use in your configuration file, specify the section to import after the configuration file name. For example:

   ```
   [SSLOptions1] < "../sharedconfig/ssloptions.cfg" [SharedSSLOptions]
   ```

   In this example, IDOL Speech Server uses the values in the `[SharedSSLOptions]` section of the external configuration file as the values in the `[SSLOptions1]` section of the IDOL Speech Server configuration file.

   > **NOTE:**
   > You can include additional configuration parameters in the section in your file. If these parameters also exist in the imported external configuration file, IDOL Speech Server uses the values in the local configuration file. For example:
   >
   > ```
   > [SSLOptions1] < "ssloptions.cfg" [SharedSSLOptions]
   > SSLCACertificatesPath=C:\IDOL\HTTPConnector\CACERTS\
   > ```

4. Save and close the configuration file.

Configure Client Authorization

You can configure IDOL Speech Server to authorize different operations for different connections.

Authorization roles define a set of operations for a set of users. You define the operations by using the `StandardRoles` configuration parameter, or by explicitly defining a list of allowed actions in the

`Actions` and `ServiceActions` parameters. You define the authorized users by using a client IP address, SSL identities, and GSS principals, depending on your security and system configuration.

For more information about the available parameters, see the *IDOL Speech Server Reference*.

To configure authorization roles

1. Open your configuration file in a text editor.
2. Find the `[AuthorizationRoles]` section, or create one if it does not exist.
3. In the `[AuthorizationRoles]` section, list the user authorization roles that you want to create. For example:

   ```
   [AuthorizationRoles]
   0=AdminRole
   1=UserRole
   ```

4. Create a section for each authorization role that you listed. The section name must match the name that you set in the `[AuthorizationRoles]` list. For example:

   ```
   [AdminRole]
   ```

5. In the section for each role, define the operations that you want the role to be able to perform. You can set `StandardRoles` to a list of appropriate values, or specify an explicit list of allowed actions by using `Actions`, and `ServiceActions`. For example:

   ```
   [AdminRole]
   StandardRoles=Admin,ServiceControl,ServiceStatus

   [UserRole]
   Actions=GetVersion
   ServiceActions=GetStatus
   ```

   > **NOTE:**
   > The standard roles do not overlap. If you want a particular role to be able to perform all actions, you must include all the standard roles, or ensure that the clients, SSL identities, and so on, are assigned to all relevant roles.

6. In the section for each role, define the access permissions for the role, by setting `Clients`, `SSLIdentities`, and `GSSPrincipals`, as appropriate. If an incoming connection matches one of the allowed clients, principals, or SSL identities, the user has permission to perform the operations allowed by the role. For example:

   ```
   [AdminRole]
   StandardRoles=Admin,ServiceControl,ServiceStatus
   Clients=localhost
   SSLIdentities=admin.example.com
   ```

7. Save and close the configuration file.
8. Restart IDOL Speech Server for your changes to take effect.

Customize Logging

You can customize logging by setting up your own *log streams*. Each log stream creates a separate log file in which specific log message types (for example, action, index, application, or import) are logged.

To set up log streams

1. Open the IDOL Speech Server configuration file in a text editor.
2. Find the `[Logging]` section. If the configuration file does not contain a `[Logging]` section, add one.
3. In the `[Logging]` section, create a list of the log streams that you want to set up, in the format *`N=LogStreamName`*. List the log streams in consecutive order, starting from 0 (zero). For example:

   ```
   [Logging]
   LogLevel=FULL
   LogDirectory=logs
   0=ApplicationLogStream
   1=ActionLogStream
   ```

   You can also use the `[Logging]` section to configure any default values for logging configuration parameters, such as `LogLevel`. For more information, see the *IDOL Speech Server Reference*.
4. Create a new section for each of the log streams. Each section must have the same name as the log stream. For example:

   ```
   [ApplicationLogStream]
   [ActionLogStream]
   ```

5. Specify the settings for each log stream in the appropriate section. You can specify the type of logging to perform (for example, full logging), whether to display log messages on the console, the maximum size of log files, and so on. For example:

   ```
   [ApplicationLogStream]
   LogTypeCSVs=application
   LogFile=application.log
   LogHistorySize=50
   LogTime=True
   LogEcho=False
   LogMaxSizeKBs=1024

   [ActionLogStream]
   LogTypeCSVs=action
   LogFile=logs/action.log
   LogHistorySize=50
   LogTime=True
   LogEcho=False
   LogMaxSizeKBs=1024
   ```

6. Save and close the configuration file. Restart the service for your changes to take effect.

IDOL Speech Server Logs and Error Reports

In addition to the standard logging incorporated into all IDOL ACI servers, IDOL Speech Server logs:

- Any errors or warnings encountered during the execution of tasks and actions
- Any significant state changes for running tasks
- Available resources

By default, these messages are written to the `softsound.log` log file.

Related Topics

- Customize Logging, on the previous page

Event Handlers

Some of the actions that you can send to IDOL Speech Server are asynchronous. Asynchronous actions do not run immediately, but are added to a queue. This means that the person or application that sends the action does not receive an immediate response. However, you can configure IDOL Speech Server to call an event handler when an asynchronous action starts, finishes, or encounters an error.

You might use an event handler to:

- Return data about an event back to the application that sent the action.
- Write event data to a text file, to log any errors that occur.

IDOL Speech Server can call an event handler for the following events.

OnStart	The `OnStart` event is called when IDOL Speech Server starts processing an asynchronous action from the queue.
OnFinish	The `OnFinish` event is called when IDOL Speech Server successfully finishes processing an asynchronous action.
OnError	The `OnError` event is called when an asynchronous action fails and cannot continue.

Configure an Event Handler

Use the following procedure to configure an event handler.

To configure an event handler

1. Open the IDOL Speech Server configuration file in a text editor.
2. Use the `OnStart`, `OnFinish`, or `OnError` parameter to specify the name of a section in the configuration file that contains event handler settings for the corresponding event.
   - To run an event handler for all asynchronous actions, set these parameters in the `[Actions]` section. For example:

     ```
     [Actions]
     OnStart=NormalEvents
     ```

```
OnFinish=NormalEvents
OnError=ErrorEvents
```

- To run an event handler for a specific action, set these parameters in the [*ActionName*] section, where *ActionName* is the name of the action. The following example runs an event handler when the AddTask action starts and finishes successfully:

```
[AddTask]
OnStart=NormalEvents
OnFinish=NormalEvents
```

3. Create a new section in the configuration file to contain the settings for your event handler. You must name the section using the name you specified with the OnStart, OnFinish, or OnError parameter.
4. In the new section, set the following parameters.

LibraryName (Required) The name of the library to use to handle the event.

- To write event data to a text file, set this parameter to **TextFileHandler**, and then set the FilePath parameter to specify the path of the file.
- To send event data to a URL, set this parameter to **HttpHandler**, and then use the HTTP event handler parameters to specify the URL, proxy server settings, credentials, and so on.

For example:

```
[NormalEvents]
LibraryName=TextFileHandler
FilePath=./events.txt

[ErrorEvents]
LibraryName=HTTPHandler
URL=http://handlers:8080/lo-proxy/callback.htm?
```

5. Save and close the configuration file. You must restart IDOL Speech Server for your changes to take effect.

The following XML is an example of the ACI response sent to the event handler on the OnFinish event:

```
<action>
       <status>Finished</status>
       <queued_time>2009-Jun-08 17:57:29</queued_time>
       <time_in_queue>0</time_in_queue>
       <process_start_time>2012-Jun-08 17:57:29</process_start_time>
       <time_processing>86</time_processing>
       <token>MTkyLjE2OC45NS4yNDoxMTAwMDpJTkdFU1Q6LTEyOTc5NjgxODY=</token>
</action>
```

Configure Input and Output Directories

By default, IDOL Speech Server can read from and write to any directory. However, for increased security you can specify a set of allowed directories to restrict access to the file system.

You can use the following parameters in the [Server] section of the IDOL Speech Server base configuration file to configure input and output directory settings:

- RestrictOutputDirs
- RestrictInputDirs
- AllowedOuputDirectoriesCSVs
- AllowedInputDirectoriesCSVs

For more information on how to configure these settings, see the *IDOL Speech Server Reference*.

The configuration of allowed input and output directories affects the following settings:

1. **File name parameters**. Any file name parameters (for example, audio files, output files, and so on) that you specify in the configuration file or at the command line must be in an allowed directory or subdirectory.
2. **File path parameters**. File path parameters (for example, the temporary directory) used by specific tasks must point to an allowed directory or subdirectory.
3. **List path parameters**. The paths that are used with list files (where a file name is created by combining the element of the list with the specified path) must point to an allowed directory or subdirectory.
4. **List elements**. When you restrict directories, list elements must be relative rather than absolute, and must not contain .. or ~.
5. **Action-level restrictions**. You can use the File action parameter with the getResults action to specify the results file to read. The file must be in an allowed directory or subdirectory.

If you set input and output directory restrictions but specify a file or path that is outside the allowed directories, IDOL Speech Server returns an error and the related task fails.

Use XSL Templates to Transform Action Responses

You can transform the action responses returned by IDOL Speech Server using XSL templates. You must write your own XSL templates and save them with either an .xsl or .tmpl file extension. For more information about XSL, see http://www.w3.org/TR/xslt.

After creating the templates, you must configure IDOL Speech Server to use them, and then apply them to the relevant actions.

To enable XSL transformations

1. Ensure that the autnxslt library is located in the same directory as your configuration file. If the library is not included in your installation, you can obtain it from HPE Technical Support.
2. Open the IDOL Speech Server configuration file in a text editor.
3. In the [Server] section, set the XSLTemplates parameter to **True**.

   > **CAUTION:**
   > If XSLTemplates is set to **True** and the autnxslt library is not present in the same directory as the configuration file, the server will not start.

4. In the [Paths] section, set the TemplateDirectory parameter to the path to the directory that contains your XSL templates.

5. Save and close the configuration file.
6. Restart IDOL Speech Server for your changes to take effect.

To apply a template to action output

- Add the following parameters to the action.

`Template`	The name of the template to use to transform the action output. Exclude the folder path and file extension.
`OutputEncoding`	Set to **`UTF8`**.
`ForceTemplateRefresh`	(Optional) If you modified the template after the server started, set this parameter to **`True`** to force the ACI server to reload the template from disk rather than from the cache.

For example:

```
action=QueueInfo&QueueName=AddTask
                &QueueAction=GetStatus
                &Token=MTkyLjE2OC45NS4yNDox
                &Template=Form
                &OutputEncoding=UTF8
```

In this example, IDOL Speech Server applies the XSL template `Form.tmpl` to the response from a `QueueInfo` action.

> **NOTE:**
> If the action returns an error response, IDOL Speech Server does not apply the XSL template.

Chapter 5: Run IDOL Speech Server

This chapter describes how to start and stop IDOL Speech Server, and how to send actions and tasks.

- Start and Stop IDOL Speech Server 75
- Send Actions to IDOL Speech Server 76
- Verify IDOL Speech Server Runs Correctly 79
- Display Online Help 84
- Audio Requirements 84
- Start and Stop Tasks 85
- Schedule Tasks 88
- Standard Tasks 91
- Display Configured Tasks 98
- Display Task Details 99
- Display Configured Resources 101
- Check Available Resources 102
- Get Task Results 105
- Create and Manage Lists 109
- Back Up and Restore IDOL Speech Server 111
- Troubleshooting 112

Start and Stop IDOL Speech Server

The following sections describe how to start and stop IDOL Speech Server.

Start IDOL Speech Server

Use the following procedure to start IDOL Speech Server.

To start IDOL Speech Server

1. Start the License Server by performing one of the following actions.
   - Double-click the *Installation*LicenseServer.exe file in your License Server installation directory (Windows), where *Installation* is the service prefix that you selected during the installation process.
   - Use the start script (UNIX).
   - Start the License Server Service from a system dialog box (Windows).
2. Start the IDOL Speech Server using one of the following methods:
   - In the Start IDOL Speech Server folder, run the following command:

     ```
     softsoundserver.exe -configfile configfilename
     ```

where *configfilename* is the name of the configuration file. The default value of -configfile is softsoundserver.cfg, so if your configuration file is also softsoundserver.cfg, you can omit the parameter.

- Double-click the *Installation*SoftSoundServer.exe file in your IDOL Speech Server installation folder (Windows).
- Use the start script (for UNIX).
- Start the IDOL Speech Server Service from a system dialog box (Windows).

To start the License Server Service from a system dialog box

1. Open the Windows **Services** dialog box.
2. Select the ***Installation*LicenseServer** service and click **Start** to start the License Server, where ***Installation*** is your service prefix.
3. Click **Close** to close the **Services** dialog box.

To start the IDOL Speech Server Service from a system dialog box

1. Display the Windows **Services** dialog box.
2. Select the ***Installation*Softsound** service and click **Start** to start IDOL Speech Server.
3. Click **Close** to close the **Services** dialog box.

Stop IDOL Speech Server

Use one of the following procedures to stop IDOL Speech Server.

- Send the following action to IDOL Speech Server's service port. (You must specify a service port in the IDOL Speech Server configuration file.)

  http://*SpeechServerHost*:*ServicePort*/action=stop

  where:

SpeechServerHost	is the name or IP address of the host where IDOL Speech Server runs.
ServicePort	is the IDOL Speech Server service port specified in the [Service]section of the IDOL Speech Server configuration file. For more information on how to configure your service port, see the *IDOL Speech Server Reference*.

- Use the stop script (for UNIX).
- Stop the IDOL Speech Server Service from a system dialog box (Windows for NT):
  1. Display the Windows **Services** dialog box.
  2. Select the ***Installation*SoftsoundServer** service and click **Stop** to stop IDOL Speech Server.
  3. Click **Close** to close the **Services** dialog box.

Send Actions to IDOL Speech Server

You can make requests to IDOL Speech Server by using either GET or POST HTTP request methods.

- A GET request sends parameter-value pairs in the request URL. GET requests are appropriate for sending actions that retrieve information from IDOL Speech Server, such as the `GetStatus` action. For more information, see Send Actions by Using a GET Method, below.
- A POST request sends parameter-value pairs in the HTTP message body of the request. POST requests are appropriate for sending data to IDOL Speech Server. In particular, you must use POST requests to upload and send files to IDOL Speech Server. For more information, see Send Data by Using a POST Method, on the next page.

> **NOTE:**
> The `MaxInputString` and `MaxFileUploadSize` configuration parameters set a limit on the maximum size of HTTP strings and the maximum size of files that you can upload. The default values for these parameters allow HTTP strings that contain a maximum of 64,000 characters, and uploaded files with a maximum size of 10,000,000 bytes.
>
> The default IDOL Speech Server configuration file overrides the default `MaxFileUploadSize` by setting it to -1 (unlimited). You can modify this value to set a limit, if required.

Send Actions by Using a GET Method

You can use GET requests to send actions that retrieve information from IDOL Speech Server.

When you send an action using a GET method, you use a URL of the form:

`http://`*`host`*`:`*`port`*`/?action=`*`action`*`&`*`parameters`*

where:

`host`	is the IP address or name of the machine where IDOL Speech Server is installed.
`port`	is the IDOL Speech Server ACI port.
`action`	is the name of the action that you want to run. The `action` (or `a`) parameter must be the first parameter in the URL string, directly after the host and port details.
`parameters`	are the required and optional parameters for the action. These parameters can follow the `action` parameter in any order.

You must:

- Separate each parameter from its value with an equals symbol (`=`).
- Separate multiple values with a comma (`,`).
- Separate each parameter-value pair with an ampersand (`&`).

For more information about the actions that you can use with IDOL Speech Server, refer to the *IDOL Speech Server Reference*.

GET requests can send only limited amounts of data and cannot send files directly. However, you can set a parameter to a file path if the file is on a file system that IDOL Speech Server can access. IDOL Speech Server must also be able to read the file.

Send Data by Using a POST Method

You can send files and binary data directly to IDOL Speech Server using a POST request. One possible way to send a POST request over a socket to IDOL Speech Server is using the cURL command-line tool.

The data that you send in a POST request must adhere to specific formatting requirements. You can send only the following content types in a POST request to IDOL Speech Server:

- `application/x-www-form-urlencoded`
- `multipart/form-data`

> **TIP:**
> IDOL Speech Server rejects POST requests larger than the size specified by the configuration parameter `MaxFileUploadSize`.

Application/x-www-form-urlencoded

The `application/x-www-form-urlencoded` content type describes form data that is sent in a single block in the HTTP message body. Unlike the query part of the URL in a GET request, the length of the data is unrestricted. However, IDOL Speech Server rejects requests that exceed the size specified by the configuration parameter `MaxFileUploadSize`.

This content type is inefficient for sending large quantities of binary data or text containing non-ASCII characters, and does not allow you to upload files. For these purposes, HPE recommends sending data as `multipart/form-data` (see Multipart/form-data, below).

In the request:

- Separate each parameter from its value with an equals symbol (`=`).
- Separate multiple values with a comma (`,`).
- Separate each parameter-value pair with an ampersand (`&`).
- Base-64 encode any binary data.
- URL encode all non-alphanumeric characters, including those in base-64 encoded data.

Multipart/form-data

In the `multipart/form-data` content type, the HTTP message body is divided into parts, each containing a discrete section of data.

Each message part requires a header containing information about the data in the part. Each part can contain a different content type; for example, `text/plain`, `image/png`, `image/gif`, or `multipart/mixed`. If a parameter specifies multiple files, you must specify the **`multipart/mixed`** content type in the part header.

Encoding is optional for each message part. The message part header must specify any encoding other than the default (7BIT).

Multipart/form-data is ideal for sending non-ASCII or binary data, and is the only content type that allows you to upload files. For more information about form data, see http://www.w3.org/TR/html401/interact/forms.html.

> **NOTE:**
> In cURL, you specify each message part using the `-F` (or `--form`) option. To upload a file in a message part, prefix the file name with the `@` symbol. For more information on cURL syntax, see the cURL documentation.

Example

The following example sends `audio.mp3` (using cURL) as `multipart/form-data` to IDOL Speech Server located on the `localhost`, using port 13000. The example action runs the `SpeechToText` task on the audio file.

> **NOTE:**
> To send binary data to IDOL Speech Server tasks you must set the `InputType` parameter to `Data`.

```
curl http://localhost:13000/?action=AddTask -F Type=SpeechToText -F InputType=Data
-F taskdata=@audio.mp3  -F Out=test.ctm
```

Verify IDOL Speech Server Runs Correctly

After you install IDOL Speech Server, you can run the following actions to verify that IDOL Speech Server runs correctly.

GetRequestLog

Send a `GetRequestLog` action to IDOL Speech Server to return a log of the requests that have been made to it, including the following information:

- The date and time that a request was made
- The client IP address that made the request
- The internal thread that handled the action

For example:

`http://`*`IDOLSpeechServerHost`*`:`*`port`*`/action=GetRequestLog`

For further details on the `GetRequestLog` action, see the *IDOL Speech Server Reference*.

Related Topics

- Send Actions to IDOL Speech Server, on page 76

GetLicenseInfo

Send a `GetLicenseInfo` action to IDOL Speech Server to return information about your license. This action checks whether your license is valid and identifies the IDOL Speech Server operations your license includes.

For example:

```
http://IDOLSpeechServerhost:port/action=GetLicenseInfo
```

The following result indicates that your license is valid.

```
<autn:license>
	<autn:validlicense>True</autn:validlicense>
</autn:license>
```

The following result indicates that your license includes specified actions.

```
<autn:actions
	<autn:action>
		<autn:name>addTask</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
	<autn:action>
		<autn:name>checkResources</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
	<autn:action>
		<autn:name>getResults</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
	<autn:action>
		<autn:name>abortTask</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
	<autn:action>
		<autn:name>loadLanguage</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
	<autn:action>
		<autn:name>unloadLanguage</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
	<autn:action>
		<autn:name>getStatus</autn:name>
		<autn:licensed>True</autn:licensed>
	</autn:action>
</autn:actions>
```

GetLicenseCounts

Send a `GetLicenseCounts` action to IDOL Speech Server to view information on modules and resources, in addition to the general information on licensed actions returned by the `GetLicenseInfo` action.

The action returns two lists, a list of module license counts, and a list of resources. For example:

```
<autnresponse>
   <action>GETLICENSECOUNTS</action>
   <response>SUCCESS</response>
   <responsedata>
      <modules>
         <license>
            <name>STT</name>
            <max>4</max>
            <used>3</used>
            <free>1</free>
         </license>
         <license>
            <name>WOUT</name>
            <max>unlimited</max>
            <used>5</used>
            <free>unlimited</free>
         </license>
      </modules>
      <resources>
         <license>
            <name>ENUK</name>
            <max>1</max>
            <used>1</used>
            <free>0</free>
         </license>
      </resources>
   </responsedata>
</autnresponse>
```

The action returns the following information for each module and resource license key:

`<name>`	The name of the license key.
`<max>`	The maximum number of concurrent licensed usage slots.
`<used>`	The number of instances currently being used.
`<free>`	The number of available instances.

You can use the following action parameters to filter the contents of the response:

`HideUnlicensed`	By default, IDOL Speech Server returns both licensed and unlicensed modules and resources; set `HideUnlicensed` parameter to **`True`** to hide any unlicensed modules or resources.
`Modules`	By default, IDOL Speech Server includes module license information in the response; set `Modules` to **`False`** to exclude this information.
`Resources`	By default, IDOL Speech Server includes resource license information in the response; set `Resources` to **`False`** to exclude this information.
`Key`	The license key to search for (for example, `ENUK`). If you specify a key to search

	for, the response shows only the matching entry (if found).

For more information, see the *IDOL Speech Server Reference*.

GetStatus

Use the `GetStatus` action to verify that the IDOL Speech Server service is running. For information on the `GetStatus` action, see the *IDOL Speech Server Reference*.

For example:

`http://IDOLSpeechServerHost:port/action=GetStatus`

You can view details of the path to the base configuration file and the tasks configuration file that IDOL Speech Server uses in the `<baseconfigpath>` and `<tasksconfigpath>` fields in the `GetStatus` response.

```
<autnresponse>
  <action>GETSTATUS</action>
  <response>SUCCESS</response>
    <responsedata>
      <version>10.9.0.1</version>
      <servicePort>19000</servicePort>
      <binaryDataPort>16000</binaryDataPort>

<baseconfigpath>/opt/HewlettPackardEnterprise/IDOLServer/IDOLSpeech/speechserver.cf
g</baseconfigpath>

<tasksconfigpath>/opt/HewlettPackardEnterprise/IDOLServer/IDOLSpeech/speechserver-
tasks.cfg</tasksconfigpath>
      <resources></resources>
    </responsedata>
</autnresponse>
```

In addition, you can also view information on tasks, such as the output files associated with any task, including the number of files, the file name, and the label that you can use to retrieve the file. This can be particularly useful in cases where the output files were automatically generated.

The `GetStatus` response also includes details of any limitations on resource loading, as well as details of currently loaded and active resources. For example:

```
<autnresponse>
    <action>GETSTATUS</action>
    <response>SUCCESS</response>
    <responsedata>
        <resources>
            <limits>
                <maxLangResources>2</maxLangResources>
                <minLangResources>1</minLangResources>
                <maxAfpResources>Unlimited</maxAfpResources>
                <minAfpResources>1</minAfpResources>
                <maxSidResources>Unlimited</maxSidResources>
```

```
                <minSidResources>1</minSidResources>

            </limits>
            <usage>
                <totalLoaded>4</totalLoaded>
                <totalActive>4</totalActive>
                <langsLoaded>2</langsLoaded>
                <langsActive>2</langsActive>
                <afpLoaded>1</afpLoaded>
                <afpActive>1</afpActive>
                <sidLoaded>1</sidLoaded>
                <sidActive>1</sidActive>
            </usage>
        </resources/
    </responsedata>
</autnresponse>
```

This example shows the response when you submit a `GetStatus` action to an instance of IDOL Speech Server with four loaded resources in total. Four resources are active (that is, in use or manually loaded and not currently unloadable). There is no limitation on the total number of resources.

Two of the loaded resources are language resources (`Lang` or `Langvt`). The limit on the number of language resources is set to 2, so you cannot load any additional language resources.

One of the resources is for audio fingerprinting (`Fpdb`), and one is for speaker identification (`SidBase`). There is no upper limit for SID or AFP resources.

For more information on how to use the `maxLangResources`, `maxAfpResources`, and `maxSidResources` parameters to configure resource loading, see the *IDOL Speech Server Reference*.

The `GetStatus` response also includes task information, including the number of active and queued tasks. For example:

```
<tasks>
    <maxTasks>8</maxTasks>
    <running>8</running>
    <queued>5</queued>
    ...
</tasks>
```

For each specific task, the response includes task details, including information on any warnings associated with the task, the time and date of the warning, and the warning index. For example:

```
<nwarnings>2</nwarnings>
<maxwarnings>5</maxwarnings>
<warnings>
    <warning>
        <index>1</index>
        <message>[ivdevel] No positive examples for template Brown</message>
        <time>14/01/2017 15:45:20</time>
    </warning>
    <warning>
        <index>2</index>
```

```
        <message>[ivdevel] No development scores found for template [Smith] –
threshold not set</message>
        <time>14/01/2017 15:45:20</time>
    </warning>
</warnings>
```

For information on how to use the `MaxWarnings` parameter to restrict the number of individual warnings included in the `GetStatus` response, and how to use the `Brief` parameter to reduce the amount of information in the response to prevent it from becoming too bloated, see the *IDOL Speech Server Reference*.

Related Topics

- Check the Status of a Task, on page 87
- Customize Logging, on page 69
- View Queue Status, on page 64
- Configure Variable Parameters, on page 237

Display Online Help

You can display the IDOL Speech Server Reference by sending an action from your web browser. The IDOL Speech Server Reference describes the actions and configuration parameters that you can use with IDOL Speech Server.

For IDOL Speech Server to display help, the help data file (`help.dat`) must be available in the installation folder.

To display help for IDOL Speech Server

1. Start IDOL Speech Server.
2. Send the following action from your web browser:

   `http://host:port/action=Help`

   where:

   host is the IP address or name of the machine on which IDOL Speech Server is installed.

   port is the ACI port by which you send actions to IDOL Speech Server (set by the `Port` parameter in the `[Server]` section of the configuration file).

   For example:

   `http://12.3.4.56:9000/action=help`

Audio Requirements

The following audio requirements apply to all tasks that accept audio input.

Audio and Video Files

If you set the `FFmpegDirectory` parameter in the configuration file, IDOL Speech Server accepts several audio and video file types, such as MP3, MP4, WMA, WMV, MOV, and so on. For a complete list of accepted file formats, see the FFmpeg documentation.

If you do not set the `FFmpegDirectory` configuration parameter, IDOL Speech Server supports only 16-bit, linear, little-endian PCM data with sampling rates of either 8,000 or 16,000 Hz.

Streamed Audio

Streamed audio must meet the following requirements for use in IDOL Speech Server:

- 16-bit linear PCM.
- A sampling rate of 8,000 or 16,000 Hz. Audio originally sampled at other rates must be downsampled to either 8,000 or 16,000 Hz. For best results, HPE recommends that you use the highest possible sampling rate.

  For example, downsample 44 kHz audio to 16 kHz; downsample 11 kHz audio to 8 kHz. Do not upsample the data, because upsampling often leads to poor results.
- Not encoded.

There are several third-party tools that you can use to convert audio into the required format.

Start and Stop Tasks

Tasks are defined in the IDOL Speech Server configuration file. The `[TaskTypes]` configuration section defines a list of all the available tasks. Individual task configuration sections define operations for each task to perform.

To send actions related to specific tasks, you must specify a `Type` parameter. The `Type` parameter corresponds to the name of the configuration section that defines the task details. For example, if there is a `[SpeechToText]` configuration section for a task, set the `Type` parameter to **`SpeechToText`**.

For each successfully submitted task, a unique task ID token returns that you can use to get any task-specific information later. The token is a base-64 encoded set of characters, which encapsulates information such as the ACI server's host name, port, and the name of the action sent. Tokens have no fixed size; however, they are case sensitive.

Related Topics

- Send Actions to IDOL Speech Server, on page 76
- Configure Custom Tasks, on page 235

Start a Task

To create a task, send an `AddTask` ACI action to your IDOL Speech Server host and ACI port.

For example:

`http://`*`host`*`:`*`port`*`/action=AddTask&Type=`*`task`*`&`*`requiredParams`*`&`*`optionalParams`*

where:

host	is the IP address or name of the machine where IDOL Speech Server is installed.
port	is the ACI port by which actions are sent to IDOL Speech Server (set by the Port parameter in the IDOL Speech Server configuration file's [Server]section).
task	is the name of the configuration section where the options for the task are defined in the IDOL Speech Server tasks configuration file.
requiredParams	You must supply *requiredParams* for the action you request. (Not all tasks have required parameters.)
optionalParams	You can supply *optionalParams* for the action you request. (Not all tasks have optional parameters.)

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&File=Speech.wav&Out=Text.ctm
```

This action creates a SpeechToText task on the IDOL Speech Server located on the local machine, with the ACI port 13000.

Run Tasks Across Multiple Cores

Some tasks can run across multiple cores for faster performance. The audio to process is split into chunks, which are shared out between the IDOL Speech Server task managers (HPE recommends that you set one task manager for each core). Each task manager processes its allocated chunks, and the results from all task managers are combined at the end.

The following modules support multicore processing. If you request multicore processing for a task that contains modules that are not in this list, IDOL Speech Server returns an error.

audiopreproc	frontend	postproc
audio	mixer	stt
ctm	normalizer	wout
filter	plh	

When using the stt module, multiple core processing is semantically meaningful only in fixed mode; relative mode is not supported.

Each running task manager counts as an IDOL Speech Server instance. If insufficient licenses are available for the task, IDOL Speech Server returns an error.

Multicore processing is not supported if server queuing is enabled (see Configure Task Queues, on page 63).

To run a task on multiple cores, you must set the following three action parameters when you send the AddTask action.

TaskManagers	The number of task managers to split the task across. If you specify more task managers than are available, IDOL Speech Server returns an error.

`SplitSize`	The size of the chunks, in seconds, to divide the audio into. For most tasks, HPE recommends between 60 and 300 seconds.
`Overlap`	The number of seconds of overlap between chunks. Increasing the overlap improves accuracy but reduces the overall processing speed. For most tasks, HPE recommends an overlap of about 5 seconds.

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&File=Speech.wav&Out=Text.ctm&TaskManagers=3
```

This action creates a `SpeechToText` task across three task managers on the IDOL Speech Server located on the local machine, with the ACI port `13000`.

Check the Status of a Task

To check the status of a task, send a `GetStatus` action.

`http://`*`host`*`:`*`port`*`/action=GetStatus&Token=`*`token`*

where:

`host`	is the IP address or name of the machine where IDOL Speech Server is installed.
`port`	is the ACI port by which actions are sent to IDOL Speech Server (set by the `Port` parameter in the IDOL Speech Server configuration file's `[Server]`section).
`token`	is the token of the task for which you want to check the status.

For example:

```
http://localhost:13000/action=GetStatus&Token=MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEyOTk0OTU3Ng==
```

This action checks the status of the task with token `MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEyOTk0OTU3Ng==` on the IDOL Speech Server located on the local machine with the ACI port `13000`.

The `GetStatus` action returns task information, including the current task status, settings, and warnings. For more information, see the *IDOL Speech Server Reference*.

If you enabled queuing, you can add the `ShowQueue` parameter to the `GetStatus` action. If the task is queued, the status information returns the position in the queue. For example:

```
http://localhost:13000/action=GetStatus&Token=MTAuMS4zLjgyOjEzMDARFRBU0s6LTEyOTk0OTU3Ng==&ShowQueue=True
```

Related Topics

- Configure Task Queues, on page 63

End a Task

To end a specified task, send an `AbortTask` action.

`http://host:port/action=AbortTask&Token=token`

where:

host	is the IP address or name of the machine where IDOL Speech Server is installed.
port	is the ACI port by which actions are sent to IDOL Speech Server (set by the `Port` parameter in the IDOL Speech Server configuration file's `[Server]`section).
token	is the token of the task that you want to stop.

For example:

```
http://localhost:13000/action=AbortTask&Token=MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEyOTk0OTU3Ng==
```

This action ends the task with token `MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEyOTk0OTU3Ng==` on the IDOL Speech Server located on the local machine with the ACI port `13000`.

Queue Tasks

By default, IDOL Speech Server refuses tasks if the resources are not available. However, you can enable queuing so that tasks are added to a queue and processed when the resources become available.

Related Topics

- Configure Task Queues, on page 63

Schedule Tasks

You can configure IDOL Speech Server to run tasks on a schedule. For example, if you want to regularly update your language models with new data, you can set up a schedule to run the `LanguageModelBuild` action regularly.

You schedule tasks either by using the `[Schedule]` configuration section, or by sending the `ScheduleTask` action:

- Configure Schedules, below
- Schedule a Task by Sending an Action, on page 90

> **NOTE:**
> Scheduling a task does not impose additional limits on the languages supported.
>
> However, you might need to consider existing limits for certain tasks. For example, for `LanguageModelBuild`, some languages do not support text normalization, so you must normalize the text before running the `LanguageModelBuild` task.

Configure Schedules

You can configure schedules by enabling the schedule in the IDOL Speech Server configuration file and adding a schedule file that defines the task schedule.

Create a Schedule File

The schedule file is a text file that contains the details of the schedules. Each line of the schedule file contains the details of a particular task schedule.

> **TIP:**
> You can add comment lines by starting the line with a pound sign (#).

Specify the task details in the following format:

Name;Hours;Minutes;Task_Type;Task_Parameters

where

Name	is the name of the schedule.
Hours	are the hours (0-23) that you want to run the schedule. Separate multiple hours with commas. If you want to run the task every hour, set this value to an asterisk (*).
Minutes	are the minutes (0-59) in the specified hour that you want to run the schedule. Separate multiple minutes with commas.
Task_Type	is the type of task that you want to run. This value is the same as the `Type` parameter for the `AddTask` action.
Task_Parameters	are the parameters for the task that you want to run. Specify parameters in the same way that you use in the `AddTask` action string. For details of the available task parameters, see the *IDOL Speech Server Reference*.

For scheduled tasks, you can use the following special placeholder tags in the task parameters to add an appropriate value for each schedule run, rather than a fixed value for all runs:

`${NAME}`	The schedule label.
`${ITERATION}`	The iteration of this scheduled task being run.
`${YEAR}`	The year that the task runs (YYYY).
`${MONTH}`	The month that the task run runs (1-12).
`${DAY}`	The day that the task runs (1-31).
`${HOUR}`	The hour that the task runs (0-23).
`${MINUTE}`	The minute that the task runs (0-59).

For example:

```
BreakingNewsENUK;*;0,30;Type=LanguageModelBuild&Lang=ENUK&ContentServer=content1.example.com&ContentDatabase=news&NewLanguageModel=${NAME}-${HOUR}${MINUTE}.tlm
```

This example schedule runs a `LanguageModelBuild` task at 0 and 30 minutes past each hour. Each time the schedule runs, it creates a new language model with a name that follows the template `${NAME}-${HOUR}${MINUTE}.tlm` (for example `BreakingNewsENUK-1230.tlm`).

Enable the Schedule in the Configuration File

After you create the schedule file, you must enable the schedule in the IDOL Speech Server configuration file.

To enable a schedule

1. Open your configuration file in a text editor.
2. Find the `[Schedule]` section, or create one if it does not exist.
3. Set `EnableSchedule` to **`True`**.
4. Set `ScheduleFile` to the path and file name of the file that contains your schedule details. You can specify an absolute or relative file path. If you specify a relative path, it is considered as being relative to the IDOL Speech Server installation directory.
5. Save and close the configuration file.
6. Restart IDOL Speech Server to implement your changes. IDOL Speech Server starts to run the schedule at the next scheduled time.

For example:

```
[Schedule]
EnableSchedule=True
ScheduleFile=C:\SpeechServerFiles\Schedule\speechserver-schedule.txt
```

Schedule a Task by Sending an Action

To schedule a task, send an `ScheduleTask` ACI action to your IDOL Speech Server host and ACI port.

```
http://
host
:
port
/action=AddTask&ScheduleName=
MyScheduleName&ScheduleTime=Hours;Minutes&Type=task&TaskParameters
```

where:

host	is the IP address or name of the machine where IDOL Speech Server is installed.
port	is the ACI port by which actions are sent to IDOL Speech Server (set by the `Port` parameter in the IDOL Speech Server configuration file's `[Server]`section).
ScheduleName	is the name that you want to give the new schedule.
Hours	are the hours that you want to run the schedule (0-23). Separate multiple hours with commas. If you want to run the schedule every hour, set *Hours* to an asterisk (*).
Minutes	are the minutes that you want to run the schedule (0-59). Separate multiple minutes with commas.

task	is the name of the configuration section where the options for the task are defined in the IDOL Speech Server tasks configuration file.
TaskParameters	are the parameters (required and optional) for the task that you want to schedule.

For example:

```
http://localhost:13000/action=AddTask&ScheduleName=BreakingNewsENUS&ScheduleTime=*;0,30Type=LanguageModelBuild&ContentServer=content1.example.com&ContentDatabase=news&NewLanguageModel=${NAME}-${HOUR}${MINUTE}.tlm
```

This action creates a schedule to run a `LanguageModelBuild` task at 0 and 30 minutes past every hour. The `LanguageModelBuild` contacts an IDOL Content component to retrieve the training data, and creates a new language model with a name in the format `${NAME}-${HOUR}${MINUTE}.tlm`, which uses the schedule name and the run time in the file name.

> **IMPORTANT:**
> If you use the `ScheduleTask` action to add a schedule, IDOL Speech Server does not update the configured schedule file. Schedules that you add in an action are available only until you stop the server.

Standard Tasks

This table describes the standard tasks that are already defined in the tasks configuration file. You can run any of these tasks straight out of the box.

Task	Description
`AfpAddTrack`	Adds a new audio track to an audio fingerprint database.
`AfpDatabaseInfo`	Returns a list of all tracks that are currently stored in the specified database.
`AfpDatabaseOptimize`	Optimizes the internal indexing of the specified database. This task permanently removes files that have been tagged for deletion using the `AfpRemoveTrack` task, and optimizes lookup functions for newly added tracks.
`AfpMatch`	Receives audio data in an audio file or stream, and searches it for any indexed audio sections.
`AfpRemoveTrack`	Removes specified tracks from an audio fingerprint database.
`AudioAnalysis`	Runs all the audio preprocessing tasks that are supported by the `audiopreproc` module in a single task.
`AudioSecurity`	Detects and labels segments of audio that contain alarms, screams, breaking glass, or gunshots.
`ClippingDetection`	Analyzes an audio file for the issue of audio clipping.
`ClusterSpeech`	Clusters wide-band speech into speaker segments.

Task	Description
ClusterSpeechTel	Clusters telephony speech into speaker segments.
ClusterSpeechToTextTel	Clusters two speakers in a phone call, and uses the resulting speaker clusters to improve speech-to-text performance slightly by using speaker-sided acoustic normalization. Any telephony artifacts such as dial tones or DTMF tones are included, interspersed with the recognized words.
CombineFMD	Combines several phoneme track files, which can then be used for phrase search.
CreateFMD	Creates a phoneme time track (.fmd) file from a single audio file.
DialToneIdentification	Detects and identifies dial tones in audio.
IvSpkId	Performs iVector-based speaker identification on an audio file or stream.
IvSpkIdDevel	Processes one or more speaker ID feature files to generate scores for tuning iVector-based score thresholds.
IvSpkIdDevelAudio	Processes an audio file or stream to generate scores for tuning iVector thresholds.
IvSpkIdDevelFinal	Calculates the iVector score threshold based on one or more development files, and generates a new set of iVectors with the thresholds.
IvSpkIdFeature	Uses an audio file that contains sample speech from one person to create speaker ID feature files for use in iVector based template training, and template score threshold development.
IvSpkIdEditThresh	Modifies the threshold of a template file or a single template stored in an iVector template set file.
IvSpkIdSetInfo	Produces a log file that lists the contents of the specified iVector template file or template set file.
IvSpkIdSetAdd	Adds a number of iVector speaker templates to a single speaker set file.
IvSpkIdSetDelete	Removes a speaker template from an iVector template set file.
IvSpkIdTrain	Takes one or more speaker ID feature files containing speech data from the speaker to be trained, and creates a new iVector speaker template.
IvSpkIdTrainAudio	Takes an audio file or stream containing speech data from the

Task	Description
	speaker to be trained, and creates a new iVector speaker template file.
`LangId`	Receives audio data from a file or stream, converts it into language identification features, and identifies the languages.
`LangIdFeature`	Converts audio files in the relevant language into language identification feature (.lif) files, which are required for training classifiers.
`LangIdOptimize`	Optimizes the balance between language classifiers. After training, some classifiers might be stronger than others because of properties of the training material and the languages in question. The optimization process weights the language models so that weaker languages have increased accuracy, without compromising accuracy for stronger language models. This process improves consistent performance.
`LangIdTrain`	Reads in a set of language ID feature files created from audio representing a single language (using the `LangIdFeature` task), and uses this data to train a new language classifier.
`LanguageModelBuild`	Builds a new language model from a set of text files.
`LMListVocab`	Lists the most common words in the specified language model.
`LMLookUp`	Verifies whether a specified word is present in the vocabulary of a particular language model, and, if it is present, how frequently it occurs.
`LMPerplexity`	Analyzes the perplexity of a sample text file, when given a specific language model.
`PhraseSearch`	Searches for a specified phrase or phrases in an audio file.
`PunctuateCtm`	Adds punctuation to any `.ctm` file.
`Scorer`	Scores the recognition transcript (such as that generated by the `SpeechToText` task), when given a reference transcript file.
`SearchFMD`	Searches a phoneme track file for one or more specified phrases.
`SegmentText`	Inserts whitespace between words in a text file (for languages that do not separate words with whitespace).
`SNRCalculation`	Calculates SNR levels across an audio file.

Task	Description
SpeechSilClassification	Segments an audio file into sections of speech, non-speech, and music.
SpeechToText	Converts an audio file or stream into a text transcript.
SpeechToTextFilter	Converts an audio file or stream into a text transcript and categorizes the audio so that you can remove any sections consisting of music or noise.
SpeechToTextTelephony	Transcribes a telephony audio file or stream, including dial tones and DTMF dial tones.
TextNorm	Takes a raw text transcription file and produces a normalized form (by removing punctuation, rewriting numbers as words, altering word cases, and so on).
TranscriptAlign	If a transcript is available for an audio recording, the transcript alignment function can place time locations for each word in the transcript. You can use this function to align subtitles with audio or video files.
TranscriptCheck	Checks how well a text transcript matches the audio data, and identifies large missing or erroneous sections.

For details about each task, including the required action and configuration parameters, see the *IDOL Speech Server Reference*.

Deprecated Tasks

The following table describes tasks that are included in the IDOL Speech Server configuration file for backwards compatibility. These tasks are deprecated, and might be deleted in future.

Task	Description
AfpAddTrackStream	Adds a new audio track to an audio fingerprint database, receiving the audio data as a stream, and converting to AFP features before indexing. Use AfpAddTrack.
AfpAddTrackWav	Adds a new audio track to an audio fingerprint database, reading the data from an audio file, and converting to AFP features before indexing. Use AfpAddTrack.
AfpMatchStream	Receives audio data as a binary stream, and searches it for any indexed audio sections. Use AfpMatch.
AfpMatchWav	Reads in data from an audio file, and searches it for any indexed audio sections. Use AfpMatch.
AfptAddTrackStream	Performs the same task as AfpAddTrackStream, but uses a template database (fptdb), which improves robustness to audio mismatches

Task	Description
	at the cost of scalability. Use `AfpAddTrack`.
`AfptAddTrackWav`	Performs the same task as `AfpAddTrackWav`, but uses a template database (`fptdb`), which improves robustness to audio mismatches at the cost of scalability. Use `AfpAddTrack`.
`AfptDatabaseInfo`	Performs the same task as `AfpDatabaseInfo`, but uses a template database (`fptdb`), which improves robustness to audio mismatches at the cost of scalability. Use `AfpDatabaseInfo`.
`AfptMatchStream`	Performs the same task as `AfpMatchStream`, but uses template-based matching as opposed to landmarks, which improves robustness to audio mismatches at the cost of scalability. Use `AfpMatch`.
`AfptMatchWav`	Performs the same task as `AfpMatchWav`, but uses template-based matching as opposed to landmarks, which improves robustness to audio mismatches at the cost of scalability. Use `AfpMatch`.
`AfptRemoveTrack`	Performs the same task as `AfpRemoveTrack`, but uses a template database (`fptdb`), which improves robustness to audio mismatches at the cost of scalability. Use `AfpRemoveTrack`.
`AmTrain`	Presents training audio and transcription data to the acoustic model training process, and creates accumulator files that are used to produce a final adapted acoustic model.
`AmTrainFinal`	Produces the adapted acoustic model, given a set of accumulator files created by the `AmTrain` task.
`DataObfuscation`	Prepares training data with any sensitive or classified information concealed.
`IvSpkIdDevelStream`	Takes a single audio stream, along with the name of the speaker the stream is associated with, and generates scores for tuning iVector thresholds. Use `IvSpkIdDevelAudio`.
`IvSpkIdDevelWav`	Processes a single audio file to generate scores for tuning iVector thresholds. Use `IvSpkIdDevelAudio`.
`IvSpkIdEvalStream`	Runs iVector-based identification of any sections of an audio stream where the trained speakers are present. Use `IvSpkId`
`IvSpkIdEvalWav`	Performs iVector-based speaker identification on a single audio file. Use `IvSpkId`
`IvSpkIdSetEditThresh`	Modifies the threshold of a single template stored in an iVector template set file. Use `IvSpkIdEditThresh`.
`IvSpkIdSetInfo`	Produces a log file that lists the contents of the specified iVector template set file. Use `IvSpkIdInfo`.

Task	Description
IvSpkIdTmpEditThresh	Modifies the threshold of a single iVector template file. Use IvSpkIdEditThresh.
IvSpkIdTmpInfo	Produces a log file that lists the contents of the specified iVector template file. Use IvSpkIdInfo.
IvSpkIdTrainStream	Takes a single audio stream containing speech data from the speaker to be trained, and creates a new iVector speaker template file. Use IvSpkIdTrainAudio.
IvSpkIdTrainWav	Takes a single audio file containing speech data from the speaker to be trained, and creates a new iVector speaker template file. Use IvSpkIdTrainAudio.
LangIdBndLif	Reads in language identification features from file and determines boundaries in the feature sequence where the language changes. Returns the language identification results between boundaries. Use LangId.
LangIdBndStream	Receives audio data as a binary stream, converts the audio into language ID features, and determines boundaries where the language changes. Returns the language identification results between boundaries. Use LangId.
LangIdBndWav	Reads in data from an audio file, converts it into language ID features, and determines boundaries where the language changes. Returns the language identification results between boundaries. Use LangId.
LangIdCumLif	Reads in language ID features from file, and returns the running language identification score at periodic intervals (that is, the score for all the input data from the start to the current point). Use LangId.
LangIdCumStream	Receives audio data as a binary stream, and converts it into language ID features. Returns the running language identification score at periodic intervals (that is, the score for all the input data from the start to the current point). Use LangId.
LangIdCumWav	Reads in data from an audio file, and converts it into language ID features. Returns the running language identification score at periodic intervals (that is, the score for all the input data from the start to the current point). Use LangId.
LangIdSegLif	Reads in language ID features from file, processes the data in fixed-sized chunks, and returns the language identification results for each chunk. Use LangId.
LangIdSegStream	Receives audio data as a binary stream, and converts it into language ID features. Processes the data in fixed-sized chunks, and returns the language identification results for each chunk. Use LangId.
LangIdSegWav	Reads in data from an audio file and converts it into language ID

Task	Description
	features. Processes the data in fixed-sized chunks and returns the language identification results for each chunk. Use `LangId`.
`SegmentWav`	Attempts to segment audio into sections by speaker even if no trained speakers exist in the system. Use `ClusterSpeech`.
`SpkIdDevel`	Processes speaker ID feature files to generate scores for tuning model thresholds. Use `IvSpkIdDevel`.
`SpkIdDevelFinal`	Estimates the thresholds for a set of speaker templates. Use `IvSpkIdDevelFinal`.
`SpkIdDevelStream`	Creates or updates a development (`.atd`) file for an audio stream. Use `IvSpkIdDevelAudio`.
`SpkIdDevelWav`	Creates or updates a development (`.atd`) file for an audio file. Use `IvSpkIdDevelAudio`.
`SpkIdEvalStream`	Analyzes an audio stream to identify any sections where the trained speakers are present. Use `IvSpkId`.
`SpkIdEvalWav`	Analyzes an audio file to identify any sections where the trained speakers are present. Use `IvSpkId`
`SpkIdFeature`	Creates a speaker ID feature file. Use `IvSpkIdFeature`.
`SpkIdSetAdd`	Takes one or more audio template files, and adds them to an audio template set file. Use `IvSpkIdSetAdd`.
`SpkIdSetDelete`	Removes a template from an audio template set file. Use `IvSpkIdSetDelete`.
`SpkIdSetEditThresh`	Modifies the threshold of a single template in an audio template set file. Use `IvSpkIdEditThresh`.
`SpkIdSetInfo`	Retrieves information on an audio template set file. Use `IvSpkIdInfo`.
`SpkIdTmpEditThresh`	Modifies the threshold of a single template. Use `IvSpkIdEditThresh`.
`SpkIdTmpInfo`	Retrieves information on an audio template file. Use `IvSpkIdInfo`.
`SpkIdTrain`	Uses one or more feature files to train a speaker template. Use `IvSpkIdTrain`.
`SpkIdTrainStream`	Takes an audio stream containing speech data from the speaker to be trained, and creates a new speaker template file. Use `IvSpkIdTrainAudio`.
`SpkIdTrainWav`	Takes a single audio file containing speech data from the speaker to be trained, and creates a new speaker template file. Use `IvSpkIdTrainAudio`.

Task	Description
StreamToText	Converts live audio into a text transcript. Use SpechToText.
StreamToTextMusicFilter	Converts live audio into a text transcript and categorizes the audio so that you can remove any sections consisting of music or noise. Use SpeechToTextFilter.
TelWavToText	Transcribes a telephony audio file, including dial tones and DTMF dial tones. Use SpeechToTextTelephony.
WavPhraseSearch	Searches for a specified phrase or phrases in an audio file. Use PhraseSearch.
WavToFMD	Creates a phoneme time track (.fmd) file from a single audio file. Use CreateFMD.
WavToPlh	Reads data from an audio file and produces an audio feature (.plh) file, such as those used in the acoustic model adaptation process (the AmTrain task).
WavToText	Converts an audio file into a text transcript. Use SpeechToText. **NOTE:** To use WavToText to submit audio data as a binary data block for speech-to-text, submit the task data without specifying a .wav file.

Display Configured Tasks

When IDOL Speech Server is running, you can use the ListTasks action to retrieve a list of all tasks that are set up in the configuration file.

To display all configured tasks

- Send the ListTasks action to IDOL Speech Server. The action does not require any parameters.

For example:

```
http://localhost:13000/action=ListTasks
```

IDOL Speech Server returns the task list as plain text. For example:

```
afpaddtrack
afpdatabaseinfo
afpdatabaseoptimize
afpmatch
...
```

Display Task Details

To retrieve information about a particular task, including the task schema and available parameters, send a `TaskHelp` action to IDOL Speech Server.

To display information about a task

- Send the `TaskHelp` action to IDOL Speech Server, and set the following parameter:

  `Type` The task name.

For example:

```
http://localhost:13000/action=TaskHelp&Type=SpeechToText
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to return details about the `SpeechToText` task.

IDOL Speech Server returns task information as plain text. For example:

```
Task: speechToText

================================================================
SCHEMA
================================================================
The following table represents the schema that defines the task:

----------------------------------------------------------------
Module          Mode        Inputs      Outputs
----------------------------------------------------------------
audio           MONO        input       a,ts
frontend        _           a           f
normalizer      _           f           nf
stt             _           nf          w
postproc        _           w           w2
wout            _           ts,w2       output
----------------------------------------------------------------

For a description of each module, mode, input, and output, refer to the IDOL
Speech Server Reference.

================================================================
PARAMETERS
================================================================
You can set the following action parameters for this task:

-------------------------------------------------------------------------------
Corresponding
Action Parameter        Configuration Parameter         Type            Default
-------------------------------------------------------------------------------
```

```
inputType               audio.inputtype                 string          file
file                    audio.file                      file
sugdInputFrequency      audio.sugdinputfrequency        int             0
sugdInputChannels       audio.sugdinputchannels         string
startTime               audio.starttime                 real            0.00
endTime                 audio.endtime                   real            0.
.....................................................................................
mode                    stt.mode                        string          fixed
modeValue               stt.modevalue                   real            4
lang                    stt.lang                        resource        ENUK
conf                    stt.enableconfidence            bool            true
dnnScale                stt.dnnscale                    real            0.0
diag                    stt.diag                        bool            false
diagFile                stt.diagfile                    file            $outfile
latFile                 stt.latfile                     file
latWordFile             stt.latwordfile                 file
latScale                stt.latscale                    real            0.5
latWinSize              stt.latwinsize                  int             0
frameDupl               stt.framedupl                   int             0
speedBiasLevel          stt.speedbiaslevel              int             0
.....................................................................................
punctuation             postproc.dopunctuation          bool            false
wordBar                 postproc.dowordbarring          bool            false
wordBarList             postproc.barredlist             file
forceRecompoundOn       postproc.forcerecompoundon      bool            false
.....................................................................................
out                     wout.output                     file            $outfile
-------------------------------------------------------------------------------------

Depending on the resources that the task uses, there might be additional
parameters that are not listed here.

For a description of each parameter, refer to the IDOL Speech Server Reference.
```

> **NOTE:**
> If a task uses one or more resources, you can sometimes set additional parameters depending on the resource configuration. Sometimes the additional parameters are mandatory. The additional parameters are determined after you have specified the resources for a task, therefore they do not appear in the Parameters table.

Related Topics

- Display Online Help, on page 84
- Configure Custom Tasks, on page 235

Display Configured Resources

Use the following procedure to display details of all resources that are configured and available for use in tasks.

To display all configured resources

- Send the `ListResources` action to IDOL Speech Server. To filter the list of resources that IDOL Speech Server returns, set any of the following optional parameters:

`Found`	Whether to exclude languages that do not have lang files on disk from the results.
`LangCode`	A language code to restrict results to (for example, ENUK).
`Licensed`	Whether to exclude unlicensed languages from the results.
`Type`	A resource type to restrict results to (`lang`, `langvt`, `fpdb`, or `sidbase`).

For example:

```
http://localhost:13000/action=ListResources&Type=lang&Found=True
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to return a list of all configured lang resources that are on disk.

IDOL Speech Server returns resource details in XML format. For example:

```
<autnresponse>
     <action>LISTRESOURCES</action>
     <response>SUCCESS</response>
     <responsedata>
         <resources>
               <resource>
                   <name>ENUK</name>
                   <type>lang</type>
                   <found>TRUE</found>
                   <licensed>TRUE</licensed>
                   <lang>ENUK</lang>
                   <sampleRate>16000</sampleRate>
               </resource>
               <resource>
                   <name>ENUS</name>
                   <type>lang</type>
                   <found>TRUE</found>
                   <licensed>TRUE</licensed>
                   <lang>ENUS</lang>
                   <sampleRate>8000</sampleRate>
               </resource>
       </resources>
     </responsedata>
 </autnresponse>
```

For each resource, IDOL Speech Server returns the following information.

`name`	The name of the resource as specified in the configuration file.
`type`	The resource type (`lang`, `langvt`, `fpdb`, or `sidbase`).
`found`	Whether the resource was found on disk.
`licensed`	Whether a license to use the resource is available. IDOL Speech Server returns the value `Unknown` if it cannot determine the resource language code.
`lang`	The language code associated with the resource. IDOL Speech Server returns the value `Unknown` if it cannot determine the resource language code, or `N/A` for resource types that are language independent.
`sampleRate`	The sample rate associated with the resource. IDOL Speech Server returns the value `Unknown` if it cannot determine the sample rate.

Check Available Resources

You can verify whether IDOL Speech Server has all the necessary resources for a task before you start the task. For example, if you run multiple installations of IDOL Speech Server, you can use the `CheckResources` action to check which one can best process a task.

`http://`*`host`*`:`*`port`*`/action=CheckResources&Type=`*`task`*`&`*`requiredParams`*`&`*`optionalParams`*

where:

`host`	is the IP address or name of the machine where IDOL Speech Server is installed.
`port`	is the ACI port by which actions are sent to IDOL Speech Server (set by the `Port` parameter in the IDOL Speech Server configuration file's `[Server]` section).
`task`	is the name of the configuration section that define the options for the task in the IDOL Speech Server tasks configuration file.
`requiredParams`	are the parameters that you must supply for the action you request. (Not all tasks have required parameters.)
`optionalParams`	are the parameters that you can supply for the action you request. (Not all tasks have optional parameters.)

You must specify all parameters that you want use to perform the task. If you intend to run the task across multiple cores (see Run Tasks Across Multiple Cores, on page 86), you must set the `TaskManagers` parameter.

`TaskManagers`	The number of task managers to split the task across. If you specify more task managers than are available, IDOL Speech Server returns an error.

> **NOTE:**
> You do not need to set the `SplitSize` or `Overlap` parameters for the `CheckResources` action.

The `CheckResources` action returns one of the following status messages.

`AVAILABLE_`	The language pack is already loaded and the server can accept your task.

LOADED

AVAILABLE_NOT_LOADED	The language pack is either already loaded or the task does not use one, so the server can accept your task.
NOT_AVAILABLE	The server has either reached the maximum number of tasks it can run in parallel or the maximum number of languages it can load. The task is rejected if submitted.

For example:

```
http://localhost:13000/action=CheckResources&Type=SpeechToText&File=Speech.wav&Out=Text.ctm
```

This action checks the server resources for a IDOL Speech Server on the local machine using ACI port `13000`. It checks resources for the speech-to-text task defined in the `[SpeechToText]` section of the IDOL Speech Server tasks configuration file, on the file `Speech.wav`.

> **NOTE:**
> If you enabled queuing, the `CheckResources` action is not available.

> **NOTE:**
> The results of a `CheckResources` action might not continue to be valid if you subsequently submit new tasks to the server. In addition, `CheckResources` does not consider any pending tasks that might be in the system (that is, tasks that have been submitted using the `AddTask` action, but are not yet fully registered with the task tracker). These tasks are shown as `PENDING` in the `GetStatus` output.
>
> A|fter you submit a task, HPE recommends that you use the `GetStatus` action with the task token to check the status of the task. Do not make any calls to `CheckResources` for subsequent tasks until the task status changes from `PENDING` to `STARTING`, `RUNNING`, `LOADING_LM`, and so on.

Find Recommended Language Resources to Use in a Task

IDOL Speech Server can recommend language packs to use with a particular audio file, if provided with the file and a language code. This is helpful if multiple languages are defined for a single language code, or if you are not sure whether to use a broadband or telephony pack.

For IDOL Speech Server to recommend a language pack:

- You must configure the language pack in the tasks configuration file (see Configure Language Packs, on page 62).
- All required language pack files must be available on disk.
- The language must be licensed.
- The language pack must have a sample frequency equal to, or lower than, the sample frequency of the audio file (IDOL Speech Server does not permit upsampling).

To find language packs to use with a file

- Send the `ResourceSelection` action to IDOL Speech Server, and set the following action parameters:

`File` The audio file to find a language pack for.

`Lang` The language code. You can search using the base language code (for example, `EN`) or the full code (for example, `ENUS`).

For example:

```
http://localhost:13000/action=ResourceSelection&Lang=ENUK&File=C:\data\sample.wav
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to recommend British English language packs that you can use with the `sample.wav` file.

The action returns a list of language packs that you can use in tasks that process the specified file. For example:

```
<autnresponse>
	<action>RESOURCESELECTION</action>
	<response>SUCCESS</response>
	<responsedata>
		<audioRate>16000</audioRate>
		<resourceRate>16000</resourceRate>
		<resources>
			<resource>enuk</resource>
			<resource>enuk-news</resource>
		</resources>
	</responsedata>
</autnresponse>
```

In this example, IDOL Speech Server suggests the standard `ENUK` language pack and a pack that contains the custom `news` language model.

If you use the base code, IDOL Speech Server returns all language packs that cover the base language that you specified.

For example:

```
http://localhost:13000/action=ResourceSelection&Lang=EN&File=C:\data\sample.wav
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to recommend language packs that cover the base language `EN` that you can use with the `sample.wav` file. For example:

```
<resources>
    <resource>enau</resource>
    <resource>enca</resource>
    <resource>enuk</resource>
    <resource>enus</resource>
</resources>
```

The tasks configuration file contains language sections for each of the suggested resources.

> **NOTE:**
> The names of the resource configuration sections might be in a different case to the names returned by the `ResourceSelection` action.

If IDOL Speech Server cannot find any suitable language packs, it still returns a `SUCCESS` response (the action itself did not fail) but includes an explanatory message. For example:

```
<autnresponse>
	<action>RESOURCESELECTION</action>
	<response>SUCCESS</response>
	<responsedata>
		<audioRate>16000</audioRate>
		<details>
		No suitable language packs found (lang code zhtw, with sample rate 16000)
		</details>
	</responsedata>
</autnresponse>
```

Similarly, IDOL Speech Server provides details if it finds appropriate language packs, but they are unavailable for use (for example, unlicensed packs). If IDOL Speech Server finds appropriate language packs, but they are for the wrong sample frequency (for example, 8 kHz telephony packs for a 16 kHz audio file), IDOL Speech Server lists the packs but includes a warning in the `details` section.

If the action fails (for example, because the audio is missing or in the wrong format), IDOL Speech Server returns an error message.

Get Task Results

Use the following procedure to retrieve task results from IDOL Speech Server.

To retrieve task results

- Send the `GetResults` action, and set one of the following parameters:

  `Token` The action token for the task.

  `File` The results file that you want to return.

  > **NOTE:**
  > If you set the `File` parameter, IDOL Speech Server returns results in `raw` format. To return results in a different format, set the `Format` parameter.

  You can set additional parameters. (The exact parameters available depend on the task type. For more information, see the *IDOL Speech Server Reference*.)

  `First` The number of results to return, starting at the beginning of the results file.

  `Format` The results format. Set this parameter to override the default output format for a task.

  `Label` The label of the output file to return, if the task produces more than one file. By default, IDOL Speech Server returns the results file specified in the task's `DefaultResults` configuration parameter.

  `Last` The number of results to return, starting from the end of the results file.

  `NBest` The number of highest-scoring results to return.

`Start` The position of the first result to return.

`StartTime` The time of the first result to return. You can use this parameter for output from the `stt`, `lid`, and `lib` modules.

For example:

```
http://localhost:13000/action=GetResults&Token=MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEyOTk0OTU3Ng==
```

```
http://localhost:13000/action=GetResults&File=C:\softsoundserver\results\out.ctm
```

Results Format

IDOL Speech Server tasks produce results in different file formats. You can change the format that results are displayed in whenever you use the `GetResults` action to retrieve results. To change the display format, set the `Format` parameter in the action. The value that you can set depends on the task and the results file format.

For all tasks and file formats, you can set `Format` to one of the following values:

`bin` Binary data read directly from disk.

`raw` XML that contains text lines directly from the results file.

`txt` Plain text read directly from disk.

For any task that generates an XML file, the following value is also available:

`xml` XML data read directly from disk.

For some tasks, other values are also available. These tasks must generate CTM files, otherwise the values are unavailable.

Task	Parameter value	Description
Audio fingerprint identification (`AFPMatch`)	`afp`	XML derived from an AFP CTM file.
Language identification (`LangId`)	`lid`	XML derived from a language identification CTM file.
Speech-to-text (`SpeechToText`, `SpeechToTextFilter`, `SpeechToTextTelephony`, `ClusterSpeechToTextTel`)	`stt`	XML derived from a speech-to-text CTM file.

Continuation Tokens

When you retrieve results in `stt`, `lid`, or `lib` format, the server response includes the `<continueToken>` XML tag.

You can include the value of the `continueToken` tags in a `GetResults` action to return only results generated after the set of results that produced the `continueToken`.

For example, the following action:

```
http://localhost:13000/action=GetResults&Token=MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEy
OTk0OTU3Ng==
```

might produce the following XML response:

```
<autnresponse>
	<action>GETRESULTS</action>
	<response>SUCCESS</response>
		<responsedata>
			<stt_transcript>
				<stt_record>
					<start>0.000</start>
					<end>3.390</end>
					<label><s></label>
					<score>0.394</score>
					<rank>0</rank>
				</stt_record>
				<stt_record>
					<start>3.390</start>
					<end>3.780</end>
					<label>Hello</label>
					<score>0.344</score>
					<rank>0</rank>
				</stt_record>
			</stt_transcript>
			<continueToken>OjEzMDAwOkFs6LTEyOTk0OTU3Ng==</continueToken>
			<file>C:/temp/test.ctm</file>
		</responsedata>
</autnresponse>
```

You can then use the following action to return all results from 3.78 seconds onwards:

```
http://localhost:13000/action=GetResults&Token=MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEy
OTk0OTU3Ng==&ContinueToken=OjEzMDAwOkFs6LTEyOTk0OTU3Ng==
```

The results of this action also include a new `ContinueToken` that you can use to retrieve more results as IDOL Speech Server generates them.

Delete Output Files

IDOL Speech Server generates different types of output files (for example, diagnostics files, language model files, acoustic model files, and so on). You can delete these output files in several ways.

Use ClearTmpResults Action

You can use the `ClearTmpResults` action to request the deletion of **all** output files generated by IDOL Speech Server in the temporary directory.

To clear all temporary files

- Send the following action:

```
http://localhost:13000/action=ClearTmpResults
```

The action requests the deletion of all output files generated by IDOL Speech Server that are located in the temporary directory. Information on the number of files that were deleted and the number of files that were not deleted (because they are currently being used by running processes or shared by other tasks) is written to the server log once the request has been processed.

Use DeleteResults Action

You can also use the `DeleteResults` action to request the deletion of any output files, not just temporary files. You can optionally specify the following parameters:

- `TempOnly`. Set this parameter to **`True`** to delete only files in the temporary directory.
- `Token`. Set this parameter to a task token so that you delete only files associated with a specific task.
- `Task`. Set this parameter to delete only files associated with a specific task type (for example, `SpeechToText`).
- `Label`. Set this parameter to the output label associated with the file or files to be deleted (for example, `Diag`).

For example:

```
http://localhost:13000/action=DeleteResults&TempOnly=True&Token=MTAuMS4zLjgyOjEzMDAwOkFERFRBU0s6LTEyOTk0OTU3Ng==
```

This example requests the deletion of all files in the temporary directory that are associated with the specified task token.

```
http://localhost:13000/action=DeleteResults&Task=SpeechToText
```

This example requests the deletion of all output files produced by `SpeechToText` tasks.

Information on the number of files that were deleted and the number of files that were not deleted (because they are currently being used by running processes or shared by other tasks) is written to the server log once the request has been processed.

Use DeleteResult Action Parameter with GetResults Action

When you request to view a temporary results file, you can request to delete it at the same time. To do this, include the `DeleteResult` action parameter in the action, and set it to **`True`**.

```
http://localhost:13000/action=GetResults&Token=...&DeleteResult=True
```

This action deletes the file that the `GetResults` action is accessing. For example, if the `GetResults` action is accessing a file that contains diagnostics information, the action deletes that file if you set `DeleteResult` to **`True`**.

The action returns `ResultDeleted=True` if the file has been deleted, and `ResultDeleted=False` if the file is still being used by the process.

Monitor the Status of the Output File Manager

You can use the `GetOutFileStatus` action to view the status of output file tracking in the server, and view information on whether the file manager is idle or processing a request, the number of queued requests, and the number of output files being tracked. For more information, see the *IDOL Speech Server Reference*.

Restrictions on File Deletion

In all the three methods mentioned previously, certain restrictions might prevent you from deleting a file:

- If a file is being used by a task that is running or queued, IDOL Speech Server does not delete the file.
- If a file is associated with several tasks (for example, if multiple tasks had the same output file), IDOL Speech Server does not delete the file until all records or tasks that use that file have been removed.
- If you have configured IDOL Speech Server to prohibit file deletion, IDOL Speech Server does not delete the file. For more information on how to use the `DeleteAllowed` and `DeleteReqsToken` configuration parameters to prohibit file deletion in the configuration file, see the *IDOL Speech Server Reference*.

Create and Manage Lists

Many IDOL Speech Server tasks require you to send an action that includes a list of words or files. For example, the `LanguageModelBuild` task requires a list of training text files. IDOL Speech Server has a list manager that allows you to easily create and manage lists.

> **NOTE:**
> You can also create a list in a text file on a drive that the server can access, instead of using the list manager.

Create and Edit Lists

You first create a list in the list manager, and then populate it by sending each list entry as a separate action. You can delete individual entries or the entire list.

To create a list

- Send the `AddList` action to IDOL Speech Server, and set the following parameter:

  `Key` The list name. The name must consist only of characters included in the Portable Character Set.

  IDOL Speech Server creates a new list with the specified key.

For example:

```
http://localhost:13000/action=AddList&Key=TrainingText
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to create a new list called TrainingText.

To add a list entry

- Send the AddListLine action to IDOL Speech Server, and set the following parameters:

Key	The list name.
Line	The list entry to add.

  IDOL Speech Server adds the list entry to the list with the specified key.

For example:

```
http://localhost:13000/action=AddListLine&Key=TrainingText&Line=
T:\Data\testFile.txt
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to add the entry T:\Data\testFile.txt to the TrainingText list.

To delete a list entry

- Send the DelListLine action to IDOL Speech Server, and set the following parameters:

Key	The list name.
Line	The list entry to delete.
Once (optional)	Set to **True** to delete all instances of the entry in the list.

  IDOL Speech Server returns the NbLines tag that contains the number of lines that IDOL Speech Server has deleted.

For example:

```
http://localhost:13000/action=DelListLine&Key=TrainingText&Line=
T:\Data\testFile.txt&Once=True
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to delete the entry T:\Data\testFile.txt wherever it appears in the TrainingText list.

To delete a list

- Send the DelList action to IDOL Speech Server, and set the following parameter:

Key	The list name.

  IDOL Speech Server deletes the list, unless the list is being used by a task. If the list is in use, IDOL Speech Server returns the message Resource key is busy.

For example:

```
http://localhost:13000/action=DelList&Key=TrainingText
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to delete the TrainingText list.

View Lists

You can view the names of the lists in the list manager, or view the entire contents of an individual list.

To view all lists

- Send the `ShowAllLists` action to IDOL Speech Server.

  IDOL Speech Server returns the names of all lists.

For example:

```
http://localhost:13000/action=ShowAllLists
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to display the names of all lists in the list manager.

To view the contents of a list

- Send the `ShowList` action to IDOL Speech Server, and set the following parameter:

  `Key` The list name.

  IDOL Speech Server returns all entries in the list.

For example:

```
http://localhost:13000/action=ShowList&Key=TrainingText
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to display all entries in the `TrainingText` list.

Use Lists in Actions

After you create and populate your lists, you can use them in IDOL Speech Server actions by setting the relevant parameter to `ListManager/`*`Key`*, where *`Key`* is the name of the list.

For example:

```
http://localhost:13000/action=AddTask&Type=AudioSecurity&File=
C:\data\Sample.wav&Out=SampleSec.ctm&TemplateList=ListManager/alarmTemplates
```

The `AudioSecurity` task requires a list of alarm templates to identify the alarms that it detects. The `TemplateList` parameter specifies the `alarmTemplates` list in the list manager.

Back Up and Restore IDOL Speech Server

You can create a backup of the IDOL Speech Server that you can use to restore the server's state.

The backup includes:

- Copies of the IDOL Speech Server configuration file and tasks configuration file.
- Details of actions that have been queued, are running, and have finished.

To create a backup

- Send the `BackupServer` action to IDOL Speech Server, and set the following parameter:

  `Path` The folder in which to save the backup file.

For example:

```
http://localhost:13000/action=BackupServer&Path=C:/HewlettPackardEnterprise/speechs
erver/backups
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to create a backup file in `C:/HewlettPackardEnterprise/speechserver/backups`.

To restore the server state from a backup

1. Send the `RestoreServer` action to IDOL Speech Server, and set the following parameter:

   `Filename` The path of the backup file.

   IDOL Speech Server returns a response.
2. If the action was successful, restart the server.

Troubleshooting

IDOL Speech Server creates log files, which often contain information useful for debugging. In the event of any problem, first check the log files.

- To confirm that an action was received correctly and with the appropriate parameters, check the `action.log` file.
- To check for any internal IDOL Speech Server error, check the `softsound.log` file.
- To check for licensing quota information, check the `softsound.log` file soon after you start or restart the server.

By default, IDOL Speech Server processes actions that contain unrecognized parameters. The server logs a warning to the `action.log` file but allows the action to continue. To configure IDOL Speech Server to reject actions with unrecognized parameters, set the `AllowUnusedParams` parameter in the `[Server]` configuration section to **`False`**. IDOL Speech Server ends the action and returns an error message.

Part 2: IDOL Speech Server Operations

This section describes how to perform IDOL Speech Server operations.

- Manage Language Packs and Other Resources
- Speech-to-Text
- Create Custom Language Models
- Normalize Text
- Align Transcripts
- Phonetic Phrase Search
- Use Speaker Clustering
- Identify Speakers in Audio
- Audio Fingerprint Identification
- Audio Security
- Stream Live Audio
- Preprocess Audio
- Postprocess Results

Chapter 6: Manage Language Packs and Other Resources

This section describes how to load, unload, and manage language packs and other shared resources that you need for your IDOL Speech Server tasks.

- Overview 115
- Load Language Resources 115
- Unload Language Resources 116
- Monitor Resource Usage 117

Overview

Some tasks in IDOL Speech Server require you to load a shared resource. A single instance of a shared resource in memory can be used simultaneously by one or more tasks.

There are several different resource types:

- **Lang**. Standard language packs (including the language identification base pack).
- **Langvt**. Language packs used for phrase matching.
- **SidBase**. The base packs used for speaker identification.
- **Fpdb**. Audio fingerprinting databases.

> **NOTE:**
> Speaker classifier files are **not** considered as a shared resource, and are not handled in the same way.

Load Language Resources

You can manually load a resource before you start a task. If the resource is not loaded when the task starts, the task automatically loads the resource.

A resource can be loaded only if there are free slots available for that type of resource.

> **NOTE:**
> Whether the resource is loaded manually or automatically affects whether the resource is unloaded after it is no longer needed, as is covered in detail in the next section.

The following parameters control the maximum number of each resource type:

- The **maxLangResources** parameter controls the maximum number of language packs (`Lang` or `Langvt`) that you can load simultaneously.
- The **maxSidResources** parameter controls the maximum number of speaker identification base packs (`SidBase`) that you can load simultaneously.

- The **maxAfpResources** parameter controls the maximum number of audio fingerprint databases (Fpdb) that you can load simultaneously.

If you set any of the parameters to 0, loading of that resource type is unavailable.

If you set any of the parameters to -1, you can load an unlimited number of that resource type, memory permitting.

The more resources that you have loaded at any one time, the more memory the server uses. However, because loading resources can take time, you can save time when running a task for which the required resource is already loaded.

> **NOTE:**
> You shonsider memory usage and load times when you determine the limits that you want to place on the number of resources that can be loaded simultaneously.
>
> Memory usage and load times are significant considerations for language packs. Each language pack might take 10 or more seconds to load, and can take up approximately 700 MB of memory. However, speaker identification base packs are much smaller, and load almost instantly; in addition, it is unlikely that you would need to use more than one speaker identification base pack.
>
> Audio fingerprint databases tend to load considerably faster than language packs, and use significantly less memory. However, performance does depend on the amount of data stored in the database, and the cache size selected for database access.

Related Topics

- Monitor Resource Usage, on the next page
- Check Available Resources, on page 102
- Load a Language Pack Manually , on page 240

Unload Language Resources

The way in which IDOL Speech Server handles resources depends on whether the resource is manually or automatically loaded. For example, if you load a resource manually, it remains in memory until you explicitly unload it. By contrast, if a language pack is loaded automatically when the task that requires that resource starts, the resource might be automatically unloaded after it is no longer needed.

A loaded language resource is considered **active** while it is being used by a running task. After that task finishes (and assuming that there are no other running tasks using the same resource), it is considered **inactive**. If you did not load the resource manually, it might be automatically unloaded. The point at which inactive resources are unloaded is controlled primarily by the minimum resource count settings. You can use the following parameters to set these for the various resource types :

- The **minLanguageResources** parameter sets the minimum number of language packs (Lang and Langvt) to leave loaded.
- The **minSidResources** parameter sets the minimum number of speaker identification base packs (SidBase) to leave loaded.
- The **minAfpResources** parameter sets the minimum number of audio fingerprint databases (Fpdb) to leave loaded.

If the number of loaded resources of a particular type exceeds the threshold specified by the parameters above, IDOL Speech Server unloads inactive resources of that type until the threshold is met.

If you set the minimum resource count parameters to **0**, IDOL Speech Server always unloads any inactive resources of that type (unless they were manually loaded).

If you set the minimum resource count parameters to **-1**, IDOL Speech Server sets the threshold to equal the maximum count for that resource. In this case, a resource is only ever unloaded if a different resource (of the same type) is required, but there are no slots available for it.

> **NOTE:**
> If you set both the maximum and minimum thresholds for a resource to **-1**, IDOL Speech Server **never** unloads loaded resources, and records a warning in the logs when you start the server.

Unload Manually Loaded Resources

You must explicitly unload manually loaded resources. If you use the `UnloadLanguage` action, IDOL Speech Server does not unload the resource immediately by default, but marks it as unloadable. It is then subject to the same process that applies to automatically loaded resources, as described above.

You can also ensure that IDOL Speech Server unloads the language immediately by setting the `Force` parameter to **`True`** in the `UnloadLanguage` action.

Monitor Resource Usage

To view information on any minimum and maximum limits placed on loaded resources, send the `GetStatus` action and check the information in the `<resources>` section of the response.

The `GetStatus` response also includes information about the number of each resource type currently being used, including whether the resource is active, and whether it is loaded; finally, information on each of the loaded and or active resources is given in detail.

For more information, see GetStatus, on page 82.

Chapter 7: Speech-to-Text

This section describes how to use IDOL Speech Server to perform speech-to-text.

- Prepare the Audio 119
- Select the Language Pack 119
- Run the Task 120
- Interpret the Results 121
- Use a Lattice File 123
- Control Speech-to-Text Process Speed 123
- Enable Word Confidence Scores 124
- Tune Parameters 125
- Use a Custom Language Model 125
- Troubleshooting 126
- Score Speech Recognition 127

Prepare the Audio

IDOL Speech Server accepts most media files. However, this does not apply:

- If you are processing streamed audio data. (For details about how to stream audio data into IDOL Speech Server, see Stream Live Audio, on page 203.)
- If the `FFmpegDirectory` configuration parameter is not set.

In these cases, you must supply the audio as either 8 kHz or 16 kHz, mono or stereo 16-bit linear, little-endian PCM data. For details of the audio quality requirements, see Audio Quality Guidelines, on page 279.

Select the Language Pack

For maximum speech-to-text accuracy, you must choose the correct language pack. Select the pack based on the language and sampling rate of the audio. For the complete list of available language packs, see Available language packs, on page 49.

You must configure the language pack in the `[Resources]` section of the tasks configuration file. For more information, see Configure Language Packs, on page 62.

When you load the language model, you can add words to the model and increase the weighting of specific words by using the `ClassWordFile` parameter. You can also add or edit pronunciations by using the `PronFile` parameter. For more information on these parameters and how to use them, see Load a Language Pack Manually , on page 240 and the *IDOL Speech Server Reference*.

Run the Task

IDOL Speech Server provides four preconfigured speech-to-text tasks:

- `SpeechToText`, which performs speech-to-text on an audio file or stream
- `SpeechToTextFilter`, which performs speech-to-text on an audio stream and categorizes the audio so that you can remove any sections categorized as music or noise from the resulting `.CTM` file.
- `SpeechToTextTelephony`, which performs speech-to-text on audio files of telephone conversations. The task also detects and reports dial tones and DTMF dial tones (see DTMF Identification, on page 208).
- `ClusterSpeechToTextTel`, which clusters two speakers in a phone call, and uses the resulting speaker clusters to improve speech-to-text performance slightly by using speaker-sided acoustic normalization. Any telephony artifacts such as dial tones or DTMF tones are included, interspersed with the recognized words.

You can set `Punctuation` to **`True`** in any of these tasks to perform speech-to-text that includes simple sentence-forming punctuation (for example, full stops and initial capital letters) in the `.CTM` file. The speech-to-text task estimates the start and end of the sentence, although this is a best guess only and is not 100% accurate.

> **NOTE:**
> The `Punctuation` parameter should be used only for languages that use the Latin alphabet.

You can use the `SpeedBiasLevel` parameter in any speech-to-text task to quickly set the balance between speed and accuracy in the decoder. By default, `SpeedBiasLevel` is set to **`0`**, which leaves the underlying parameter settings untouched (that is, quick configuration of relevant parameters is disabled). To enable the speed configuration, set `SpeedBiasLevel` to a value between **`1`** (slowest) and **`6`** (fastest). The default speech-to-text parameters are equivalent to a speed bias of **`2`**.

> **NOTE:**
> You can use the `SpeedBiasLevel` functionality only when the speech-to-text mode is **`fixed`** (see Control Speech-to-Text Process Speed, on page 123), and with a DNN-based language resource.

You can also use the `PunctuateCtm` task to add punctuation to any `.CTM` file. For more information, see the *IDOL Speech Server Reference*.

To run speech-to-text on an audio file

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Specify **`SpeechToText`**.

  `File` The audio file to process.
  To restrict processing to a section of the audio file, set the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

  `Out` The file to write the transcription to.

  `Lang` The language pack to use.

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&File=C:/myData/Speech.wav&Out=SpeechTranscript.ctm&Lang=ENUS
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `SpeechToText` task on the `Speech.wav` file and write the results to the `SpeechTranscript.ctm` file. The `Speech.wav` file contains U.S. English dialect speech.

If you are using a lattice file and want to reduce the lattice output size by including only one sample of each word in a specific window size, you can also set the `LatWinSize` parameter. See Use a Lattice File, on page 123 and the *IDOL Speech Server Reference* for more information.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results. The speech-to-text operation produces results while it crunches through the audio data, so you can retrieve results before the task is complete. See Find Recommended Language Resources to Use in a Task, on page 103.

When you use IDOL Speech Server to process multiple data streams or files at the same time, the server might not have enough CPU or memory to process all of them at once. Speech-to-text operation is very CPU-intensive. To check whether a server has sufficient resources to run a `SpeechToText` task, send a `CheckResources` action. See Check Available Resources, on page 102.

Interpret the Results

IDOL Speech Server supports two speech-to-text output formats: the CTM format and the XML format.

CTM Transcript Output

This is a sample of the transcript text in a `.CTM` file:

```
1 A 53.200 0.200 it's 0.000
1 A 53.400 0.060 a 0.000
1 A 53.460 0.340 big 0.000
1 A 53.800 0.720 logistical 0.000
1 A 54.520 0.920 exercise 0.000
```

From left to right, the columns in the output data file contain:

- The channel ID (usually `1`)
- A fixed field, `A`
- The start time of the recognized word in seconds
- The duration of the recognized word in seconds
- The recognized word
- The confidence value for the recognized word

If you have used any speech-to-text tasks with `Punctuation` set to **`True`**, the output appears as follows, with a capital letter and a full stop at the start and end of each sentence within the CTM file:

```
1 A 53.200 0.200 It's 0.000
1 A 53.400 0.060 a 0.000
1 A 53.460 0.340 big 0.000
1 A 53.800 0.720 logistical 0.000
1 A 54.520 0.920 exercise. 0.000
```

> **NOTE:**
> The `Punctuation` parameter should be used only for languages that use the Latin alphabet.

XML Transcript Output

To view the output of a speech-to-text transcription file in XML format, send a `GetResults` action. The contents of the .ctm output file return in a series of XML tags. For example:

```
<stt_transcript>
	<stt_record>
		<start>0.000</start>
		<end>3.390</end>
		<label>&lt;SIL&gt;</label>
		<score>0.987</score>
		<rank>0</rank>
	</stt_record>
	<stt_record>
		<start>3.390</start>
		<end>3.780</end>
		<label>hello</label>
		<score>0.765</score>
		<rank>0</rank>
	</stt_record>
	<stt_record>
		<start>3.780</start>
		<end>3.970</end>
		<label>there</label>
		<score>0.875</score>
		<rank>0</rank>
	</stt_record>
</stt_transcript>
```

This example shows the XML output for a transcript that contains a silent period (with the label `<SIL>`) followed by the words *hello there*.

The `<stt_transcript>` tag represents the start of a recognition sequence. This tag contains `<stt_record>` nodes that contain the following information for each recognized word.

`<start>`	The start time (in seconds) of the word.
`<end>`	The end time (in seconds) of the word.
`<label>`	The recognized word.

`<score>`	The confidence value of the recognized word.
`<rank>`	Not used for speech-to-text (used for different operation results).

Pauses in Recognition Output

When looking at word output data, the two symbols `<SIL>` and `<s>` sometimes appear. These acoustically represent periods of audio without speech, such as silence or background noise. In the former case (`<SIL>`), the recognition process decides this 'silence' probably has no linguistic role. In the latter case (`<s>`), the recognition process decides it is more likely to end a chain of words and start anew at this point. This is analogous to starting a new sentence, but is only a weak analogy since the decision is made only with local context words.

Use a Lattice File

You can record details of word hypotheses that were considered during recognition, not simply the best scoring hypothesis, in a lattice file. You can then search for occurrences of specific words in the lattice to aid recall. In addition, you can restrict the lattice to instances of particular words.

You can also restrict the lattice to a single example of each word in the window that you specify with the `LatWinSize` parameter. Alternatively, you can control the depth of the lattice by specifying that poorly scoring hypotheses must not be included. For more information on setting up and configuring lattice files for use in speech-to-text processing, see the *IDOL Speech Server Reference*.

Control Speech-to-Text Process Speed

Speech-to-text can run in one of two primary modes–`fixed` and `relative`.

In `fixed` mode, the speech-to-text engine attempts to produce a 'uniform quality' transcription, where the transcription is performed to roughly the same quality across the entire speech data regardless of how long it takes. However, the progress of the speech-to-text can fluctuate depending on factors related to the data. For example, noisy or poorly discernible data requires more CPU usage to compute compared with 'clean' data.

In `relative` mode, IDOL Speech Server aims to crunch through the audio data at a uniform pace dictated by a specified target rate.

HPE recommends that for most deployments you use the `fixed` mode unless there is a good reason not to. Although `relative` mode offers time guarantees, recognition quality can sometimes suffer if the CPU cannot keep up with the target rate. However, if you are processing live data, HPE recommends that you use `relative` mode. For information about how to specify the mode, see Use Live Mode for Streaming, on page 204.

The default mode is `fixed`, with a default mode value of `4`. You can specify a mode value between `1` and `4`, trading speed against accuracy. Specifying a value of `1` results in fast processing, with a potentially lower accuracy. Specifying a value of `4` results in the most accurate analysis but can take longer to process.

In the `relative` mode, the mode value can range from `0.5` to `2.0`. These mode values represent the data processing rate compared to real time. A mode value of `1.0` means that the speech data is

processed at almost real time. A mode value of `0.5` processes the data twice as fast as real time. If your target value is relatively high, the process might finish earlier than requested, if the process determines that spending more time on recognition would not give better results.

> **NOTE:**
> If you want to use the `SpeedBiasLevel` functionality, you must run speech-to-text in `fixed` mode. See Run the Task, on page 120 for more information.

Versions of IDOL Speech Server from 10.8 and later and the 6.0+ versions of the language packs use DNN acoustic models to improve speech-to-text accuracy. Each language pack contains at least two (typically three) DNN acoustic models of different sizes. In fixed mode, the default option is to use the largest, most accurate DNN file. If you use relative mode, the default option selects a smaller, faster DNN acoustic file.

IDOL Speech Server version 11.5 and later, and the 9.0+ versions of the language packs, use a new Neural Network technology to improve recognition accuracy. These language packs contain only one DNN file, which it uses in both fixed and relative mode. This option is typically faster than the older DNN technology. The new DNN files also reduce the benefit of using small DNN files to improve speech-to-text processing speed, because other options are now available (such as `FrameDupl`, and `SpeedBiasLevel`).

To change the default settings, use the `DNNFile` command line parameter, or edit the configuration file.

> **CAUTION:**
> You can use DNN acoustic modeling in relative mode only if your DNN files are smaller than a certain size. In addition, you must be using Intel (or compatible) processors that support SIMD extensions SSSE3 and SSE4.1. If this is not possible, you can set the `DNNFile` parameter to **`none`** to allow non-DNN speech-to-text without hardware limitations.

> **TIP:**
> In relative mode, IDOL Speech Server must ensure that it can meet a specified operational speed. It determines a minimum permitted target speed based on the size of the smallest DNN in the language pack, the frame duplication approximation value, and the speed/SIMD capability of the processor.
>
> If you attempt to use a target speed below the allowed minimum, IDOL Speech Server returns an error message, which states the minimum value.

Enable Word Confidence Scores

The speech-to-text engine can also generate word confidence scores. A word confidence score represents how acoustically similar the recognized word in the audio is to the acoustic model of the word in IDOL Speech Server. Word confidence scores do not indicate whether the word has been correctly recognized, because there are many words in any spoken language that sound very similar to each other. However, there is a general correlation between confidence scores and recognition rates.

Word confidence scoring is enabled by default. To turn off word confidence scores, set the `Conf` action parameter to **`False`** when you send the `AddTask` action to run the speech-to-text task.

Unless you are using a version 6.0+ acoustic model (that uses DNN), calculating confidence values adds an overhead to memory and CPU usage.

Tune Parameters

You can set additional parameters in the tasks configuration file to improve the performance of speech-to-text.

Noisy Data

If the audio data contains a lot of background noise or foreground music, you can enable speech detection to improve speech-to-text rates:

- In the `[frontend]` module used by the speech-to-text task, set the `DetectSpeech` parameter to **`True`** to modify how the speech-to-text engine processes audio sections that are labeled as speech, which can improve recognition in these sections.
- In the `[normalizer]` module used by the speech-to-text task, set `ZeroSilFrames` to **`True`**. The speech-to-text engine skips over sections of audio that are identified as silence.

Missing Words in Transcript

If many of the words in the audio do not appear in the transcript, the language model might be too strongly weighted. In the language pack section of the configuration file, experiment with the following parameters:

- Lower the weighting of the `LmScale` parameter (the recommended range is between `0.2` and `2.0`).
- Raise the weighting of the `LmOffset` parameter (the recommended range is between `-0.5` and `+0.5`).

Extra Words in Transcript

If the speech-to-text is producing many more words in the transcript file than are spoken in the audio, the language model might be too weakly weighted. In the language pack section of the tasks configuration file, experiment with the following parameters:

- Raise the `LmScale` parameter (the recommended range is between `0.2` and `2.0`).
- Lower the `LmOffset` parameter (the recommended range is between `-0.5` and `+0.5`.

You can also tune the following speech-to-text parameters to improve general speech-to-text performance :

- `Mode`
- `ModeValue`

For more information about these parameters, see the *IDOL Speech Server Reference*.

Use a Custom Language Model

Using a custom language model can substantially improve speech-to-text rates, if the model represents the speech data very accurately.

For instructions on how to build custom language models, see Create Custom Language Models, on page 129. After you build a custom language model, you must add it to the language pack section in the tasks configuration file (see Configure Language Packs, on page 62).

To use a custom language model for speech-to-text

- Include the `CustomLm` parameter when you send the `AddTask` action. Set it to the name of the custom language model and the interpolation weight, separated by a colon (:).

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&File=C:/myData/Speech.wav&Out=SpeechTranscript.ctm&Lang=ENUS&CustomLm=myLangModel:0.4
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `SpeechToText` task on the `Speech.wav` file using both the U.S. English and `myLangModel` language packs, and to write the results to the `SpeechTranscript.ctm` file. The `myLangModel` language pack is weighted at `0.4` against the U.S. English pack.

IDOL Speech Server suggests the interpolation weight to use at the time that you build the custom language model. All custom language models are placed in a specific folder, so you need to specify only the base name of the custom language model.

Troubleshooting

IDOL Speech Server returns error messages if it encounters problems responding to actions.

- For errors associated with actions, for example incorrect parameters, see the individual parameters in the *IDOL Speech Server Reference* for the correct usage.
- For language pack loading errors, see Load a Language Pack Manually , on page 240 and Unload a Language Pack, on page 242.
- For stream timeout errors, see Stream Live Audio, on page 203.

Generate Diagnostics

The speech-to-text tasks can also generate diagnostic information. Generate diagnostic information before you contact customer support, to help them to solve issues more quickly.

To enable speech-to-text diagnostics

- Include the `Diag` and `DiagFile` parameters when you send the `AddTask` action. Set `Diag` to **`True`** and `DiagFile` to the name of the file to write the diagnostic information to.

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&File=C:/myDataSpeech.wav&Out=SpeechTranscript.ctm&Lang=ENUS&Diag=True&DiagFile=Diagnostics.txt
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `SpeechToText` task on the `Speech.wav` file, and to write the results to the `SpeechTranscript.ctm` file and the diagnostics information to `Diagnostics.txt`.

Score Speech Recognition

This section describes how to score speech-to-text results against a truth transcript.

If a truth transcript corresponding to an audio recording is available, you can use the `Scorer` standard task to calculate how accurate the speech-to-text output is. To get an accurate estimate, the transcript must be verbatim–that is, every word must be transcribed, regardless of whether it is grammatically correct. The key metrics that the scorer reports are general word precision, recall, and the F-measure.

Prepare the Truth Transcript Text

To prepare the truth transcript, you must use the `TextNorm` task to normalize the text. Normalization ensures that the speech-to-text transcript and the truth transcript text are comparable, avoiding situations such as the speech-to-text transcript containing “three” while the truth transcript contains “3”.

For details of the normalization procedure, see Run Text Normalization, on page 142. For more information about transcription requirements, see Audio Transcript Requirements, on page 281.

Run the Scorer

The scorer first compares the speech-to-text transcript with the truth transcript to align the two. Then, it counts the number of words that are matched, as well as those that are mismatched in the alignment.

To run the scorer task

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`Scorer`**.
`TxtFile`	The normalized ‘truth’ transcript file.
`CtmFile`	The .ctm file that was produced by speech-to-text on the audio file.
`ScoreFile`	The file to write the scores to.
`Out`	The .ctm file to write the ‘truth’ transcript including timestamps to.

For example:

```
http://localhost:13000/action=AddTask&Type=Scorer&TxtFile=C:\data\transcript.txt&Ct
mFile=C:\misc\alignment.ctm&ScoreFile=score.ctm&Out=FinalTranscript.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `Scorer` task by comparing the `transcript.txt` file to the `alignment.ctm` file. The action instructs IDOL Speech Server to write the scores to the `score.ctm` file, and the truth transcript that contains the timestamps to the `FinalTranscript.ctm` file .

Use the `GetResults` action to return the results.

Format of Scorer Results

By default, the returned score file contains the following sections:

Matching details	Each word in the truth transcript is shown alongside the word from the speech-to-text transcript that it is matched to (truth transcript is on the left; speech-to-text transcript is on the right) and a score that shows the distance between the two words. A score of 0 (zero) indicates that the words match exactly.
Truth analysis	For each word in the truth transcript, this section shows whether the word was correctly recognized (1) or not (0).
Recognition analysis	For each word in the speech-to-text transcript, this section shows whether the word was correctly recognized (1) or not (0).
Analysis details	This section shows line-by-line matching of the truth transcript with the speech-to-text transcript.
Summary	Displays the recall, precision, and f-measure scores.

To return the Summary section only, you can either:

- Set the `ShowAlignment` and `DetailedScore` configuration parameters to **`False`** in the `align` module.
- Send the `ShowAlignment` and `ShowDetails` action parameters with the `AddTask` action; the action parameters override the settings in the configuration file.

For more information about these parameters, see the *IDOL Speech Server Reference*.

Common Problems Related to Scoring Outputs

The common problems that can reduce the performance of transcript alignment are:

- Errors or missing sections in the transcript
- Missing sections in the audio
- The text is not normalized
- Inconsistent word compounding or breaking can lead to perceived errors; for example, “taxrate” compared to “tax” and “rate” as two separate words

The scorer uses the `align` module, which it shares with the transcript alignment task.

Chapter 8: Create Custom Language Models

This section describes the case for building custom language models, and provides instructions for how to do so.

- Overview 129
- Calculate Perplexity 130
- Look Up Vocabulary 132
- Look Up Pronunciations 133
- Training Text for Custom Language Models 135
- Build the Language Model 136
- Evaluate the Custom Language Model 138
- Troubleshooting 138

Overview

IDOL Speech Server requires language packs to perform speech processing tasks. Several language packs are available (see Supported Resources, on page 47). A language pack contains a language model and an acoustic model. The key components of the language model are:

- The word vocabulary and a pronunciation dictionary that contains these words
- The word N-gram probabilities

The language model covers a broad vocabulary, reflecting the general spoken language. However, you might want to process speech data that covers specialized topics, such as financial or medical topics. The standard language model might not cover such specialized vocabulary or sentence structures (relating to N-gram patterns). In such cases, you can build custom language models with specialized vocabulary for IDOL Speech Server to use when processing this audio.

When you build a language model, you can control:

- The vocabulary size
- The vocabulary contents, by providing a 'must include' list and an 'exclusion' list
- The overall size of the language model

You must also decide whether to treat the training text as a 'closed' set or an 'open' set. Most language models are built with the assumption that the training text is part of an 'open' set, meaning that it does not represent the entire set of sentences expected from the language. A closed set contains all the sentences that occur in the data to be processed. An example of a closed set of text is a transcript language model (see Transcript Language Models, on page 33).

Building a new language model requires a lot of text–in the order of millions or billions of words. The standard language packs are usually built with many billions of words of text. Therefore, the best way to customize a language model is to build a small custom language model that uses the specialized text, and then combine it with the standard language model when you perform speech-to-text.

You can use an IDOL Content component as a source of text data when you build a language model, either in addition to, or instead of, local text data. The data in the IDOL Content component index must be appropriate for building the language model. You can normalize the text after retrieval before using it to build the language model.

For more details about the IDOL Content component configuration, refer to the *IDOL Server Administration Guide*.

Standard and Custom Language Models

Before you run a speech processing operation, you must decide whether to use a standard language model or a custom model. This table summarizes the main features of both types of language model.

Standard language model	Custom language model
Available out of the box	Need to create yourself using IDOL Speech Server tools
Even coverage of topics	Focuses on particular topics, which increases accuracy
Trained on billions of words	Generally trained on a smaller number of words; however, you can combine it with the standard language model when you perform tasks, to increase the coverage
Might not cover specialist vocabulary	Can cover specialist vocabulary, such as technical terms or product names

You can also estimate the suitability of a language model for a task by calculating its perplexity (see Calculate Perplexity, below).

Calculate Perplexity

Perplexity is a metric used in the language modeling. It indicates the average branching factor for a typical language.

To measure perplexity, you need:

- Sample text that closely resembles the speech transcripts that you intend to process with the language model, or the speech transcripts themselves
- The language model to measure

To measure perplexity of a language model

1. Normalize the sample text (see Run Text Normalization, on page 142).
2. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

   `Type` The task name. Set to **`LmPerplexity`**.

`Tlm`	The base language model to analyze.
`Text`	The sample text file.
`Out`	The log file to generate.

For example:

```
http://localhost:13000/action=AddTask&Type=LmPerplexity&Text=C:\data\transcript.txt
&Tlm=C:\LP\ENUK\ver-ENUK-5.0.tlm&Out=PerpScore.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to calculate the perplexity of the `ver-ENUK-5.0.tlm` language model using the `transcript.txt` sample text, and to write the results to the `PerpScore.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results. See Find Recommended Language Resources to Use in a Task, on page 103.

Format of Perplexity Results

The perplexity task log file contains the following information:

Perplexity score	The average branching factor of the language in the text.
Total word count	The number of words in the sample text file.
OOV (out-of-vocabulary) rate	The number of words in the sample text that are not included in the language model vocabulary, as a percentage of the total word count.
Unique OOV rate	The number of unique OOV words as a percentage of the total number of unique words in the sample text.

The log file also lists all the OOV words, sorted by number of occurrences in the text and then alphabetically. Each word is listed alongside its number of occurrences.

For example:

```
Perplexity is 142.567 over 492 words, ignoring 24 OOV words.
Total word count is: 516  Count without <s> is: 492
OOV rate: 24 OOV / 492 words = 4.878%
Unique OOV: 20 OOV / 246 words = 8.130%

OOV WORDS: (20 unique words, 24 instances in text)
3 But
2 That
2 Well
1 Airbase
1 All
1 And
1 A
1 Beginning
1 Dramatic
```

```
1 He's
1 Interestingly
1 Of
1 She's
1 So
1 Tell
1 There
1 They
1 This
1 What
```

Perplexity values around or below 100 are typical and acceptable for call center-like conversations. Aim for this value when you process telephone data (8 kHz sampling rates).

Perplexity values around or below 250 are typical and acceptable for broad coverage content, such as news. Aim for this value when you process such audio data (16 kHz sampling rates).

Look Up Vocabulary

IDOL Speech Server can list the 30,000 most frequent words in a specific language model.

To obtain the vocabulary listing

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`LmListVocab`**.

  `Tlm` The language model to query. The language model file is a .tlm file in the relevant language pack.

  `Out` The file to write the results to.

For example:

```
http://localhost:13000/action=AddTask&Type=LmListVocab&Tlm=C:\LP\ENUK\ver-ENUK-5.0.tlm&Out=ENUKvocab.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to write the 30,000 most frequently occurring words in the `ver-ENUK-5.0.tlm` language model to the `ENUKvocab.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results. See Get Task Results, on page 105.

The results specify the 30,000 most frequently occurring words, starting from the most frequent word.

In addition to listing the vocabulary, you can search the language model for a specific word and, if present, where it is ranked in terms of frequency.

To search for a word

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`LmLookup`**.

  `Tlm` The language model to query. The language model file is a .tlm file in the relevant language pack.

  `Word` The word to look up.

  `Out` The file to write the results to.

For example:

```
http://localhost:13000/action=AddTask&Type=LmLookup&Tlm=C:\LP\ENUK\ver-ENUK-5.0.tlm&Word=garden&Out=SearchResult.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to search the `ver-ENUK-5.0.tlm` language model file for the word '`garden`' and write the results to the `SearchResult.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results. See Get Task Results, on page 105.

IDOL Speech Server displays the results in the format:

`Word [`*`word`*`], rank: `*`xxx`*

where *word* is the search term and *xxx* is the frequency ranking in the language model (`1` is the most frequent word).

Look Up Pronunciations

If you are unsure as to whether a particular word exists in a language pack with the right pronunciation, IDOL Speech Server can list the pronunciations for a specific word in the pronunciation dictionary file.

To obtain the pronunciation listing

- Send the `GetWordPron` action to IDOL Speech Server, and set the following parameters:

  `Name` The name of the language pack.

  `Token` The word for which to retrieve pronunciations.

For example:

```
http://localhost:13000/action=GetWordPron&Name=ENUK&Token=pronunciation
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to return the pronunciations listed in the pronunciation dictionary file for the word `pronunciation`.

This action returns a response:

```
<responsedata>
   <word>pronunciation</word>
      <pronunciations>
         <pron>p r ax n ah n s iy ey sh ax n</pron>
         <pron>p r ow n ah n s iy ey sh ax n</pron>
      </pronunciations>
</responsedata>
```

where `<word>` is the word that you submitted as the value of the `Token` parameter, and `<pron>` is the pronunciation listed for that word in the dictionary.

You can replace or add alternative pronunciations of words when you load a language pack; see Load a Language Pack Manually , on page 240 for more information.

You can also return a list of all the phonemes in a language pack, with the top five words and their corresponding full pronunciations for each phoneme.

To obtain a phoneme listing

- Send the `GetPhoneList` action to IDOL Speech Server, and set the following parameter:

  `Name` The name of the language pack.

For example:

```
http://localhost:13000/action=GetPhoneList&Name=ENUK
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to return a list of all phonemes in the `ENUK` language pack.

This action returns a response:

```
<phone>ax</phone>
   <phoneExamples>
      <word>that - dh ax d</word>
      <word>are - ax r</word>
      <word>this - dh ax s</word>
      <word>then - dh ax n</word>
      <word>had - hh ax d</word>
   </phoneExamples>
<phone>ih</phone>
   <phoneExamples>
      <word>in - ih n</word>
      <word>is - ih s</word>
      <word>to - t ih</word>
      <word>with - w ih th</word>
      <word>it - ih d</word>
   </phoneExamples>
etc.
```

where `<phone>` is the phoneme, and `<word>` is the example word and pronunciation.

Training Text for Custom Language Models

To produce an effective custom language model, you must build it using text that resembles the data that you want to process. For example, if you intend to apply the speech-to-text task to news monitoring, you would train the language model using recent news articles gathered from a wide range of sources.

The standard IDOL Speech Server English language model is constructed using text that contains many billions of words and covering a wide range of topics. Such wide coverage significantly reduces the amount of text required to build the custom language model. In the deployment, the standard language model is used together with the custom language model interpolated with an appropriate weight.

The amount of text required to build a custom language model can vary from a few thousand words to several hundred thousand words, depending on the topic. Generally, the more text that is used to build the custom language model, the more accurate the model is. However, the gains in accuracy start tapering off beyond a certain number of words. The number of words depends on the size of the topic; for a typical topic (for example, technical support), the tapering might begin around 100,000 words.

Select Appropriate Text

The type of training text that you can use in building a custom language model depends on the data that you want to process. Examples of relevant text include:

- Slides used in delivering a lecture
- Other articles written by the same author who delivered the lecture
- A web article that discusses the particular topic
- Literature that describes the product or company
- Company websites
- Any other document related to the topic or event

The training text does not need to completely cover all the vocabulary that you expect in the data that you want to process. The custom language model combines with the standard language model, so that the language coverage is the sum of the two models. Therefore, building a custom language model using even small quantities of training text still provides benefits.

Prepare Text

It is important to ensure that the text used for custom language model building is cleaned up to a reasonable level.

- Remove any material that does not occur in standard written English, such as HTML tags, and anything that would not usually occur in spoken language, such as tables.
- Ensure that sentence breaks (periods) are present in the text.
- Ensure that there are no duplicated sections in the text.
- Ensure that the text encoding is UTF-8.

> **NOTE:**
> Japanese, Korean, Mandarin, and Taiwanese Mandarin languages require text segmentation before IDOL Speech Server can process them. Text segmentation inserts whitespace between words. The `LanguageModelBuild` task segments text if you set the `DoSegment` parameter to **`True`** (see Build the Language Model, below).

Normalize Text

You must normalize the text used for language model building before processing so that word representations are standardized. For example, '1' and 'one' are treated as two different representations of the same digit. For more information about the normalization scheme used in IDOL Speech Server, see Audio Transcript Requirements, on page 281.

Many of the IDOL Speech Server text operations require you to normalize input text as an initial step. For details of the normalization procedure, see Run Text Normalization, on page 142.

Build the Language Model

After you have selected and prepared the training text files, you can build the custom language model.

To build the language model

1. Create a list that contains the file names (including file extensions) of all training text files. You do not have to include the file paths because you can use the `DataPath` parameter to specify the directory path in the next step.

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
2. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`LanguageModelBuild`**.
`ContentDatabase`	The IDOL Content component database to retrieve text data from. This parameter has an effect only if you set `ContentHost`. If you set `ContentHost` but do not set this parameter, IDOL Speech Server retrieves text from all databases.
`ContentHost`	The host name or IP address of the IDOL Content component that you want to retrieve training text data from.
`ContentPort`	The ACI port of the IDOL Content component that you want to retrieve training text data from. This parameter has an effect only if you set `ContentHost`. By default, IDOL Speech Server uses port 9100 to contact the IDOL Content component.
`ContentTextTag`	The IDOL document fields that contain the text that you want to use to train the language model. Separate multiple field names with spaces, commas, or plus symbols (+). By default, IDOL Speech Server uses the content of the `DRECONTENT` document field as training text.

DataList	The list that specifies the training text files.
DataPath	The path to the directory that contains the files specified in the DataList parameter.
KeepList	The path to a file that contains a list of words that the language model must contain. For more information on the format of the file, see the *IDOL Speech Server Reference*.
Lang	The language pack to use as a base (for example, ENUK-tel).
NewLanguageModel	The name to give the custom language model that is generated. You must include the file extension (.tlm) in the parameter.
NewLmInfoFile	The output Language Model Information file name. If you set this parameter, you must include the file extension (.lmi) in the parameter. If you do not set this parameter, the file has the same as the generated language model (and is located in the same directory), but with the extension .lmi instead of .tlm. **NOTE:** You can use the GetResults action to retrieve the .lmi file by setting the Label parameter to lmi.
NewDictionary	The name of the dictionary to generate; usually it is the same value as NewLanguageModel. If you set this parameter, you must include the file extension (.dct.sz) in the parameter. If you do not set the NewDictionary parameter, Speech Server uses the output language model file name specified as the value of the NewLanguageModel parameter, but with the extension .dct.sz rather than .tlm.
DoSmoothing	If you are using a custom language model for a transcript alignment task, set DoSmoothing to **False**. Otherwise, you can use the default value of **True**.

If the training text files contain Japanese, Korean, Mandarin, or Taiwanese Mandarin languages, set the DoSegment parameter.

DoSegment	Set to **True** to enable text segmentation.

For example:

```
http://localhost:13000/action=AddTask&Type=LanguageModelBuild&DataList=ListManager/
Langmodel&DataPath=C:\LanguageModelFiles&Lang=ENUK-tel&NewLanguageModel=mymodel.tlm
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to use the training text specified in the Langmodel list and the ENUK-tel language pack to build a new language model and dictionary file, both named mymodel. This action also calculates a recommended interpolation weight at the end of the language model building process.

NOTE:
The interpolation weight is only a suggested weight–you can choose to set other weights.

The new language models are placed in the custom language models folder that is specified by the `CustomLmDir` parameter in the IDOL Speech Server configuration file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Send the `GetResults` action to retrieve the recommended interpolation weight for the custom language model. See Get Task Results, on page 105.

Evaluate the Custom Language Model

You can check whether the custom language model improves the accuracy of the speech-to-text over the standard language model. The best method is to compare the speech-to-text results when you use the custom language model with the results when you use the standard language model alone. You can use such a comparison iteratively to better select the training text, as well as the custom language model. For more information, see Score Speech Recognition, on page 127.

Troubleshooting

If you receive errors associated with actions (for example, incorrect parameters), see the individual parameters in the *IDOL Speech Server Reference* for the correct usage.

Chapter 9: Normalize Text

This section describes how to normalize text, which is a preprocessing requirement for several speech processing operations.

- Overview 139
- Supported Languages 139
- Segment Text 140
- Metadata Tag Syntax 140
- Run Text Normalization 142
- Troubleshooting 143

Overview

All text materials must be normalized before any processing by IDOL Speech Server. The text normalizer prepares the data in a consistent form so that words with the same meaning (for example, “two” and “2”) are treated as being the same.

The text normalizer focuses on the following parts of a set of text:

- Numbers, ordinals, and cardinals
- Dates and time
- Acronyms and alphanumerics
- Punctuation marks, brackets, quotation marks, and so on (filtering these out as appropriate)

You can also format the text as metadata to mark text that you want IDOL Speech Server to exclude from the normalizing process (for example, notes about the transcript). Metadata sections pass through the text normalizer unmodified, and IDOL Speech Server flags them with metadata tags in the normalized transcript. Metadata tags do not affect further processing, such as transcript alignment. For further information on how to use metadata tags, see Metadata Tag Syntax, on the next page.

Supported Languages

IDOL Speech Server can normalize text in the following languages.

Australian English (ENAU)	Hungarian (HUHU)
Brazilian Portuguese (PTBR)	Italian (ITIT)
British English (ENUK)	Japanese (JAJP)
Canadian English (ENCA)	Korean (KOKR)
Canadian French (FRCA)	Latin American Spanish (ESLA)
Castilian Spanish (ESES)	Mandarin (ZHCN)

Dutch (NLNL)	Singaporean English (ENSG)
French (FRFR)	Turkish (TRTR)
German (DEDE)	U.S English (ENUS)
Hindi (HIIN)	

For Japanese, Korean, Mandarin, and Taiwanese Mandarin languages, you must use the SegmentText task to segment the text before you submit it for normalization (see Segment Text, below).

For transcripts in languages for which normalization is not supported, you must manually normalize the text (see Manually Normalize Text, on page 281).

Segment Text

Japanese, Korean, Mandarin, and Taiwanese Mandarin languages do not separate words with whitespace. You must segment text in these languages into words before IDOL Speech Server can process them.

To segment text

- Send an AddTask action to IDOL Speech Server, and set the following parameters:

Type	The task name. Set to **SegmentText**.
Lang	The language pack to use.
TxtFileIn	The text file to segment.
TxtFileOut	The text file to write the segmented text to.
Pgf	The pronunciation information file to use.

To exempt a section of text from segmentation, move the section to a new line and add hash symbols (#) at the beginning and end of the section. You must also set the IgnoreHashLines parameter:

IgnoreHashLines	Set to **True** to exempt sections bounded by hash symbols from segmentation.

For example:

```
http://localhost:13000/action=AddTask&Type=SegmentText&Lang=JAJP&TxtFileIn=C:/Data/
Japanese.txt&TxtFileOut=JA_seg.txt&PgfFile=T:\LP\ENUK\ver-ENUK-5.0.pgf
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to segment text in the Japanese.txt file and write the results to the JA_seg.txt file in the Temp directory.

Metadata Tag Syntax

IDOL Speech Server allows you to add extra information into a transcript, which it then marks with a metadata tag in the processed transcript, enabling you to easily find it. The extra information does not

affect transcription alignment. For example, you might want to mark the point in a lecture transcript where a video was played. You could add the following to the transcript:

```
...
Let’s
look
at
this
example
<metadata video = GlobalIssues />
Environmental
issues
...
```

The aligned transcript that is generated contains the following:

```
...

 1   A   10.1    0.3    Let’s                        1.0

 1   A   10.4    0.2    look                         1.0

 1   A   10.6    0.3    at                           1.0

 1   A   10.9    0.3    this                         1.0

 1   A   11.2    0.3    example                      1.0

 1   A   11.5    0.0    video = GlobalIssues         1.0    <-- metadata tag

 1   A   11.5    0.4    Environmental                1.0

 1   A   11.9    0.3    issues                       1.0
```

For more information about the aligned transcript format, see Align the Transcript, on page 147.

Metadata tags must conform to the following syntax to pass through the text normalizer unmodified. The syntax is loosely based on the format of tags used in XML.

```
Metadata ::= '<metadata' (S Attribute)* S? '/'? '>'

S ::= (#x20 | #x9 | #xD | #xA)+

Attribute ::= Name Eq AttValue

Name ::= NameStartChar (NameChar)*

NameStartChar ::=   ":"
                  | [A-Z]
                  | "_"
                  | [a-z]
                  | [#xC0-#xD6]
                  | [#xD8-#xF6]
                  | [#xF8-#x2FF]
```

```
                    | [#x370-#x37D]
                    | [#x37F-#x1FFF]
                    | [#x200C-#x200D]
                    | [#x2070-#x218F]
                    | [#x2C00-#x2FEF]
                    | [#x3001-#xD7FF]
                    | [#xF900-#xFDCF]
                    | [#xFDF0-#xFFFD]
                    | [#x10000-#xEFFFF]
NameChar ::=   NameStartChar
              | "-"
              | "."
              | [0-9]
              | #xB7
              | [#x0300-#x036F]
              | [#x203F-#x2040]
Eq ::= S? '=' S?
AttValue ::=   '"' [^"]* '"'
              | "'" [^']* "'"
```

Run Text Normalization

To normalize text, complete the following steps.

To run the text normalizer

- Send an `AddTask` action to IDOL Speech Server, and set the following action parameters:

`Type`	The task name. Specify **`TextNorm`**.
`File`	The text file to normalize.
`Lang`	The language pack to use.
`Out`	The file to write the normalized text to.

For example:

```
http://localhost:13000/action=AddTask&Type=TextNorm&File=C:/myData/Speech.txt&Out=SpeechNorm.ctm&Lang=ENUS
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `TextNorm` task on the `Speech.txt` file and write the results to the `SpeechNorm.ctm` file. The `Speech.txt` file contains U.S. English dialect speech.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the normalized text. See Find Recommended Language Resources to Use in a Task, on page 103.

Troubleshooting

If you receive errors associated with actions (for example, incorrect parameters), see the individual parameters in the *IDOL Speech Server Reference* for the correct usage.

Chapter 9: Align Transcripts

This section describes how to align transcripts with the corresponding audio, and how to add time locations for each word in the transcript.

- Overview 143
- Common Problems 144
- Normalize the Transcript 145
- Build the Transcript Language Model 145
- Run Speech-to-Text 145
- Check Transcript 145
- Align the Transcript 147
- Two Pass Alignment 148
- Troubleshooting 148

Overview

If a transcript is available for an audio recording, you can use the `TranscriptAlign` task to place the time location for each word in the transcript. Use this task to align subtitles to audio or video files.

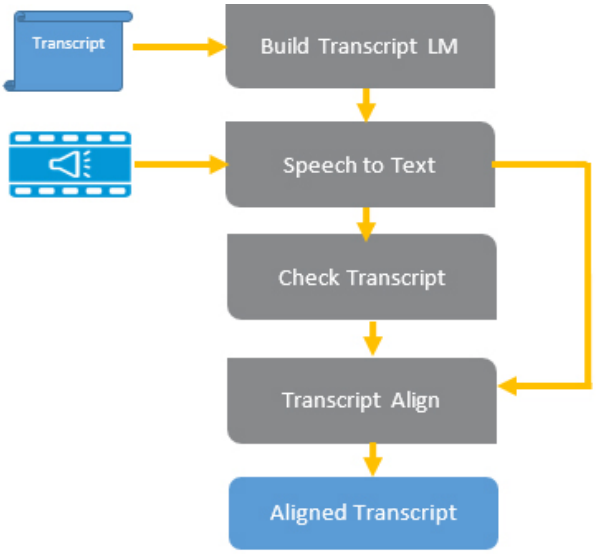
The transcript does not need to match the speech data exactly. The transcript aligner can tolerate small numbers of errors in the transcript, as well as mitigating factors related to audio, such as background noise and music.

The transcript aligner can also place metadata tags in the transcript to allow you to easily identify sections. These metadata tags do not affect the alignment process. For more information, see Metadata Tag Syntax, on page 140.

The alignment process works by using speech-to-text to identify words, sounds, or characters from the transcript within the audio, and assigning them a time location.

The accuracy of the speech-to-text process affects the accuracy of the end alignment. For best results, you should run speech-to-text using a custom language model built from the transcript text. The custom language model models the words in the transcript text and makes them much more likely to come out in the speech-to-text transcript.

The following diagram describes the transcript alignment workflow.



The transcript alignment process includes the following steps:

1. Normalize the transcript so that you can identify numbers written in numeric form, and so on. See Normalize the Transcript, on the next page.
2. Build a transcript language model with the normalized text. See Build the Transcript Language Model, on the next page.
3. Run the speech-to-text task using the custom language model from Step 2. See Run Speech-to-Text, on the next page.
4. If your audio and transcript is likely to contain misaligned sections, or if you have very large files and speed is important, run a check transcript task using the output from the speech-to-text task. This task provides rough time estimates and information on how well the audio matches the transcript. These time estimates make the alignment process faster and more accurate. See Check Transcript, on the next page.
5. Run the transcript alignment task using the information from the check transcript task in Step 4.

Common Problems

Common problems that can reduce the performance of transcript alignment are:

- Errors or missing sections in the transcript
- Missing audio sections
- Background noise and music

The transcript aligner can cope well if the erroneous sections are not too large. However, if there are too many errors, alignment fails. In this situation, HPE recommends that you use the `TranscriptCheck` task (see Check Transcript, below).

Normalize the Transcript

To normalize the original transcript text, use the `TextNorm` task. The normalized file must contain only one word on each line. Before you send the `TextNorm` action, set the `WordPerLine` parameter to **`True`** in the `[TextNorm]` section of the tasks configuration file.

> **NOTE:**
> Normalization might split single entities into multiple words. For example, 35 can become thirty five.

For more information about text normalization, see Normalize Text, on page 139.

Build the Transcript Language Model

Use the `BuildLanguageModel` task to create a language model using just the normalized transcript text .

For more information on building a custom language model, see Create Custom Language Models, on page 129.

Run the speech-to-text task using the transcript language model with a suitable interpolation weight. The suggested range of weighting is between 0.5 and 0.9. Use the higher value if the transcripts are almost exact.

Run Speech-to-Text

Run speech-to-text to generate a recognition transcript. IDOL Speech Server then compares the original transcript with the speech-to-text transcript.

For more information about how to perform speech-to-text, see Speech-to-Text, on page 119.

Check Transcript

Now that you have a normalized transcript and a recognition transcript, you can check the transcript for any large, mismatched sections, and also generate some location pointers for the subsequent alignment task.

To check a transcript

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`TranscriptCheck`**.
`TxtFile`	The normalized transcript file.
`CtmFile`	The speech-to-text transcript file produced from the audio file.
`Out`	The file to write a transcript that contains the approximate timestamps to.
`DiagFile`	The alignment diagnostics file to generate.

For example:

```
http://localhost:13000/action=AddTask&Type=TranscriptCheck&TxtFile=C:\data\transcri
pt.txt&CtmFile=C:\misc\speechtext.ctm&Out=AlignResults.ctm&DiagFile=myResults.diag
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to compare the `transcript.txt` and `speechtext.ctm` files to produce the transcript file with approximate timestamps (`AlignResults.ctm`), and the diagnostics file (`myresults.diag`). The diagnostics file contains information on how well the speech-to-text transcript file and the normalized transcript file match.

The `AlignResults.ctm` output file is in the following format:

```
She 0.000 255.190
was 0.000 255.190
the 0.000 255.190
guardian 0.000 255.190
of 0.000 255.190
```

The file contains approximate timings of the transcript words in the audio file. You can use this information as the first step to aid the subsequent alignment task.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results files. See Get Task Results, on page 105.

Format of Diagnostic File

The diagnostic file has a five column format. For example:

```
0               304             *               *               0.000
304             618             *               *               0.000
618             933             *               *               0.000
933             1245            1.130           136.830         0.553
1245            1568            1.130           136.830         0.667
1568            1891            136.830         273.430         0.607
```

- The first two columns display word numbers in the original transcript. The first column displays the number of the first word in an analyzed segment, and the second column displays the final word in the segment.
- The third and fourth columns display timestamps from the speech-to-text transcript. These show the time locations for the first and final words in each segment.
- The last column shows the degree of match between the original and speech-to-text transcript segments (between 0 and 1, indicating no match and total match respectively).

If a column contains an asterisk (*), the text was not found in either the original transcript or the speech-to-text transcript. Many rows with low values in the fifth column indicates a problem.

Align the Transcript

The transcript aligner compares the speech-to-text transcript and the original transcript text to produce an aligned transcript. The aligner either uses words as whole units or breaks them down into phonemes or letters. You can therefore select one of three modes:

- `words`
- `prons` (phonemes)
- `letters`

In addition, the alignment algorithm can also work in one of two polarity modes:

- In positive polarity mode, only the matched units get a positive score, and mismatches are not penalized. This mode is the default mode for transcript alignment, and is more robust to transcript and speech data errors.
- In negative polarity mode, the matched units get a zero score, and the mismatched units are penalized. HPE recommends this mode if the transcripts are very accurate or the speech-to-text output is of a poor quality.

To run the transcript alignment task

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`TranscriptAlign`**.
`TxtFile`	The normalized transcript file.
`CtmFile`	The speech-to-text transcript produced for the audio file.
`Out`	The file to write the aligned transcript to.
`MatchType`	The mode–either `words`, `prons`, or `letters`.

For example:

```
http://localhost:13000/action=AddTask&Type=TranscriptAlign&TxtFile=C:\data\transcript.txt&CtmFile=C:\misc\speechtext.ctm&Out=AlignedTranscript.ctm&MatchType=words
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to compare the original transcript `transcript.txt` with the speech-to-text transcript `speechtext.ctm` to produce an aligned transcript, `AlignedTranscript.ctm`. The action instructs IDOL Speech Server to use the `words` alignment mode.

The output file is in the following format:

```
1 A 0.000 0.420 behind 1.000
1 A 0.420 7.790 it 1.000
1 A 8.210 2.870 all 1.000
1 A 11.080 0.000 <s> 1.000
1 A 11.080 0.000 Teaism 1.000
1 A 11.080 0.000 was 1.000
1 A 11.080 0.000 Taoism 1.000
1 A 11.080 0.000 in 1.000
1 A 11.080 0.000 disguise 1.000
1 A 11.080 0.000 <s> 1.000
```

From left to right, the columns in the output data file contain:

- The channel ID (usually 1)
- A fixed field, A
- The start time of the recognized word in seconds
- The duration of the recognized word in seconds
- The recognized word
- The confidence value for the recognized word

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results. See Find Recommended Language Resources to Use in a Task, on page 103. For an explanation of the results file, see Interpret the Results, on page 121.

Two Pass Alignment

HPE recommends two-pass alignment if the alignment quality is poor or if large sections of audio have not been aligned. This situation can arise when aligning very long audio. In two-pass alignments, alignment occurs over two steps:

1. Use `TranscriptCheck` to produce approximate timings for the transcript.
2. Perform alignment at the `prons` level by using the `TranscriptCheck` approximate transcript time output file.

Troubleshooting

If you receive errors associated with actions (for example, incorrect parameters), see the individual parameters in the *IDOL Speech Server Reference* for the correct usage.

Chapter 10: Phonetic Phrase Search

This section describes how to perform phonetic phrase search.

- Overview 149
- Create the Phoneme Time Track File 149
- Search the Phoneme Time Track File 150
- Phonetic Phrase Search in a Single Step 151

Overview

Phonetic phrase search is a fast and approximate way of searching for words and phrases. In essence, IDOL Speech Server attempts to look for sections of audio that sound similar to the word or phrase being searched for. Phonetic search is broken into two separate stages:

1. Analysis of the audio file to create phoneme time tracks. You can combine time tracks from multiple audio files in a single file. This stage is independent of the word or phrase that you want to search for, and is known as the *ingestion stage*.
2. When queried for a word or phrase, the phonetic matching engine searches the time track file to identify matching entries, and generates a score for each match. This stage is known as the *search stage*. It is much faster than the ingestion stage, so you can perform very fast searches after the time track information has been generated.

You can carry out the search in two separate steps (creating the time track file and then searching it) or in a single combined step.

Create the Phoneme Time Track File

IDOL Speech Server stores phoneme time track information in an .fmd file. You process an individual audio file to create a phoneme time track file. You can then search this file, or combine time track files for multiple audio files into a single file, which you can then search.

To create a phoneme time track file for an audio file

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`CreateFMD`**.

  `File` The audio file to process.
  To restrict processing to a section of the audio file, set the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

  `Lang` The phonetic phrase match language pack to use. For more information about available language packs, see Supported Resources, on page 47.

  `Out` The name of the phoneme time track file to produce.

For example:

```
http://localhost:13000/action=AddTask&Type=CreateFMD&File=C:/myData/Speech.wav&Lang
=ENUS-pm&Out=Speech1.fmd
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to process the `Speech.wav` file to produce the `Speech1.fmd` time track file, using the `ENUS-pm` language pack.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the phoneme time track file. See Get Task Results, on page 105.

To combine multiple phoneme time track files

1. Create a list that contains the names of the individual time track files to combine. Each entry in the list must be on a separate line, and must consist of label and file name pairs separated by a semicolon. The label corresponds to the name to give to the file, and the file names correspond to the full path of the files to be combined. For example:

   ```
   Label1;filename1
   Label2;filename2
   ...
   ```

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
2. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`CombineFMD`**.
`ListFile`	The list that specifies the individual time track files to combine.
`ListPath`	The path to the directory that contains the phoneme time track files.
`FileOut`	The name of the combined phoneme time track file to produce.

For example:

```
http://localhost:13000/action=AddTask&Type=CombineFMD&ListFile=ListManager/fmdList&
ListPath=C:\PHRASESEARCH\fmd&FileOut=myData.fmd
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to combine the individual phoneme time track files specified in the `fmdList` list to produce the single file `myData.fmd`.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the phoneme time track file. See Get Task Results, on page 105.

Search the Phoneme Time Track File

Use the following procedure to search the phoneme time track file.

To search the phoneme time track file

1. Create a list that contains the phrases to search for.

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
2. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`SearchFMD`**.
`File`	The phoneme time track file to search.
`PhraseList`	The list that specifies the phrases to search for.
`Lang`	The phonetic phrase search language pack to use.
`Out`	The file to write the results to.

For example:

```
http://localhost:13000/action=AddTask&Type=SearchFMD&File=C:\Data\myData.fmd&PhraseList=ListManager/phrases&Lang=ENUS-pm&Out=SpeechResults.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to search the `myData.fmd` file for the phrases in the `phrases` list and write the search results to the `SpeechResults.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Find Recommended Language Resources to Use in a Task, on page 103.

Phonetic Phrase Search in a Single Step

Use the following procedure to perform phonetic phrase search in a single operation.

> **NOTE:**
> HPE recommends that you use this task for single searches only, because the procedure does not produce a phoneme time track file. If you use the `PhraseSearch` task to perform multiple searches on a file, IDOL Speech Server must process the audio file each time. To perform multiple searches, HPE recommends that you process the audio file to produce a phoneme time track file, and then perform searches separately.

To perform phonetic phrase search on an audio file

1. To search for more than one phrase, create a list that contains the phrases.

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.

2. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

   `Type` The task name. Set to **`PhraseSearch`**.

   `File` The audio file to process.
   To restrict processing to a section of the audio file, set the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

   `Lang` The phonetic phrase search language pack to use.

   `Out` The file to write the results to.

   Set one of the following parameters.

   `Phrase` The word or phrase to search for.

   `PhraseList` The list that specifies multiple phrases to search for.

For example:

```
http://localhost:13000/action=AddTask&Type=PhraseSearch&File=C:/Data/Speech.wav&DNN
File=ver-ENUK-tel-6.2-8k.dnn&Lang=ENUK-pm&Out=SearchResults.ctm&Phrase=financial
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to search the `Speech.wav` file for the phrase '`financial`', , using the `ver-ENUK-tel-6.2-8k.dnn` DNN acoustic model file for processing, and to write the search results to the `SearchResults.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Get Task Results, on page 105.

Chapter 11: Use Speaker Clustering

You can use speech clustering to segment a speech waveform and separate it out into a number of speakers. IDOL Speech Server produces a timed sequence of labels that correspond to speaker assignments.

IDOL Speech Server provides the following speech clustering tasks:

- **ClusterSpeech**.This task carries out the basic clustering of wide-band speech into speaker segments. For example, if two speaker clusters are identified, the output labels are `Cluster_0` and `Cluster_1` respectively.
- **ClusterSpeechTel**. This task is essentially the same as the `ClusterSpeech` task, but is optimized for telephony audio. In particular, the audio classification is configured to suit speech, noise and music in telephone calls, and the final output can feature dial tones and DTMF-recognized characters.
- **ClusterSpeechToTextTel**. This task performs clustering of two speakers in a phone call, and uses the resulting speaker clusters to improve speech-to-text performance slightly by using speaker-sided acoustic normalization. As before, any telephony artifacts such as dial tones or DTMF tones are included, interspersed with the recognized words.

For more information on the speaker clustering tasks, see the *IDOL Speech Server Reference*.

The SplitSpeech Process

The key process in clustering speech is called `SplitSpeech`. This process requires:

- A stream of homogeneous acoustic segments separated by points of significant acoustic change.
- Audio categorization data to ensure that only regions classified as speech are considered.

The `SplitSpeech` process uses an agglomerative algorithm to find the best two segment clusters to merge. This process is then repeated until all potential merges fail a Bayesian Information Criterion threshold check. The final process should result in a smaller number of acoustically homogeneous speaker clusters.

Control the Number of Speakers

You can specify the minimum and maximum numbers of speakers to produce. For example, if you know that a telephone call consists of two speakers, you can use the `MinNumSpeakers` and `MaxNumSpeakers` parameters to set both the minimum and maximum number of speakers to **2** to guarantee that the process produces exactly this number of speakers.

Alternatively, you can algorithmically determine the number of speakers based on a sensitivity threshold. In this case, you can change the `MergeThresh` parameter from the default value of `0.0` to ensure that more or fewer speakers are produced.

For more information on these parameters, see the *IDOL Speech Server Reference*.

Processing Time

The recursive process of splitting speech can be very resource-intensive. If you are processing audio files longer than 10 minutes, and those files have consistent speakers, HPE recommends that you crystallize speaker information after approximately 10 minutes. For example, if your audio file is 30 minutes in length, and you crystallize the speaker information after five minutes, the process clusters the speakers in the first 5 minutes of the file. Thereafter, any subsequent speech segments are clustered into one of the speaker segments from the first five minutes of the file, rather than being assigned to new speakers. This crystallization means that classification of subsequent speech segments is faster.

You can also configure processing depending on whether accuracy or speed is more important. For instance, full matrices are more accurate, but slower; if processing speed is more important to you than accuracy, you could use diagonal covariance matrices (which are faster but less accurate) instead.

For more information on using the `DiagCov` and `FixTime` parameters, see the *IDOL Speech Server Reference*.

Chapter 12: Identify Speakers in Audio

This section describes how to perform speaker identification and speaker segmentation.

- Overview 155
- Create Speaker ID Feature Files 155
- Train Speaker Templates 156
- Estimate Speaker Score Thresholds 158
- Package Templates 162
- Identify Speakers in Audio 165
- Troubleshooting 167

Overview

> **NOTE:**
> IDOL Speech Server offers two sets of speaker identification tasks, based on different technologies. This section uses examples of the iVector-based tasks, but the original set of tasks are still available. The two sets of tasks have identical parameters and options unless otherwise specified. The previous tasks are maintained mainly for backwards compatibility, but in some cases might work better than new iVector tasks. The original tasks have same name as the iVector-based tasks, but without the `Iv` prefix.

The speaker identification operation identifies sections of audio that contain speech from known speakers. IDOL Speech Server lets you build a set of speaker templates to be identified based on audio samples that represent each speaker. This speaker set can be considered as either an *open-set*, or a *closed-set*:

- An **open-set** allows for the possibility of speakers in the audio that are unknown. Sections of speech that do not match any of the known speakers are identified as being from an unknown speaker. To achieve this, a score threshold is estimated for each speaker. A hit is considered genuine only if the score is over the threshold estimated for the specific speaker template. Having trained the speaker templates, the thresholds are calculated based on typical true and false hit scores for each template generated from a further set of audio samples.
- A **closed-set** assumes that all speakers in the audio being processed are from the closed speaker set. There are no score thresholds applied in this case.

Create Speaker ID Feature Files

Audio samples are used for training speaker templates, estimating speaker score thresholds, and for identification. There are tasks supporting these functions that work directly from an audio file (or stream). However, these tasks can only take a single audio source as input, due to the pipeline used for processing the audio.

For template training and for threshold estimation, you might want to use multiple audio files in a single task. To do this, create a speaker ID feature file for each audio file that you want to use, and then present the set of feature files to IDOL Speech Server.

To create a speaker ID feature file

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`IvSpkIdFeature`**.

  `File` The audio file that contains sample speech from one person.

  `Out` The name of the speaker ID feature file to create.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdFeature&File=C:/Data/BrownSpeech1
.wav&out=BrownSpeech1.ivp
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to create the `BrownSpeech1.ivp` feature file using the `BrownSpeech1.wav` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the speaker template feature file.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

Train Speaker Templates

- Create a Speaker Template from a Single Audio File 157
- Create a Speaker Template from Multiple Feature Files 157

You must create a speaker template for each speaker that you want to identify.

HPE recommends that you should use a minimum of five minutes of speech data for each speaker. There should be no speech from other speakers present in the training audio. In general, you should use good quality audio samples that contain only the speaker's voice and no significant background noise. However, it is important where possible to include examples of the speaker from a typical range of recording situations and environments that might be expected in the final system (indoors, outdoors, noisy, and so on). The spoken content can contain any vocabulary.

There are two ways to train speaker templates:

- **Single audio file**: IDOL Speech Server takes a single audio file that contains speech from a single speaker, and generates a single speaker template file. This approach is the most straightforward, but if the speech data you wish to use is stored in multiple audio files, you must use a third-party audio editing tool to combine them into a single file.
- **Multiple audio files**: This is a two-stage process, whereby you first create a set of speaker ID feature files from a set of audio files that contain speech from the single speaker, and then train a model on this set. With this approach, you can use more than one audio file to train a speaker

template, although this process does involve sending a number of action requests for generating each template.

Create a Speaker Template from a Single Audio File

This task takes a single audio file containing speech data from the speaker to be trained, and creates a new speaker template file.

To create a speaker template from a single audio file

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`IvSpkIdTrainAudio`**.

  `File` The audio file that contains sample speech from one person.

  `Out` The name of the speaker template file to create. You should include the iVector template file extension (`.iv`).

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdTrainAudio&file=C:/Data/BrownSpeech.wav&out=Brown.iv
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to create the `Brown.iv` template file by using the `BrownSpeech.wav` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the speaker template file.

To process streamed audio, you use the same task (`IvSpkIdTrainAudio`), with the `InputType` parameter set to **`Stream`**. For more information, see the *IDOL Speech Server Reference*.
You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

Create a Speaker Template from Multiple Feature Files

A two-step process allows you to perform speaker training with multiple audio files:

1. Create a set of Speaker ID feature files by using the `IvSpkIdFeature` task (see Create Speaker ID Feature Files, on page 155 for more information).
2. Use these feature files to create the speaker template file.

Assuming you have already created a set of feature files, you can train the new speaker template from this set by using the `IvSpkIdTrain` task.

First you must create a list file that contains a list of all the feature files. You can do this manually, or by using the List Manager.

Each element of the list should be the file name relative to the Speaker ID directory, and must include the file extension.

For example:

```
BrownSpeech1.ivp
BrownSpeech2.ivp
BrownSpeech3.ivp
BrownSpeech4.ivp
```

> **NOTE:**
> If the files are not stored in the Speaker ID directory, you can use the `DataPath` parameter to specify the location.

For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.

After you have created the list, send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdTrain`**.
`DataList`	A list file that lists the feature files to use.
`Out`	The name of the speaker template file to create.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdTrain&DataList=ListManager/BrownList&Out=Brown.iv
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to create the `Brown.iv` template using the feature files listed in the `BrownList` list file.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the speaker template file.

Estimate Speaker Score Thresholds

When identifying the most likely speaker in a section of audio, IDOL Speech Server scores how closely the segment acoustic properties match with each of the speaker templates. For *closed-set* operation, the top-scoring speaker is simply taken as being the true result.

However, for *open-set* identification, IDOL Speech Server needs to allow for unknown speakers in the audio. It does this by estimating a score threshold for each speaker; the hit is considered valid only if a template scores above this threshold. If for any audio segment the top-scoring template falls below the threshold associated with that speaker, the segment is assumed to be an unknown speaker.

The score threshold for each speaker template is based on an analysis of the speaker match scores observed for that template against both matching speaker data (true examples), and non-matching speaker data (false examples).

The IvSpkIdDevelAudio task takes an audio file or stream, along with the name of the speaker the file is associated with, and generates score statistics for one or more speaker templates. These statistics are then stored in an iVector template development file (.ivd).

> **NOTE:**
> If the speaker is not in the set, you can set the speaker name to “Unknown”.

You must run the IvSpkIdDevelAudio task once for each audio file to be used in threshold estimation. You can choose to append the scores for each audio file to a single .ivd file (the default method), or to create a separate development file for each audio file. You can use one or more development files when estimating the threshold for each speaker template.

To append the scores to a common development file, you must ensure that the file does not exist before you run the first IvSpkIdDevelAudio task.

The creation of an individual development file for each audio file ensures that the statistics do not get inadvertently appended to a file that already existed before running the first IvSpkIdDevelAudio task (for example, from a previous IDOL Speech Server installation). You must specify a unique name for the development file each time that you run the task, to avoid overwriting files.

You can specify the method to use by using the DevAppend configuration parameter in the task's IvDevel module, which you can set by using the Append parameter on the command line. For more information about this parameter, see the *IDOL Speech Server Reference*.

To generate a development score file

1. Gather together the required audio files for testing, including:
   - At least one file for each speaker that contains speech from that speaker only; aim to use a minimum of five minutes of speech for each speaker.

     > **NOTE:**
     > Do not use the same audio that you used to create the speaker templates.

   - Files that contain unknown speakers (those not in the training set).

     > **NOTE:**
     > It is important to use a substantial amount of unknown speaker data, from a wide range of speakers, to correctly tune the thresholds.

2. Create a list of the speaker templates. Each list entry must include the name of the speaker, and the name of their template file. Use the format:

   speakerLabel;templateFile

   For example:

   ```
   Brown;brown.iv
   Jones;jones.iv
   Smith;smith.iv
   ```

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
3. For each audio file, send an AddTask action to IDOL Speech Server, and set the following parameters:

Type	The task name. Set to **IvSpkIdDevelAudio**.
File	The audio file that contains the speaker example speech.
DataLabel	The name of the speaker that the audio is associated with.
TemplateList	The list file that specifies the set of speaker templates to use.
DevFile	The development file (.ivd) to create or update.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdDevelAudio&File=C:/Data/Brown
Speech4.wav&DataLabel=Brown&TemplateList=ListManager/speakers&DevFile=speakers.
ivd
```

This action uses port 15000 to instruct IDOL Speech Server, which is located on the local machine, to generate match statistics based on audio from the speaker named Brown, in the audio file BrownSpeech4.wav, against all of the speaker templates specified in the speakers list. The results are written to the speakers.ivd development file.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the .ivd file.

To process streamed audio, you use the same task (IvSpkIdDevelAudio), with the InputType parameter set to **Stream**. For more information, see the *IDOL Speech Server Reference*.

To estimate Speaker Template score thresholds

After you gather both positive and negative score statistics for each of the speaker templates, you can calculate the threshold associated with each speaker. This threshold is stored within the speaker template file.

You can do this for each speaker template individually, or across the whole set at once. The example given here shows the latter approach.

You can specify multiple template development files in a list file, or just a single development file. Again, the latter approach is shown here.

You can use the Bias parameter to bias the threshold calculated towards fewer false positives (at the likely cost of more misses), or the other way around. Increase the value of the Bias parameter to reduce false positives and increase *precision*, lower it to reduce misses and increase *recall*.

For details on other options associated with this task, see the *IDOL Speech Server Reference*.

- To estimate the thresholds for a set of speaker templates, given a single development score file, send an AddTask action to IDOL Speech Server, and set the following parameters:

Type	The task name. Set to **IvSpkIdDevelFinal**.
DevFile	The input template development file.
TemplateList	A list file that specifies the templates to use.

`Bias`	The bias setting to use when calculating thresholds.
`OutPath`	The output path for the updated speaker templates.
`OutExt`	The file extension for output speaker templates.

> **NOTE:**
> If you do not set either the `OutPath` or `OutExt` parameters, IDOL Speech Server overwrites the original templates.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdDevelFinal&DevFile=speakers.ivd&TemplateList=ListManager/speakers&Bias=0.2
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to use the development scores in `speakers.ivd` to calculated thresholds (using a `Bias` value of `0.2` when balancing recall against precision) for each speaker template specified in the speakers list.
IDOL Speech Server updates the template files in place to contain the threshold values calculated.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.

To modify the threshold of a single template

You can use the `IvSpkIdEditThresh` standard task to modify the threshold of a single template by specifying the template file (`.iv`).

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdEditThresh`**.
`TemplateFile`	The name of the template to modify.
`Thresh`	The value to use for the threshold.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdEditThresh&TemplateFile=brown.iv&Thresh=0.5
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to set the threshold of the `brown.iv` template file to `0.5`. IDOL Speech Server updates the template file in place to contain the new threshold value.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.

To retrieve information on a single template

You can use the `IvSpkIdInfo` standard task to write information on a specified template file to a log file.

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdInfo`**.
`TemplateFile`	The name of the template file to retrieve information for.
`Log`	The log file to write the information to.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdInfo&TemplateFile=brown.iv&Log=brown.log
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to write information on the `brown.iv` template file to the log file `brown.log`.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the log file.

Log File Example

```
<TEMPLATE_0>
  <NAME> test.atf </NAME>
  <THRESH_ENABLED> Yes </THRESH_ENABLED>
  <THRESH_VALUE> 1.158 </THRESH_VALUE>
</TEMPLATE>
```

This file shows some information about how the template was trained and optimized, along with information about the template. The log file includes the following fields:

`<TEMPLATE_0>`	The start of information on template 0, and so on.
`<NAME>`	The name associated with the template.
`<THRESH_ENABLED>`	Whether a score threshold is enabled for this template.
`<THRESH_VALUE>`	The score threshold that has been estimated for this template.

Package Templates

You can use a set of individual speaker template files during identification by supplying a list file that specifies the templates to use. However, it might be convenient to package these templates into a single speaker set file.

Using a speaker set file has several advantages:

- A single file simplifies the usage and distribution of the template set.
- The background model that is used during speaker training is packaged with the speaker templates, removing any chance of model mismatches if the base background model is subsequently updated.

You can use standard IDOL Speech Server tasks to add templates to an audio template set file (`.ats`), remove templates from a set file, modify the threshold of a single template stored within an audio template set file, or to return information on the contents of a set file or an individual template file.

To add templates to an audio template set file

The `IvSpkIdSetAdd` task takes one or more audio template files, and adds them to an audio template set file. If the set file already exists prior to running the task, IDOL Speech Server adds the templates to the existing set.

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdSetAdd`**.
`TemplateList`	A list file that specifies the templates to use, along with the name associated with each template.
`TemplateSet`	The name of the template set file.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdSetAdd&TemplateList=ListManager/speakers&TemplateSet=speakers.ivs
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to create the `speakers.ivs` template set file containing the templates listed by the `Listmanager/speakers` list file.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the speaker template set file.

To remove a template from an audio template set file

This task removes the named template from the audio template set. If the template named is not found within the set, a task error is given.

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdSetDelete`**.
`TemplateName`	The name associated with the template to remove.
`TemplateSet`	The template set file from which to remove the template.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdSetDelete&TemplateName=Brown&TemplateSet=speakers.ivs
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to remove the template associated with the name `Brown` from the `speakers.ivs` template set.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the speaker template set file.

To modify the threshold of a single template in an audio template set file

You can use the `IvSpkIdEditThresh` task to modify the threshold of a single template in an audio template set file.

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdEditThresh`**.
`TemplateName`	The name of the template to modify.
`TemplateSet`	The template set file to modify.
`Thresh`	The value to use for the threshold.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdEditThresh&TemplateName=Brown&TemplateSet=speakers.ivs&Thresh=0.5
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to set the threshold of the template associated with the name `Brown` in the `speakers.ivs` template set file to `0.5`.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the log file.

To retrieve information on an audio template set file

This task produces a log file that lists the contents of the specified audio template set file.

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`IvSpkIdInfo`**.
`TemplateSet`	The template set file to retrieve information for.
`Log`	The log file to write the information to.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkIdInfo&TemplateSet=speakers.ivs&Log=speakers.log
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to write information on the `speakers.ivs` template set file to the log file `speakers.log`.

You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the log file.

Log File Example

```
<COMPSELECT> 20 </COMPSELECT>
<NTEMPLATES> 3 </NTEMPLATES>
<TEMPLATES>
  <TEMPLATE_0>
    <NAME> Brown </NAME>
    <THRESH_ENABLED> Yes </THRESH_ENABLED>
    <THRESH_VALUE> 35.8945 </THRESH_VALUE>
  </TEMPLATE>
  <TEMPLATE_1>
    <NAME> Cameron </NAME>
    <THRESH_ENABLED> Yes </THRESH_ENABLED>
    <THRESH_VALUE> 36.25 </THRESH_VALUE>
  </TEMPLATE>
  <TEMPLATE_2>
    <NAME> Clegg </NAME>
    <THRESH_ENABLED> Yes </THRESH_ENABLED>
    <THRESH_VALUE> 36.5571 </THRESH_VALUE>
  </TEMPLATE>
</TEMPLATES>
```

This file shows some information about how the templates were trained and optimized, along with information about each template stored in the set. The log file includes the following fields:

`<COMPSELECT>`	How many base model components are used when generating the iVectors.
`<NTEMPLATES>`	The number of templates in the set.
`<TEMPLATE_0>`	The start of information on template 0, and so on.
`<NAME>`	The name associated with the template.
`<THRESH_ENABLED>`	Whether a score threshold is enabled for this template.
`<THRESH_VALUE>`	The score threshold that has been estimated for this template.

Identify Speakers in Audio

After you have trained a set of speaker templates, you can analyze audio to identify any sections where the trained speakers are present, by using the `IvSpkId` task.

You can specify the audio templates to use either by specifying a list as the value of the TemplateList parameter, or by specifying a template set as the value of the TemplateSet parameter. If you do not set any templates, IDOL Speech Server performs speaker segmentation and gender identification, but with no speaker labels.

To identify speakers in an audio file

- Send an AddTask action to IDOL Speech Server, and set the following parameters:

Type	The task name. Set to **IvSpkId**.
File	The audio file to process.
TemplateSet	The speaker template set file to use.
TemplateList	A list file that specifies a set of templates to use (if a set file is not specified in the TemplateSet parameter).
ClosedSet	Whether this is a closed-set test (by default, this parameter is set to False, resulting in an open-set test).
Out	The file to write the speaker identification results to.

  To process an audio stream instead of a file, you can set the InputType parameter to **Stream**, in which case you do not need to set the File parameter.

  You can set additional parameters. For details of the optional parameters, see the *IDOL Speech Server Reference*.

For example:

```
http://localhost:15000/action=AddTask&Type=IvSpkId&File=C:\Data\Speech.wav&TemplateSet=speakers.ats&ClosedSet=False&Out=results.sid
```

This action uses port 15000 to instruct IDOL Speech Server, which is located on the local machine, to search the Speech.wav file for speakers based on the template set file speakers.ats, and to write the identification results to the results.sid file. Because the test is set to be open-set, IDOL Speech Server marks sections where no speaker scores above their respective thresholds as Unknown_.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Find Recommended Language Resources to Use in a Task, on page 103.

Format of Speaker Identification Results

IDOL Speech Server supports two speaker identification output formats: CTM and XML.

The following example shows CTM output produced by the IvSpkId task.

```
1    A    0.000        0.520        Unknown_        NonSpeech_        0.000
1    A    0.520        10.030       Brown           MALE              3.540
1    A    10.550       0.080        Unknown_        NonSpeech_        0.000
```

```
1    A    10.630       9.460        Unknown_         FEMALE              0.000
1    A    20.090       6.150        Smith            MALE                6.983
```

From left to right, the columns in the CTM file contain:

- The channel ID (usually `1`)
- A fixed field, `A`
- The start time of the processed segment in seconds
- The duration of the processed segment in seconds
- The recognized speaker, or `Unknown_`.
- The gender of the speaker (or `NonSpeech_`, if a non-speech segment)
- The score for the recognized speaker (`0.000` if non-speech or an unknown speaker)

> **NOTE:**
> The score for an identified speaker represents how well the processed speech matches the template. Scores can be negative or positive depending on the type of score normalization used, but in all cases a higher value represents a score that is closer to the model.

The following example shows XML output with the mode set to default:

```
<sid_transcript>
       <sid_record>
              <start>0.000</start>
              <end>0.520</end>
              <label>Unknown_</label>
              <gender>NonSpeech_</gender>
              <score>0.000</score>
       </sid_record>
       <sid_record>
              <start>0.520</start>
              <end>10.550</end>
              <label>Brown</label>
              <gender>MALE</gender>
              <score>3.540</score>
       </sid_record>
</sid_transcript>
```

Troubleshooting

IDOL Speech Server provides several methods for troubleshooting performance issues.

- Generate Diagnostics, on the next page
- Improve Performance, on page 169
- Warning Messages, on the next page
- Error Messages, on page 170

Generate Diagnostics

You can configure all speaker identification-related tasks to generate a diagnostics file. The diagnostics file logs the task initialization, processing, and completion, along with any warnings.

To return diagnostic information for a task

- Include the `DiagLevel` and `DiagFile` parameters when you send the `AddTask` action. Set `DiagLevel` to a value greater than `0` to enable diagnostics. The higher the value, the more information is given. The largest value is `3`. Set `DiagFile` to the name of the file to write the diagnostic information to (if you do not set this parameter, IDOL Speech Server automatically generates an output file name).

For example:

```
http://localhost:15000/action=AddTask&Type=ivSpkID&File=C:\Data\Speech.wav&TemplateSet=speakers.ats&ClosedSet=False&Out=results.sid&DiagLevel=3&DiagFile=results.diag
```

This action uses port `15000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `IvSpkId` task on the `Speech.wav` file and write the results to the `results.ctm` file and the diagnostics information to `results.diag`.

> **NOTE:**
> The `IvSpkIdFeature`, `IvSpkIdSetAdd`, `IvSpkIdSetDelete`, and `IvSpkIdInfo` tasks do not support the creation of diagnostic files. However, the `IvSpkIdSetAdd`, `IvSpkIdSetDelete`, and `IvSpkIdInfo` tasks all generate a log file.

Warning Messages

Depending on the speaker ID task, IDOL Speech Server can return the following warnings.

Task	Warning message	Recommended action
`IvSpkIdTrain` `IvSpkIdTrainAudio`	Specified parameter file does not exist	Check that all the feature files listed in the `DataList` file exist.
	Failed to open ivector parameter file	Check that the read permission is set for all the feature files listed in the `DataList` file.
	Failed to read ivector parameter file header	Check that all the feature files listed in the `DataList` file are valid IVP files as created by IDOL Speech Server.
	NInputs in parameter file not consistent with previous file(s)	Ensure that all input feature files being used to train the templates were trained with the same configuration (for example, using the same iVector base pack).
	Input label file is longer (501 frames) than audio (500 frames)	Ensure that any label files provided to filter the input features match the length of the audio.

IvSpkIdDevel IvSpkIdDevelAudio	Segmentation window(s) set, but disabled as input label file(s) provided	If label files are provided for filtering the input features, do not attempt to specify the segmentation window.
	Could not read file (C:\data\myfile.ivp) - is it a valid ivector parameter (IVP) file?	Check that all the feature files listed in the DataList file are valid IVP files as created by IDOL Speech Server.
	Incorrect top result for segment (Smith, expected Brown)	Check the content and quality of the development audio file, as well as the audio files used to train the speaker models.
IvSpkIdDevelFinal	No development scores found for template [Brown] - threshold not set	Ensure that all speaker templates are included when you run the development tasks (ivDevel or ivDevelAudio).
	No positive examples for template Brown	Ensure that you provide positive audio examples (data from each speaker) for each speaker template.
	No negative examples for template Brown	Ensure that you provide negative audio examples (data from speakers other than just the trained speaker) for each speaker template.
	Brown shows significant overlap between TRUE and FALSE scores	Check the content and quality of the development audio files, as well as the audio files used to train the speaker models.

Improve Performance

Some common situations can reduce the performance of speaker identification.

Training and test data differ in audio quality	Try to use training data for each speaker that closely resembles the recording conditions and quality that you expect in the files that you want to process.
Unbalanced speaker thresholds	If IDOL Speech Server has too many false positives, use a higher Bias value when you finalize the speaker score thresholds (in the SpkIdDevelFinal task). If the false negative rate is too high, use a lower Bias value.
Insufficient training or optimization data	HPE recommends that you use at least five minutes of data for each speaker for training, and more if possible. A similar amount of data is required for each speaker for the development tasks if you are training score thresholds for open-set identification. If you use less data, this can compromise the general performance of the speaker templates.
Insufficient	This situation can lead to poor speaker thresholds, which in turn lead to excessive

'unknown' data used during optimization	false positives.
Very short audio segments are used in identification	Short audio segments can reduce the accuracy of speaker identification. Use the `MinSpeech` and `MinNonSpeech` parameters to specify the minimum size of audio segments, and the `DiscardShort` parameter to set the minimum segment size before discarding the result.

Error Messages

If you receive errors associated with actions (for example, incorrect parameters), see the individual parameters in the *IDOL Speech Server Reference* for the correct usage.

Chapter 13: Identify Languages in Audio

This section describes how to identify languages being spoken in an audio file.

- Overview 171
- Language Identification Modes 171
- Open Set Language Identification 172
- Configure Language Identification Tasks 172
- Run Language Identification 173
- Create Your Own Language Classifiers 175
- Troubleshooting 185

Overview

IDOL Speech Server can identify the languages that are being spoken in a section of audio.

You can use the language ID base classifier packs (available for both 8 kHz and 16 kHz audio) to perform language identification tasks straight out of the box. You can also create your own language classifiers, either to identify languages that are not covered by the pack, or to produce classifiers that are more closely matched to the properties of your target audio than the default classifiers.

You can train language classifiers for any spoken language, regardless of whether that language is supported for speech-to-text. You can train transcriptions with audio data only; transcriptions are not required.

> **NOTE:**
> To use the installed language classifier set for telephony, you must change the language ID base pack to `SYLS-tel`. You must also set the class list to be the classifier list file that is stored in the 8k classifiers directory, for example: `Classifiers-3.1-8k/classifiers.txt`.

> **NOTE:**
> If you know the languages that are likely to occur in your audio files, HPE recommends that you restrict the language classifier set to include only the classifiers for those languages. To do this, set the `LangList` parameter when you submit a language identification action.

For best performance, HPE recommends that you train a set of classifiers based on example data that is representative of the data that you want to analyze. If you use the base classifier pack, HPE recommends that you optimize the classifiers on your own data. For instructions, see Optimize the Language Identification Set, on page 179.

Language Identification Modes

Language identification can run in three modes:

- **Cumulative**. Cumulative language detection returns the running identification score at periodic intervals. This score is the score for all the audio from the start to the current point. This mode is the most reliable way to determine the language for a whole file (assuming it contains only one language), because the final result is calculated across the entire file.
- **Segmented**. Segmented language detection splits the incoming audio into fixed-sized chunks and returns the identification results for each chunk, independent of the results of neighboring chunks. You can use this mode to determine if there are multiple languages present, although it does not give the exact boundary points.
- **Boundary**. Boundary detection-based language recognition seeks to determine boundaries in the audio where the language changes, and returns the identification results between boundaries. Although this mode is potentially powerful, it can be more difficult to configure than the other two modes.

Open Set Language Identification

By default, IDOL Speech Server language identification uses closed set language classification.

In closed set language identification, IDOL Speech Server expects all the speech to belong to one of the trained languages. The server returns a result for all valid speech segments. If there is an unknown language in the audio, IDOL Speech Server identifies it as the known language that returns the highest language score.

You can also use open set language identification. In this case, the audio can contain speech from languages other than the trained set. Each trained language has a score threshold associated with it, trained during the optimization stage. If the score for the highest ranked classifier for a given segment of audio is lower than the associated threshold, IDOL Speech Server labels the section as **unknown** (or **UNK**).

The use of thresholds means that open set language identification might lead to genuine sections of known language speech being labelled as unknown. HPE recommends that you use open set identification only where unknown language data is likely to be an issue. You might also want to use this method if you want to identify whether the speech belongs to a particular language, and do not need to identify the other possible languages.

Configure Language Identification Tasks

To perform language identification tasks, IDOL Speech Server requires files from the Language ID resource pack. There are two versions of the pack available:

- **Language ID**. Use this pack to process 16 kHz data. The pack contains two component packs:
  - Base classifier pack. Contains pretrained language classifiers for 20 languages.
  - SYLS language pack. Contains files required to create language identification feature files.
- **Language ID Telephony**. Use this pack to process 8 kHz data. It contains an SYLS language pack required to create language identification feature files. A base classifier pack is not available for telephony data.

Download the packs from the HPE Download Center.

The Language ID pack is configured in the [SYLS] section of the configuration file. Before you run language identification tasks, ensure that the settings are correct in this section. For more information about how to configure language packs, see Configure Language Packs, on page 62.

For language identification and language pack optimization tasks, you can specify a subset of language classifiers to use from the base classifier pack. Use the LangList configuration parameter in the langid or lidoptimizer modules to specify the languages.

Run Language Identification

The process of running language identification tasks is very similar regardless of which mode it is run in. As such, the bulk of this section focuses on segmented identification, with significant differences for other modes described where appropriate.

To identify the languages in streamed audio, use the LangId task, with LIDMode set to the appropriate language identification mode. For more information about this standard tasks, see the *IDOL Speech Server Reference*.

Use the following procedure to identify the languages in an audio file.

To identify languages in an audio file

1. Create a list that contains the file names (including file extensions) of the classifiers to use.

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
2. Send an AddTask action to IDOL Speech Server, and set the following parameters:

Type	The task name. Set to **LangId**.
LidMode	The mode to use. Set to **Segmented** for segmented mode (this is the default), **Boundary** for boundary mode, or **Cumulative** for cumulative mode.
File	The audio file to process. To restrict processing to a section of the audio file, set the StartTime and EndTime parameters. For more information, see the *IDOL Speech Server Reference*).
Out	The file to write the language identification results to.

   - If you want to change the audio sample rate, or if you want to use your own custom classifiers, you must also set the ClassList parameter. You might also need to specify the ClassPath parameter, depending on the location of the classifier files. See the *IDOL Speech Server Reference* for more information.
   - If you use the base classifier pack, set the languages that you want to identify in the LangList configuration parameter in the langid module.
   - If you want to use open set language identification, you must also set the ClosedSet parameter to **False**. For more information about open set language identification, see Open Set Language Identification, on the previous page and the *IDOL Speech Server Reference*.

For example:

```
http://localhost:13000/action=AddTask&Type=LangId&LIDMode=Segmented&File=C:\Data\Speech.wav&ClassList=ListManager\OptClassSet&ClassPath=C:\LangID\&Out=SpeechLang1.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to identify languages in the `Speech.wav` file using the language classifiers specified in the `OptClassSet` list, and to write the identification results to the `SpeechLang1.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Find Recommended Language Resources to Use in a Task, on page 103.

IDOL Speech Server displays the results in XML format in your web browser. You can also open the .ctm file from the configured IDOL Speech Server temporary directory (or another location if you specified a path in the `Out` parameter).

The following is an example of the .ctm output produced by the `LangId` task in `Segmented` mode.

```
1    L1    0.00      30.58      English      1.000      1.252
1    L2    0.00      30.58      German       0.686      1.252
1    L3    0.00      30.58      French       0.550      1.252
1    L1    30.58     28.30      German       1.000      1.306
1    L2    30.58     28.30      English      0.562      1.306
1    L3    30.58     28.30      Italian      0.517      1.306
1    L1    58.88     31.12      English      1.000      1.295
1    L2    58.88     31.12      French       0.680      1.295
1    L3    58.88     31.12      German       0.511      1.295
```

From left to right, the columns in the .ctm file contain:

- The channel ID (usually `1`)
- The ranking of the language result at this time (`L1` is the top result, `L2` the next best, and so on)
- The start time of the processed segment in seconds
- The duration of the processed segment in seconds
- The identified language
- The score for this language (if normalized scores are being used, this score ranges from `0.0` to `1.0`; otherwise a log score is reported)
- The confidence for the highest-ranked decision in the current segment (`1.0` and above–the higher the score, the more confident the system is that `L1` is the correct answer)

The example shows a 90-second file being recognized in segments, each approximately 30 seconds in duration. For the first segment, English is the language that is identified as being the most likely (`L1`), followed by German (`L2`) and French (`L3`). For the next segment, German has the highest confidence score. For the final segment, English has the highest confidence score again.

In `Segmented` mode, it is common to see different results for each segment, because the language might change throughout the file. `Cumulative` mode assesses the most dominant language across the whole file, so you would not expect to see these changes.

The following example shows some of the same information displayed in XML format.

```
<lid_transcript>
	<lid_record>
		<start>0.000</start>
		<end>30.580</end>
		<label>English</label>
		<score>1.000</score>
		<confidence>1.252</confidence>
		<rank>1</rank>
	</lid_record>
	<lid_record>
		<start>0.000</start>
		<end>30.580</end>
		<label>German</label>
		<score>0.686</score>
		<confidence>1.252</confidence>
		<rank>2</rank>
	</lid_record>
</lid_transcript>
```

This output format is common to the `Segmented` and `Cumulative` modes. The output format for `Boundary` mode is similar, but the time points occur whenever a language change is detected, instead of after a fixed time period.

Create Your Own Language Classifiers

IDOL Speech Server requires a classifier file for each language that you want to identify. The base classifier pack provided with IDOL Speech Server covers the following languages for broadband (16 kHz) audio; for any other languages, or to process telephony (8 kHz) audio, you must create your own classifiers.

Arabic	Hebrew	Portuguese
Danish	Italian	Romanian
Dutch	Japanese	Russian
English	Korean	Slovak
French	Mandarin	Spanish
German	Persian	Swedish
Greek	Polish	

For each language, gather a set of audio files that contain speech from that language only. The audio must not include excessive music or noise, although it must closely resemble the quality of audio that you intend to process.

You must convert each audio file into a language identification feature (.lif) file.

Create Language Identification Feature Files

Follow the steps to convert audio files into language identification feature files.

To convert an audio file into a language identification feature file

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`LangIdFeature`**.
`File`	The audio file to process.
`Out`	The name of the feature file to create.

For example:

```
http://localhost:13000/action=AddTask&Type=LangIdFeature&File=C:\Data\FrenchSpeech.wav&Out=frenchSpeech.lif
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to create the `frenchSpeech.lif` file from the `FrenchSpeech.wav` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the language identification feature file. See Get Task Results, on page 105.

Performance Considerations

In the 10.11 release of IDOL Speech Server , language identification feature generation is based on a DNN decode, similar to that used in speech-to-text. This feature provides greater accuracy than decodes that use the previous acoustic model technology, but at the expense of slower processing speed.

Depending on your hardware specification, you might want to switch to using acoustic models instead of DNN decodes.

> **NOTE:**
> This process also applies if you are running language identification from a wav file or an audio stream, which generates a language ID feature stream as part of the process.

To run language identification without DNNs

1. **Disable DNN usage**. For all actions that use the base SYLS or SYLS-tel language packs, specify `DNNFile=None` in the action to disable DNN usage.
2. **Specify the language classifiers**. If you are using the base language classifiers in the 10.11 release, you must specify the non-DNN classifier list. To do this, change entries in the configuration file such as:

   ```
   ClList = $params.ClassList = Classifiers-3.0-16k/classifiers.txt
   ```

to:

```
ClList = $params.ClassList = Classifiers-3.0-16k/classifiers_noDNN.txt
```

> **NOTE:**
> If you disable the DNN when you process the test data, but you use the standard classifiers (or those trained based on DNN-generated features), the features do not match and the accuracy suffers. The same applies in the reverse situation (that is, if you use the non-DNN classifiers with DNN decoding enabled). Because of this, you must be consistent when you generate language identification features for the training, development, and evaluation stages.

Create the Language Classifier

You can use the language identification feature files to create a language classifier file.

To create a language classifier

1. Create a list that contains the names of all feature files. Include the file extensions but not the file paths.

   > **NOTE:**
   > The path is excluded so that you can move the data files at a later stage without needing to update the list. This is the case for most lists that IDOL Speech Server uses. You must provide the path in an action parameter when you send an action.

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
2. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`LangIdTrain`**.
`Context`	If you are using telephony (8 kHz) data, set `Context` to **`2`** for best results.
`DataList`	The list that specifies the feature files.
`DataPath`	The path to the directory that contains the feature files specified in the `DataList`.
`Out`	The name of the language classifier to create.

For example:

```
http://localhost:13000/action=AddTask&Type=LangIdTrain&DataList=ListManager/FrenchList&DataPath=C:\LangID\Data&Out=FRFR.lcf
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to use the feature files specified in the `FrenchList` list and stored in `C:\LangID\Data` to create the `FRFR.lcf` language classifier file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the language classifier file. See Get Task Results, on page 105.

Combine Classifiers into a Language Identification Set

You must combine the individual language classifiers into a single language identification set. To identify the language being spoken, IDOL Speech Server compares audio against this set.

To combine the language classifiers into a set

- Create a list of classifiers. Use the following format for each list entry:

  LanguageName;*BaseLanguage*;*ClassifierName*;*LanguageWeight*;*Threshold*

  where:

LanguageName	is the name that is reported in the results.
BaseLanguage	is the base language that this language classifier belongs to. You can use this option if you have multiple classifiers for dialects of the same language (for example, ENUK and ENUS). This value is used only during optimization. This column is optional. If you do not want to use base languages, add the following line to the file, above the first language row: `$baselangs;NO`
ClassifierName	is the name of the trained classifier file for this language. **NOTE:** It is not necessary to specify the full path to the classifier in this list, because you can use an action parameter to provide this information when you use the classifier set.
LanguageWeight	is the weight to apply to the language. IDOL Speech Server uses weights to scale the scores for each language before it compares them. You can leave this field empty before you run the `LangIdOptimize` task, which generates optimized weights (see Optimize the Language Identification Set, on the next page).
Threshold	is the language score threshold that IDOL Speech Server uses for this language. In open set language identification, IDOL Speech Server uses this threshold to determine whether the audio matches the language. In this case, if the language score is below the specified threshold, IDOL Speech Server returns the language as unknown. You can leave this field empty before you run the `LangIdOptimize` task, which generates optimized thresholds (see Optimize the Language Identification Set, on the next page).

  NOTE:
  Separate the three fields with semi-colons (;).

  For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.

The following example list contains four language classifiers: ENUK, ENUS, ESES, and FRFR.

```
$baselangs;YES
ENUK;EN;Classifiers-3.3-16k/ENUK.lcf
```

```
ENUS;EN;Classifiers-3.3-16k/ENUS.lcf
ESES;ES;Classifiers-3.3-16k/ESES.lcf
FRFR;FR;Classifiers-3.3-16k/FRFR.lcf
```

> **TIP:**
> You can use the optional `$sfreq` parameter at the start of the classifier list to identify the sample frequency for the classifier set.
>
> For example:
>
> ```
> $sfreq;16000
> ENUK;EN;ENUK.lcf;1.003
> ESES;ES;ESES.lcf;0.985
> ```
>
> This example sets the sample frequency for this set to 16 kHz, and then lists the two classifiers in the set. In addition, usage is restricted to audio of the specified sample frequency, to prevent accidental mismatches.

Optimize the Language Identification Set

After you prepare the classifier list, you can run a language identification task. However, before you do so, HPE recommends that you optimize the language weights and score thresholds in the classifier list.

Optimization balances the language models. After training, some classifiers might be stronger than others owing to the properties of the training material and the languages themselves. The optimization process weights the language models so that weaker languages have increased recall, without compromising the precision for stronger language models.

Optimization also calculates score thresholds for open set language identification. In open set language identification, IDOL Speech Server checks whether a particular identified language matches the audio with a score above the specified threshold. If the language score is lower than the threshold, it returns the language identification as **unknown**.

The optimization process requires several audio files, each from a known language. It analyzes sections throughout each file, and determines whether each section was identified correctly. If a particular language is frequently not identified, IDOL Speech Server usually increases the weighting of the language classifier. Conversely, if IDOL Speech Server frequently identifies a particular language more often than it occurs, it usually reduces the weighting of the language classifier.

To optimize a language set

1. Gather the data to use. You need at least one audio file for each language represented in the classifier list. This audio data must not be the same data that you used to train the language classifiers.
2. Convert each audio file into a language identification feature file (see Create Language Identification Feature Files, on page 176).
3. Create a list that contains the name of each feature file and a label that specifies the language that the file represents. Use the following format:

   LanguageName;LifFilename

You can specify multiple files for each language. Either type the file names in a comma-separated list, or add multiple instances of a language. For example:

```
ENUK;english1.lif
ENUK;english2.lif
ENUS;us-english1.lif,us-english2.lif
ESES;spanish.lif
FRFR;french.lif
```

You can also use data for unknown languages, to optimize open set language identification. In this case, use the tag unknown. For example:

```
Unknown;unknown1.lif
Unknown;unknown2.lif
```

For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.

4. If you intend to write the optimized weights to a list in the list manager, you must create the list before you specify it in the following step.
5. Run the optimization task. Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

`Type`	The task name. Set to **`LangIdOptimize`**.
`DataList`	The list that specifies the feature files.
`DataPath`	The path to the directory that contains the feature files listed in the `DataList`.
`ClassList`	The list that specifies the classifiers (see Combine Classifiers into a Language Identification Set, on page 178).
`ClassPath`	The path to the directory that contains the language classifiers.
`Out`	The classifier list to write the optimized weights to.

For example:

```
http://localhost:13000/action=AddTask&Type=LangIdOptimize&DataList=ListManager/OptList&DataPath=C:/LangID/Data&ClassList=ListManager/ClassList&ClassPath=C:/LangID/class&Out=ListManager/OptClassSet
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to test the feature files listed in the `OptList` list against the classifiers specified in the `ClassList` list and then generate an updated classifier list–`OptClassSet`.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the updated classifier file. See Get Task Results, on page 105.

The `LangIdOptimize` action also generates a log file that contains information about the optimization process. See Review the Optimization Log File, on the next page.

Review the Optimization Log File

When you run the classifier set optimization task, IDOL Speech Server writes information about the optimization performance to a log file. To specify the log file, use the `OutputLog` parameter in the `[lidoptimizer]` section of the tasks configuration file (`speechserver-tasks.cfg`). To retrieve this log file, use the `GetResults` action, and set the `Token` action parameter to the token for the optimization task.

Related Topics

- Find Recommended Language Resources to Use in a Task, on page 103

The log file shows how the language classifiers are performing on each iteration of the optimization. The `<lidoptimize_iteration_record>` summarizes the system performance across all languages for a specific iteration. For example:

```
<lidoptimize_iteration_record>
   <iteration>1</iteration>
   <learning_rate>1.5</learning_rate>
   <avg_score>0.968</avg_score>
   <min_score>0.848</min_score>
   <std_dev>0.00917</std_dev>
   <wt_nincreased>12</wt_nincreased>
   <wt_increased>ARMSA,DEDE,ELGR,ENUK,ESES,ESLA,FRFR,HBIL,ITIT,RORO,RURU,SKSK</wt_
increased>
   <wt_ndecreased>10</wt_ndecreased>
   <wt_decreased>DADK,ENUS,FAIR,JAJP,KOKR,NLNL,PLPL,PTBR,SVSE,ZHCN</wt_decreased>
   <wt_avgchange>-0.0131</wt_avgchange>
   <wt_avgchange_abs>0.0327</wt_avgchange_abs>
   <result_delta>0</result_delta>
   <best_so_far>Yes</best_so_far>
</lidoptimize_iteration_record>
```

The record contains the following elements.

`iteration`	The current iteration of the optimization process.
`learning_rate`	The current learning rate of the algorithm. The learning rate decreases throughout the process.
`avg_score`	The average correct score across all languages, with the current weighting.
`min_score`	The score of the worst performing language classifier.
`std_dev`	The standard deviation across the classifier scores.
`wt_nincreased`	The number of languages whose weights were increased.
`wt_increased`	The language pack codes for the languages whose weights were increased.
`wt_ndecreased`	The number of languages whose weights were decreased.
`wt_decreased`	The language pack codes for the languages whose weights were decreased.

wt_avgchange	The average weight change in real terms.
wt_avgchange_abs	The average weight change in absolute terms.
result_delta	The delta in the result when compared to the previous iteration.
best_so_far	Whether this iteration was the best iteration.

The aim of the optimization process is to optimize the values for avg_score, min_score, and std_dev.

The log file also contains <lidoptimize_lang_record> items. This item shows the performance of a specific language at a specified optimization iteration. For example:

```
<lidoptimize_lang_record>
   <iteration>3</iteration>
   <lang>ENUK</lang>
   <weight>0.991</weight>
   <score>0.933</score>
   <threshold>-1.472</threshold>
   <threshold_missRate>0.289</threshold_missRate>
   <threshold_falseRate>0.020</threshold_falseRate>
   <threshold_recall>0.711</threshold_recall>
   <threshold_precision>0.590</threshold_precision>
   <threshold_fmeasure>0.645</threshold_fmeasure>
   <data_avg_false>0.648</data_avg_false>
   <data_max_false>0.940</data_max_false>
   <class_avg_false>0.576</class_avg_false>
   <class_max_false>0.936</class_max_false>
   <closest_imposter>ENUS (58.3%)</closest_imposter>
   <rel_change>0.019</rel_change>
   <rel_change_str>INCREASED</rel_change_str>
   <next_weight>1.009</next_weight>
   <conf_analysis>
      <true_conf>
         <n_confs>168</n_confs>
         <min_conf>1.008</min_conf>
         <vlow_conf>1.067</vlow_conf>
         <low_conf>1.128</low_conf>
         <med_conf>1.217</med_conf>
         <high_conf>1.335</high_conf>
         <vhigh_conf>1.394</vhigh_conf>
         <max_conf>1.542</max_conf>
      </true_conf>
      <false_conf>
         <n_confs>22</n_confs>
         <min_conf>1.001</min_conf>
         <vlow_conf>1.010</vlow_conf>
         <low_conf>1.031</low_conf>
         <med_conf>1.056</med_conf>
         <high_conf>1.096</high_conf>
         <vhigh_conf>1.154</vhigh_conf>
```

```
            <max_conf>1.198</max_conf>
         </false_conf>
      </conf_analysis>
      <score_analysis>
         <true_score>
            <n_scores>180</n_scores>
            <min_score>-2.485</min_score>
            <vlow_score>-1.589</vlow_score>
            <low_score>-1.500</low_score>
            <med_score>-1.417</med_score>
            <high_score>-1.337</high_score>
            <vhigh_score>-1.277</vhigh_score>
            <max_score>-1.200</max_score>
         </true_score>
         <false_score>
            <n_scores>4487</n_scores>
            <min_score>-4.741</min_score>
            <vlow_score>-2.211</vlow_score>
            <low_score>-2.067</low_score>
            <med_score>-1.928</med_score>
            <high_score>-1.816</high_score>
            <vhigh_score>-1.726</vhigh_score>
            <max_score>-1.215</max_score>
         </false_score>
      </score_analysis>
</lidoptimize_lang_record>
```

The record item shows the existing weight and the new weight for the language, as well as some information on intermediate values used to calculate it. The record also gives information on how the language performed on the last iteration, information about confusability and the closest imposter for the language, and details about the spread of confidence (from the best result to the worst).

`iteration`	The current iteration of the optimization process.
`lang`	The language that this record represents.
`weight`	The weight for this language classifier at this iteration.
`score`	The correct rate for this classifier at this iteration (that is, is the proportion of audio segments for this language that were correctly identified).
`threshold`	The score threshold for this classifier at this iteration (that is, the language score that represents a good probability that the audio matches the language).

`threshold_missRate`
`threshold_falseRate`
`threshold_recall`
`threshold_precision`
`threshold_fmeasure`
	The statistical information for the threshold calculation:
	• `missrate`. The proportion of true language segments that were not identified.
	• `falserate`. The proportion of false language segments that were incorrectly identified as the specified language.

	• recall. The fraction of true positives identified out of all instances of the true language. • precision. The fraction of true positives identified out of all the instances identified as the language. • fmeasure. A statistical measure based on the precision and recall.
data_avg_false	The average score for an incorrect language classifier for the data for the current language, relative to the current classifier score. The maximum value is 1.0, which suggests that the incorrect language scores best on the data). IDOL Speech Server uses this value during weight estimation.
data_max_false	The highest score for an incorrect language classifier for the data for the current language, relative to the current classifier score. The maximum value is 1.0, which suggests that the incorrect language scores best on the data). IDOL Speech Server uses this value during weight estimation.
class_avg_false	The average score for the current language classifier computed across all other language data examples. The score is relative to the score of the true classifier for each data example. IDOL Speech Server uses this value during weight estimation.
class_max_false	The highest score for the current language classifier computed across all other language data examples. The score is relative to the score of the true classifier for each data example. IDOL Speech Server uses this value during weight estimation.
closest_imposter	The incorrect language that most closely matched the language.
rel_change	The amount that the weight changed by.
rel_change_str	The strength of the change in weight.
next_weight	The new weight for this classifier, for the next iteration.
conf_analysis	The conf_analysis section gives details of the confidence values seen for different hits for this language. It contains a section for true hits (true_conf) and a section for false positives (false_conf). • n_confs. The number of confidence values recorded in this section. • min_conf. The minimum confidence seen for a hit in this section. • vlow_conf. A very low confidence value seen for a hit in this section. • low_conf. A low confidence value seen for a hit in this section. • medium_conf. A medium confidence value seen for a hit. in this section • high_conf. A high confidence value seen for a hit in this section. • vhigh_conf. A very high confidence value seen for a hit in this section.

- `max_conf`. The maximum confidence value seen for a hit in this section.

`score_analysis`

The `score_analysis` section gives details of the language score values seen for different hits for this language. It contains a section for true hits (`true_score`) and a section for false positives (`false_score`).

- `n_scores`. The number of language score values recorded in this section.
- `min_score`. The minimum language score seen for a hit in this section.
- `vlow_score`. A very low language score value seen for a hit in this section.
- `low_score`. A low language score value seen for a hit in this section.
- `medium_score`. A medium language score value seen for a hit. in this section
- `high_score`. A high language score value seen for a hit in this section.
- `vhigh_score`. A very high language score value seen for a hit in this section.
- `max_score`. The maximum language score value seen for a hit in this section.

After all optimization iterations are complete, the `<lidoptimize_result>` log item reports the best iteration. For example:

```
<lidoptimize_result>
       <best_iteration>4</best_iteration>
</lidoptimize_result>
```

IDOL Speech Server writes the language classifier weights for the best iteration to the classifier list file. When you use this file to run language identification tasks, IDOL Speech Server balances the classifier scores based on the new, trained weights.

Troubleshooting

Some common situations can reduce the performance of language identification. For example:

Music and noise in audio

Significant amounts of music and noise in audio can seriously impact the language identification results.

To reduce the impact, you can enable speech detection and then skip all audio that is not identified as speech.

1. Set `DetectSpeech=True` in the `frontend` module configuration that the language identification task uses.
2. Set `ZeroSilFrames=True` in the `normalizer` module configuration section that the language identification task uses.

> **NOTE:** When you train a language classifier, you can set these

	parameters in the modules used by the `LangIdFeature` task.
Poorly balanced language classifiers	If particular languages perform significantly better than others, or IDOL Speech Server consistently misidentifies sections as particular languages (false positive results), the language classifiers might not be optimally balanced. To resolve this issue, you can repeat the optimization of the language classifier set using extra data (such as the files being processed when you detected the problem) to update the classifier weights. You can also manually adjust the weighting by lowering the weights of classifiers that are being over-identified (false positives), and raising the weights of classifiers for languages that are frequently not identified (false negatives).
Mismatched audio quality between training, optimization, and identification	The audio that you use for training and optimization must closely match the audio that is processed for identification, in terms of recording type, quality, accents, and so on. Any significant mismatch in the data degrades performance.
Poor detection of language boundaries	You can use several parameters (such as `WindowSize`, `SegmentStepSize`, `SegmentSmoothWin`, `SegmentThreshold`, and so on) to tune the boundary detection algorithm. For more details about the parameters, see the *IDOL Speech Server Reference*.
False hits in open set language identification.	If too many audio segments from unknown languages are returning as results for a known languages, you might want to repeat the optimization stage, providing data that is representative of the data you are using to run identification. Additionally, if misses (true language sections labeled as unknown) are not important, you can use the `ThresholdScale` parameter in the `langid` module configuration to modify the thresholds. Decreasing the threshold scale increases the threshold, which means fewer false hits.
Missing results in open set language identification	If too many audio segments from known languages are return as unknown results, you might want to repeat the optimization stage, providing data that is representative of the data you are using to run identification. Additionally, you can use the `ThresholdScale` parameter in the `langid` module configuration to modify the thresholds. Increasing the threshold scale decreases the threshold, which means fewer misses.

Error Messages

If you receive errors associated with actions (for example, incorrect parameters), see the individual parameters in the *IDOL Speech Server Reference* for the correct usage.

Chapter 14: Audio Fingerprint Identification

This section describes how to search for distinctive features (audio fingerprints) in audio data.

- Overview 187
- Define the AFP Database 187
- Manage AFP Databases 189
- Detect Indexed Clips in Audio 193
- Detect Repeated Audio Sections 196
- Troubleshooting 196

Overview

Audio fingerprint (AFP) identification is the process of analyzing audio data to identify occurrences of known audio clips, such as specific pieces of music or particular adverts. This process is useful for detecting when specific adverts occur, to check for copyrighted music being played, to pick out commercial jingles, and so on.

The first step is to build a database of the audio clips to identify. Each clip is represented in the database by a sequence of distinctive features. IDOL Speech Server can analyze incoming audio to detect occurrences of the stored clips. IDOL Speech Server compares the target audio to the database clips and identifies sections that closely resemble the database clips.

IDOL Speech Server supports two approaches to performing audio fingerprinting:

- The first approach is based on acoustic landmark matching. This approach is highly scalable, and very fast.
- The second approach is based on acoustic template matching. Although this approach is not quite so scalable (suitable for databases that are tens of hours in size, not hundreds), it is more robust to reverberation, echo, and other acoustic distortions.

The following section covers only the first approach in detail. However, for all the audio fingerprinting tasks except `AfpDatabaseOptimize`, you can set the `AfpMode` parameter to `robust` to use the template based approach. For more information, see the *IDOL Speech Server Reference*.

Define the AFP Database

When you perform any AFP operations, you must specify the AFP database to use. You can specify the database in two different ways:

- Set the `Pack` and `PackDir` action parameters when you send an action

  `Pack` The name of the database.

  `PackDir` The path to the directory that contains the database files. If this directory does not already exist, manually create it before you send the action. If the database does not yet exist, IDOL Speech Server creates the database in this directory.

- Define the database in the tasks configuration file, and then set the `AfpDb` action parameter to the name of this configuration section when you send an action

Define the AFP Database in the Tasks Configuration File

You can define individual AFP databases in the tasks configuration file (`speechserver-tasks.cfg`).

To define a database

1. Open the tasks configuration file in a text editor.
2. In the `[Resources]` section, add the name of the database to the list. Prefix the database name with `fpdb:`. For example:

   ```
   [Resources]
   0=fpdb:AFP
   1=fpdb:ADVERTS
   2=fpdb:TEMPLATES
   ```

   > **NOTE:**
   > The `[Resources]` section also contains all other resource types used by the server, such as language packs used by the speech-to-text operation.

3. Below the list, type the name of the database inside square brackets to create a configuration section for the database. For example:

   ```
   [Adverts]
   ```

4. In the new configuration section, add the following parameters.

   `Pack` The name of the database.

   `PackDir` The path to the directory that contains the database files. If this directory does not already exist, manually create it before you send the action. If the database does not yet exist, IDOL Speech Server creates the database in this directory.

   `FxxCacheSize` The size (in megabytes) of the cache for the fingerprint indexing file.

   `TtxCacheSize` The size (in megabytes) of the cache for the FPDB time track indexing file.

   > **NOTE:**
   > The `FxxCacheSize` and `TtxCacheSize` parameters are applicable only for the FPDB resource type (that is, for the landmark-based databases).

For example:

```
[ADVERTS]
Pack = Adverts
PackDir = C:\resources
FxxCacheSize = 2
TtxCacheSize = 200
```

For more information about these configuration parameters, see the *IDOL Speech Server Reference*.

When you send the task action, set the `AfpDb` action parameter to the name of the database configuration section. For example, `AfpDb=ADVERTS`.

> **NOTE:**
> You do not need to set the `Pack` and `PackDir` action parameters, which makes it easier to switch between databases using just the database name.

Manage AFP Databases

IDOL Speech Server allows you to:

- Add new audio clips to an AFP database.
- Remove clips from an AFP database.
- Optimize the indexing and storage of the AFP database.
- Retrieve information about the contents of an AFP database.

Add Clips to a Database

You must add all clips to recognize to an AFP database.

To add an audio clip to an AFP database

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`AfpAddTrack`**.

  `File` The audio clip to index into the AFP database.

  To restrict processing to a section of the audio file, set the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

  `Tag` The name to return in the results if the clip is identified in the audio.

  To add the clip to a database that is already defined in the tasks configuration file, you must set the `AfpDb` parameter. If the database is not defined, set both the `Pack` and `PackDir` parameters instead.

  `AfpDb` The name of a database that is defined in the tasks configuration file.

  `Pack` The name of a database that is not defined in the tasks configuration file.

  `PackDir` The path to the directory that contains the database files. If this directory does not already exist, manually create it before you send the action. If the database does not

yet exist, IDOL Speech Server creates the database in this directory.

For example:

```
http://localhost:13000/action=AddTask&Type=AfpAddTrack&File=C:\Data\Jingle.wav&Tag=MyCompanyJingle&PackDir=C:\resources&Pack=Adverts
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to index the audio clip `Jingle.wav` into the `Adverts` database. If the `Adverts` database does not already exist, IDOL Speech Server creates it. If the clip is subsequently identified in an audio file, IDOL Speech Server identifies it with the tag `MyCompanyJingle`.

This action returns a token. You can use the token to check the task status. See Check the Status of a Task, on page 87.

HPE recommends that you optimize the database after you add new clips (see Optimize a Database, on the next page). The optimizing procedure improves the database lookup functions.

To add a streamed audio track to the database, use the `AfpAddTrack` task with the `InputType` parameter set to **`Stream`**. For more details, see the *IDOL Speech Server Reference*.

Remove Clips from a Database

You can remove audio clips that are already indexed in a database.

To remove an audio clip from an AFP database

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name. Set to **`AfpRemoveTrack`**.

  `Tag` The name to return in the results if the clip is identified in the audio.

  To remove the clip from a database that is defined in the IDOL Speech Server tasks configuration file, you must set the `AfpDb` parameter. If the database is not defined, set both the `Pack` and `PackDir` parameters instead.

  `AfpDb` The name of a database that is defined in the tasks configuration file.

  `Pack` The name of a database that is not defined in the tasks configuration file.

  `PackDir` The path to the directory that contains the database files.

For example:

```
http://localhost:13000/action=AddTask&Type=AfpRemoveTrack&Tag=MyCompanyJingle&Pack=Adverts&PackDir=C:\resources
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to remove the audio clip with the tag `MyCompanyJingle` from the `Adverts` database.

This action returns a token. You can use the token to check the task status. See Check the Status of a Task, on page 87.

After you add and remove tracks to modify an AFP database, HPE recommends that you optimize the internal indexing of the database (see Optimize a Database, on the next page). Before you optimize the

database, any tracks that you have deleted using the AfpRemoveTrack task are labeled for deletion, but still take up indexing space internally. To remove these files permanently, you must run the optimization task.

Optimize a Database

You can optimize a database to permanently remove tracks labeled for deletion, and to optimize lookup functions.

> **NOTE:**
> This step is applicable only if you are using the landmark audio fingerprinting approach (when the AFPMode parameter is set to the default value, **standard**). It is not supported when running the template-based approach (AFPMode set to **robust**).

To optimize an AFP database

- Send an AddTask action to IDOL Speech Server, and set the following parameters:

  Type — The task type. Set to **AfpDatabaseOptimize**.

  To optimize a database that is defined in the IDOL Speech Server tasks configuration file, you must set the AfpDb parameter. If the database is not defined, set both the Pack and PackDir parameters instead.

  AfpDb — The name of a database that is defined in the tasks configuration file.

  Pack — The name of a database that is not defined in the tasks configuration file.

  PackDir — The path to the directory that contains the database files.

For example:

```
http://localhost:13000/action=AddTask&Type=AfpDatabaseOptimize&Pack=Adverts&PackDir=C:\resources
```

This action uses port 13000 to instruct IDOL Speech Server, which is located on the local machine, to optimize the Adverts database that is located in C:\resources.

This action returns a token. You can use the token to check the task status. See Check the Status of a Task, on page 87.

View the Database Contents

You can return a list of all tracks in an AFP database.

To return the contents of an AFP database

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task type. Set to **`AfpDatabaseInfo`**.

  `Out` The log file to write the database contents to.

  To view the contents of a database that is defined in the IDOL Speech Server tasks configuration file, you must set the `AfpDb` parameter. If the database is not defined, set both the `Pack` and `PackDir` parameters instead.

  `AfpDb` The name of a database that is defined in the tasks configuration file.

  `Pack` The name of a database that is not defined in the tasks configuration file.

  `PackDir` The path to the directory that contains the database files.

For example:

```
http://localhost:13000/action=AddTask&Type=AfpDatabaseInfo&Pack=Adverts&PackDir=C:\
resources&Out=AdvertsDb.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to write the contents of the `Adverts` database to the `AdvertsDb.ctm` file.

> **NOTE:**
> By default, the `AfpDatabaseInfo` task returns details for all AFP databases. You can restrict it to standard or template (robust) databases by using the `AFPMode` parameter. For more information, refer to the *IDOL Speech Server Reference*.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the database contents file. See Find Recommended Language Resources to Use in a Task, on page 103.

The following XML is an example of a database information log file:

```
<afpdb_info>
   <afp_standard_mode>
      <ntracks>1</ntracks>
      <total_length>1896</total_length>
      <afpdb_track_record>
         <id>0</id>
         <tag>MyAdvert1</tag>
         <start>0</start>
         <length>1896</length>
         <features>76</features>
         <feature_rate>0.040</feature_rate>
      </afpdb_track_record>
   </afp_standard_mode>
   <afp_robust_mode>
      <ntracks>1</ntracks>
      <total_length>1920</total_length>
```

```
      <afpdb_track_record>
         <id>0</id>
         <tag>MyAdvert1</tag>
         <start>0</start>
         <length>1920</length>
         <features>88</features>
         <feature_rate>0.046</feature_rate>
      </afpdb_track_record>
      <afpdb_track_record>
         <id>1</id>
         <tag>MyAdvert2</tag>
         <start>0</start>
         <length>1452</length>
         <features>71</features>
         <feature_rate>0.049</feature_rate>
      </afpdb_track_record>
   </afp_robust_mode>
</afpdb_info>
```

This example shows that the database contains three tracks: `MyAdvert1` (which uses the standard AFP mode), and `MyAdvert2` and `MyAdvert2` (which both use the template AFP mode).

It displays the length of each track (in frames, where 1 frame is 10 milliseconds). It also displays the start point of the track with respect to the original audio that was indexed. This value is usually 0. However, if the audio that was indexed was longer than four minutes, it is stored in several four-minute chunks.

> **NOTE:**
> If you attempt to delete a clip that has been split into chunks, all the chunks are deleted (all share the same tag).

Detect Indexed Clips in Audio

IDOL Speech Server can search an audio file or stream for clips that are present in an AFP database. In both cases, you use the `AfpMatch` task.

For more details about this standard task, see the *IDOL Speech Server Reference*.

To search audio for known clips

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task type. Set to **`AfpMatch`**.

  `File` The audio file to search.

  To restrict processing to a section of the audio file, you can use the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

  `Out` The file to write the search results to.

  To use a database that is defined in the IDOL Speech Server tasks configuration file, you must set the `AfpDb` parameter. If the database is not defined, set both the `Pack` and `PackDir` parameters instead.

  `AfpDb` The name of a database that is defined in the tasks configuration file.

  `Pack` The name of a database that is not defined in the tasks configuration file.

  `PackDir` The path to the directory that contains the database files.

For example:

```
http://localhost:13000/action=AddTask&Type=AfpMatch&File=C:\Data\Sample.wav&Pack=Adverts&PackDir=C:\resources&Out=SearchResults.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to search the `Sample.wav` file for sections that match audio clips in the `Adverts` database, and to write the results to the `SearchResults.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Get Task Results, on page 105.

Because the AFP task produces results while it processes the audio data, you can retrieve results before the task is complete.

The results are stored in CTM format, but you can send the `GetResults` action to view them in either CTM or XML format. An example of an AFP result in CTM format is:

```
1  1  2994.75  0.84  ADVERT1  5.59  2991.32  2995.08  2990.96  14  21
1  1  2995.59  0.55  ADVERT1  4.91  2991.32  2996.00  2990.96  15  23
1  1  2996.14  0.73  ADVERT1  5.30  2991.32  2996.60  2990.96  18  28
1  1  2996.87  0.52  ADVERT1  5.54  2991.32  2996.92  2990.96  20  31
1  1  2997.39  1.11  ADVERT1  4.82  2991.32  2997.96  2990.96  21  32
1  1  2998.50  1.22  ADVERT1  4.68  2991.32  2998.80  2990.96  23  35
1  1  2999.72  1.14  ADVERT1  4.45  2991.32  3000.08  2990.96  25  39
1  1  3000.86  2.35  ADVERT1  3.88  2991.32  3002.92  2990.96  29  45
```

From left to right, the columns in the output data file contain:

- The channel ID (usually 1).
- The rank of the current result (1 is the top ranked result for a section).
- The start time of the audio section that produced this result update.

  > **NOTE:**
  > The start time of the entire matched section is given in a different column.

- The duration of the audio section that produced this result update.
- The tag for the recognized database audio clip.
- The hit rate for this matched section (the number of audio fingerprint feature hits per second).
- The start point in the audio of the matched section.
- The current end point of the matched section (which might subsequently be updated).
- The estimated start point of the database clip in the audio (this is not the same as the start of the matched section, because the match might not begin from the start of the database clip).
- The number of hits in the matched section; each hit is the identification of an audio fingerprint feature.
- The percentage of audio fingerprint features in the database clip that have been identified in the matching section of the target audio.

You should view each result line as an update to an ongoing match, rather than a complete result. While IDOL Speech Server processes the audio, it might return multiple results for the same track, starting at the same point in the audio. In this case, the number of hits increases between successive results, and the current end point of the match increases.

The final result in such a sequence is the complete section match result for the specified hypothesis. For example, in the previous example, the match of the reference track ADVERT1 completes with the result:

```
 1   1   3000.86    2.35    ADVERT1    3.88    2991.32    3002.92    2990.96    29   45
```

This result shows that the processed audio matched the audio for the track ADVERT1 stored in the database between 2991.32s and 3002.92s. The estimated start point of the ADVERT1 data is actually 2990.96s (which suggests that the first 0.36s of the file was not matched). The match scored 29 hits, which is 45% of the total audio fingerprint features in the database clip. The last section of audio analyzed that contributed to this match was from 3000.86s to 3003.21s.

The following example shows the final output from the previous ADVERT1 match in XML format:

```
<afp_record>
       <rank>1</rank>
       <label>ADVERT1</label>
       <start>2991.32</start>
       <end>3002.92</end>
       <output>3003.210</output>
       <eststart>2990.96</eststart>
       <hits>29</hits>
       <score>45</score>
       <hitrate>2.636</hitrate>
       <scorerate>0.000</scorerate>
```

```
        <scoreavg>0.000</scoreavg>
</afp_record>
```

The <output> tags record the time that the final result was produced (the endpoint for processing the target audio, not just the matching section)

The <scorerate> and <scoreavg> tags are reserved for future functionality and always contain the value 0.000.

Detect Repeated Audio Sections

The most straightforward use of the AFP functionality is to build up a database of clips, and then search for those clips in unseen audio. However, there are other ways of making use of AFP functionality. For example, you can search a large collection of audio data for repeated sections, to help to identify features such as jingles, recorded messages, or adverts.

To detect repeated audio sections

1. Index all the audio data into the AFP database.
2. Perform AFP identification across all the data.
3. Look for any matches that are not from the same location in the same file. You could narrow your search by considering sections over a certain length, and that are matched in other locations within the audio data.

IDOL Speech Server cannot directly identify repeated sections that meet your criteria, but it does provide the underlying AFP identification results from which you can determine such information.

Related Topics

- Add Clips to a Database, on page 189
- Optimize a Database, on page 191
- Detect Indexed Clips in Audio, on page 193

Troubleshooting

The following situations can lead to reduced performance in AFP identification.

Low fingerprint rate	When you index a clip into the database, you can view the log file to see the number of fingerprints that have been created from the clip. If the fingerprint rate is too low (given that there is a threshold on fingerprint hit rate during the identification stage) the clip struggles to match. A low fingerprinting rate might be due to the nature of the audio clip being indexed. If the rate is too low, there are several fingerprinting parameters that you can alter to lead to higher rates of fingerprint creation. For information about parameters, see the *IDOL Speech Server Reference*.
Considerable noise and distortion	Although the algorithm is robust to some amount of noise, the identification rate is likely to suffer if the audio being matched is distorted or has a considerable amount of noise compared to the indexed clip.

Time-warping of audio data	The identification rate is likely to suffer if the audio being matched has dropped samples, or otherwise has a different rate to the original clip (for example, if it has been sped up or slowed down).

Chapter 15: Audio Security

This section describes how IDOL Speech Server can detect security-related sounds, such as alarms or breaking glass, in an audio file or stream.

- Overview 199
- Configure Audio Security Tasks 199
- Create the Alarm Database 199
- Run Audio Security 200

Overview

IDOL Speech Server can detect and label segments of audio that contain security-related sounds, including alarms, breaking glass, screams, or gunshots. This functionality is available as the `audiosec` module, which is implemented in the `AudioSecurity` standard task.

The `audiosec` module can also identify the type of alarm that it has detected. To identify an alarm, it must match an alarm template that exists in the database. You must use an audio sample of the alarm to create the template. If a corresponding template does not exist, the task produces a result with the label `<UNKNOWN-ALARM>`.

> **NOTE:**
> To detect breaking glass, HPE strongly recommends that the audio sampling rate is 16 kHz.

Configure Audio Security Tasks

To perform audio security tasks, IDOL Speech Server requires the `EVENTS` pack from the Audio Security resource pack. You can download the Audio Security pack from the HPE Download Center.

Before you run audio security tasks, you must ensure that the files are specified in the relevant configuration section.

- Configure the `EVENTS` pack in the `[Resources]` section. For more information about how to configure resource packs, see Configure Language Packs, on page 62.
- In the `audiosec` module for the task, set the `Model` parameter to the `EVENTS` pack.

For more information on parameters, see the *IDOL Speech Server Reference*.

Create the Alarm Database

For the `audiosec` module to identify a detected alarm, it must match the alarm against a template in its database. You must create a template for each alarm you want to identify.

> **NOTE:**
> If an alarm template does not exist for a particular alarm, IDOL Speech Server still detects the

> alarm, but labels it as `<UNKNOWN-ALARM>`.

A template consists of a short audio recording of the alarm. HPE recommends that the recording covers one whole period of the alarm, and that the sampling rate matches the sampling rate of the audio to be processed.

Save the templates in a single directory, and then create a list that contains the file names (excluding file extensions and paths) of the templates. For example:

```
alarm_2
alarm_3
alarm_4
alarm_7
alarm_9
alarm_11
alarm_12
```

For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.

Run Audio Security

Use the following procedures to run an `AudioSecurity` task and retrieve the results.

To run an AudioSecurity task

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task type. Set to **`AudioSecurity`**.

  `File` The audio file to process.
  To restrict processing to a section of the audio file, set the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

  `Out` The file to write the results to.

  To check against a database of alarm templates to identify an alarm, also set the following parameters:

  `TemplateList` The list that specifies the names of the alarm template files.

  `TemplatePath` The path to the directory that contains the alarm templates specified in the template list file.

For example:

```
http://localhost:13000/action=AddTask&Type=AudioSecurity&File=C:/speechserver/data/Sample.wav&TemplateList=ListManager/alarms&TemplatePath=C:/speechserver/alarmTemplates&Out=SampleSec.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to search the `Sample.wav` file for security-related sounds and check any detected alarms against the templates in the `alarms` list. The action instructs the server to write the results to the `SampleSec.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Find Recommended Language Resources to Use in a Task, on page 103.

You can also set some additional optional parameters for this action. For information about the parameters for the `AudioSecurity` task, see the *IDOL Speech Server Reference*.

IDOL Speech Server returns the results in the following format.

```
1    A    13.470    0.780    <gunshot>          15.515
1    A    19.760    0.860    <gunshot>          11.574
1    A    29.000    1.000    <UNKNOWN-ALARM>    0.125
1    A    30.000    1.000    <UNKNOWN-ALARM>    0.336
1    A    31.000    1.000    alarm_11           1.000
1    A    32.000    1.000    alarm_11           1.000
1    A    33.000    1.000    alarm_11           1.000
1    A    34.000    1.000    alarm_43           0.600
1    A    41.000    1.000    <UNKNOWN-ALARM>    0.256
1    A    42.000    1.000    <UNKNOWN-ALARM>    0.317
1    A    43.000    1.000    alarm_28           1.000
1    A    44.000    1.000    alarm_28           1.000
1    A    51.380    0.910    <gunshot>          11.107
1    A    60.000    1.000    <UNKNOWN-ALARM>    0.209
1    A    61.000    1.000    <UNKNOWN-ALARM>    0.261
1    A    62.000    1.000    alarm_25           1.000
1    A    63.000    1.000    alarm_25           1.000
1    A    64.000    1.000    alarm_11           1.000
1    A    65.000    0.000    <s>                1.000
```

From left to right, the columns in this file contain:

- The channel number, usually `1`.
- A fixed field, `A`.
- The time in seconds from the beginning of the audio to the start of the detected security-related event.
- The duration in seconds of the detected event.

> **NOTE:**
> For some security events, IDOL Speech Server represents the audio as a series of one-second periods and indicates the periods within which the event is present. In these cases, the results represent the beginning of each one-second period in which the event was present, rather than the precise start and end times of the events themselves.

- The name of the detected event; if IDOL Speech Server is using the alarm database, this column displays the name of the alarm template that has been matched.
- The confidence score.

Chapter 16: Stream Live Audio

This section describes how to stream live audio to IDOL Speech Server operations.

- Run Speech-to-Text on Live Audio 203
- Use Live Mode for Streaming 204

Run Speech-to-Text on Live Audio

IDOL Speech Server can analyze streamed audio data, in addition to audio files.

> **NOTE:**
> The exact task names and action parameters to use depend on the configuration in the IDOL Speech Server tasks configuration file.
>
> Streamed audio must conform to a required format. For more information, see Streamed Audio, on page 85.

To run speech-to-text on an audio stream

1. Send an `AddTask` action with the `Type` parameter set to **`SpeechToText`**, and the `InputType` parameter set to **`Stream`**. For example:

   ```
   http://localhost:13000/action=AddTask&Type=SpeechToText&Lang=ENUK&Out=Transcript1.ctm&InputType=Stream
   ```

   After this request, the server initializes the resources required for the task. When the server is ready to process the audio, the status of the task changes to `WAITING_CONNECTIONS`.

   > **CAUTION:**
   > Wait until the `WAITING_CONNECTIONS` status appears before you begin audio streaming.

2. Connect to the binary data port specified in the `BinaryDataPort` parameter in the `[Server]` section of the configuration file.

   The action ends if no connection is received within 30 minutes of the task becoming ready for connections, or if no data is received for more than 30 seconds following the connection.

   > **NOTE:**
   > To configure the length of time that IDOL Speech Server waits for data before it times out, set the `StreamReadWarning` and `StreamReadTimeout` parameters. For more information about these parameters, see the *IDOL Speech Server Reference*.

3. Send the audio stream using TCP. IDOL Speech Server uses the header to associate an audio stream with a particular task. All multiple byte integers must be sent in little-endian format.

The TCP stream header must have the following binary format.

Number of Bytes	Content
Variable	The token returned by the `AddTask` action (in ASCII). This must be `NULL` terminated.
4	The fixed integer constant `0x4D525453`.
4	The stream version number (currently 1).
4	The stream sampling rate in Hz (8000 or 16000).
4	The number of channels (1 or 2).
Remaining	Audio data samples as 2 byte, little-endian integers.

4. Speech-to-text finishes when the connection closes.

If you are using a lattice file and want to reduce the lattice output size by including only one sample of each word in a specific window size, you can also set the `LatWinSize` parameter. See Use a Lattice File, on page 123 and the *IDOL Speech Server Reference* for more information.

Remove Areas Categorized as Music or Noise

If you want to filter out any areas from the resulting `.CTM` file that are categorized as music, you can use the `SpeechToTextFilter` task instead of the `SpeechToText` task. This task combines the `SpeechToText` task with the `SpeechSILClassification` audio preprocessing task in a single step.

Related Topics

- Start and Stop Tasks, on page 85
- Check the Status of a Task, on page 87
- Run an Audio Preprocessing Task, on page 209

Use Live Mode for Streaming

When you perform speech-to-text conversion on a live audio stream, you can specify a mode that defines the rate to perform analysis. Versions of IDOL Speech Server from 10.8 upwards and the 6.0+ versions of the language packs use DNN acoustic models to improve speech-to-text accuracy. Each language pack contains at least two DNN acoustic models of different sizes. By default, in fixed mode the larger, most accurate model is used.

To override the default option, specify a different DNN file as the value of the `DNNFile` parameter in the task configuration file or at the command line.

CAUTION:
You can use DNN acoustic modelling in live or relative mode only if your DNN files are smaller than a certain size. In addition, you must be using Intel (or compatible) Processors that support SIMD extensions SSSE3 and SSE4.1. If this is not possible, you can set the `DNNFile` parameter to none to allow non-DNN speech-to-text without hardware limitations.

- If you know the rate in advance, use the `relative` mode, which uses a constant rate.
- If the rate varies, use the `live` mode. In `live` mode, recognition keeps pace with the rate at which data is sent to the server.
  - If data is sent to the server in real time, recognition is performed in real time.
  - If data is sent at twice real time, recognition is performed at twice real time, and so on.
  - The rate can change during processing and recognition continues at the correct rate.

> **NOTE:**
> If data streams too fast for the system's computational resources, recognition accuracy might be impaired.

> **TIP:**
> In live or relative mode, it is crucial to ensure that sufficient time is allowed for the recognition process to be robust and to produce transcriptions with good accuracy. To ensure this:
> - There is a limit to the minimum time allowed to perform live or relative recognition, based on the size of the smallest DNN available.
> - IDOL Speech Server monitors how deeply the search is progressing at each point, and sends the results to the speech-to-text diagnostic file at regular intervals. If this file indicates that a search has not had enough time, IDOL Speech Server issues a task warning. You can view this warning in the server log file, or retrieve it using the `GetStatus` action. IDOL Speech Server repeats the warning after each hour of audio processing.

To use `live` mode in live stream speech-to-text tasks, you must add the `Mode` configuration parameter to the configuration sections for the `stt` module, and add the `StreamMode` configuration parameter to the configuration sections for the `audio` module, if they are not already present. For example:

```
[stt]
Mode=$params.Mode

[audio]
StreamMode=$params.Mode
```

This configuration creates a `Mode` action parameter. To use live mode, set the `Mode` action parameter to **`live`** in a task action that uses the `stt` and `audio` modules, such as `SpeechToText`. For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&Lang=ENUK&Out=Transcript1.ctm&Mode=Live&InputType=Stream
```

Related Topics

- Schemas for Standard Tasks, on page 225

Chapter 17: Preprocess Audio

The audio preprocessing module allows you to categorize audio and analyze its quality before you use it in tasks. This module has options to perform clipping detection, Signal-to-Noise Ratio (SNR) calculation, and Dual-Tone Multi-Frequency (DTMF) dial tone identification.

- Audio Preprocessor Modes 207
- Configure Audio Preprocessing Tasks 208
- Run an Audio Preprocessing Task 209
- Run Audio Analysis 212

> **NOTE:**
> From the 11.3 release, IDOL Speech Server uses an implementation of audio preprocessing based on DNN technolgy, which means that you do not need to tailor thresholds to specific audio types. The new implementation uses normalized feature vector input rather than audio samples, which requires updates to the task schemas.
>
> For tasks that combine audio preprocessing with speech-to-text, you must you must include separate `frontend` and `normalizer` calls for both audio preprocessing and speech-to-text, because the form of the `frontend` feature vectors needed for the two tasks might be different. The standard IDOL Speech Server tasks configuration file(`speechserver-tasks.cfg`) includes several examples.
>
> For more information on working with the new algorithm, see the *IDOL Speech Server Reference*. All tasks in the `speechserver-tasks.cfg` file use the new algorithm, but the old algorithm is still supported for backwards compatibility, and you can use it in exactly the same way as before.

Audio Preprocessor Modes

The audio preprocessor is available as the `audiopreproc` module, which can run in different modes. These modes provide:

- Audio categorization
- Clipping detection and assessment
- SNR calculation
- DTMF dial tone identification

When you use this module in a task, you must use the appropriate module mode for the task that you want to perform. For more information about the modes, see the *IDOL Speech Server Reference*.

Audio Categorization

Audio categorization is the principal use of the `audiopreproc` module. In this mode, the module classifies regions of the audio as silence, speech, or non-speech/music. You can send these segments to the `wout` module to produce an output file that lists the times at which each segment occurs.

Audio Quality Assessment

The `audiopreproc` module can provide an assessment of the quality of the audio waveform, to give an indication of its suitability for audio indexing.

Clipping Detection

One of the audio quality assessment modes produces an assessment of the percentage of the audio that is clipped. Even a small amount of clipping is highly damaging to transcription accuracy. The numeric clipping assessment is accompanied by a qualitative description of the audio clipping quality (`none`, `insignificant`, `minor`, `moderate`, or `heavy`).

Signal-to-Noise Ratio Calculation

The module can analyze audio levels. The module estimates the Signal-to-Noise Ratio (SNR) for the telephone call as a whole, and estimates a file audio level for the waveform regions classified as speech. The module also indicates when two key speakers in a call have significantly different audio levels, which can reduce transcription accuracy.

When you provide a reasonable silence threshold value, the module can also estimate peak audio levels in the audio. For each gain level recognized, the module produces information on the percentage of audio corresponding to that gain level, and the actual energy level value. The module reports a summary of the maximum difference in decibels between speaker levels. For a good quality waveform where the two speakers speak at a similar gain level, this number can be zero (or at least very low).

DTMF Identification

The `audiopreproc` module can perform DTMF (Dual-Tone Multi-Frequency) dial tone detection and identification. In this mode, the module can accurately detect and identify tones corresponding to the numbers 0-9, the letters A-D, and the asterisk (*) and hash (#) keys.

To enable DTMF dial tone identification, set the `audiopreproc` mode to `T`. For an example schema incorporating DTMF dial tone identification, see Identify DTMF Dial Tones, on page 256. This schema produces a stream of DTMF labels (such as `<DTMF-1>`, `<DTMF-A>` or `<DTMF-#>`).

Configure Audio Preprocessing Tasks

When you configure a task that uses the `audiopreproc` module, you must use the appropriate mode for the task that you want to perform. The following modes are available:

`A`	Audio categorization.
`C`	Clipping detection and assessment.
`S`	Signal-to-noise ratio calculation.
`T`	DTMF dial tone identification.

You can use one or more of these modes in a single schema. To use multiple modes, type the letter for each mode in the schema, without spaces. For example:

```
1 = w1,d <- audiopreproc(CATS,a)
```

This example uses all four of the `audiopreproc` module modes to analyze audio.

For examples and more information on how to configure tasks, see Sample Task Schemas, on page 245.

Related Topics

- Categorize Audio, on page 255
- Assess Audio Quality, on page 255
- Identify DTMF Dial Tones, on page 256

Run an Audio Preprocessing Task

You can preprocess audio as a standalone task or as the initial step in other processing tasks. There are four standalone preprocessing tasks available out of the box, corresponding to the four `audiopreproc` modes.

`SpeechSilClassification`	Classifies segments of audio as speech, non-speech, or music.
`ClippingDetection`	Detects and assesses clipping in an audio file.
`SNRCalculation`	Estimates the SNR levels across an audio file.
`DialToneIdentification`	Detects and identifies dial tones.

To run a standalone audio preprocessing task

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  `Type` The task name.

  `File` The audio file to process.

  To restrict processing to a section of the audio file, set the `StartTime` and `EndTime` parameters. For more information, see the *IDOL Speech Server Reference*).

  `Out` The file to write the results to.

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechSilClassification&File=C:\Data\Conference.wav&Out=ConfClassification.ctm
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform the `SpeechSilClassification` task on the `Conference.wav` file and write the results to the `ConfClassification.ctm` file.

This action returns a token. You can use the token to:

- Check the task status. See Check the Status of a Task, on page 87.
- Retrieve the results file. See Get Task Results, on page 105.

Related Topics

- Configure Custom Tasks, on page 235
- Start and Stop Tasks, on page 85

Example Results

The `GetResults` action displays the audio preprocessing results as XML in your Web browser. You can also open the .ctm file from the configured IDOL Speech Server temporary directory (or another location if you specified a path in the `Out` parameter).

Audio Categorization Mode

The following is an example of results produced by the audio categorization mode.

```
1     A     22.800          1.120          Music/Noise              0.000
1     A     23.920          0.500          Speech                   0.000
1     A     24.420          0.580          Silence                  0.000
1     A     25.000          2.280          Speech                   0.000
1     A     27.280          1.570          Speech                   0.000
1     A     28.850          0.500          Silence                  0.000
1     A     29.350          2.000          Speech                   0.000
```

From left to right, the columns in this file contain:

- The channel number, usually `1`.
- A fixed field, `A`.
- The time in seconds from the beginning of the audio to the start of the segment.
- The duration in seconds of the segment.
- The segment classification–either `Silence`, `Music/Noise`, or `Speech`.

Clipping Detection Mode

The following is an example of results produced by the clipping detection mode.

```
<audiopreproc>
       <clipping>
              <assessment>none</assessment>
              <percent_frames>0</percent_frames>
       </clipping>
</audiopreproc>
```

IDOL Speech Server returns both a qualitative assessment of the clipping and the percentage of affected frames.

SNR Calculation Mode

The following is an example of results produced by the SNR calculation mode.

```
0.01    2.5    22
2.84    5.33   25
6.62    7.12   25
7.12    9.05   24
9.05    9.71   24
9.71    12.2   23
13.43   14.15  24
14.15   14.82  28
14.82   17.11  22
17.11   19.6   25
20.17   20.67  17
20.67   21.41  29
21.41   23.9   21
```

From left to right, the columns in this file contain:

- The time in seconds from the beginning of the audio to the start of a section of speech.
- The time in seconds from the beginning of the audio to the end of the section of speech.
- The SNR for the speech segment.

DTMF Mode

The following is an example of results produced by the DTMF mode.

```
1     A     0.000          0.020          none               0.000
1     A     0.020          0.300          <DTMF-0>           53.942
1     A     0.320          1.040          none               0.000
1     A     1.360          0.300          <DTMF-1>           54.591
1     A     1.660          1.040          none               0.000
1     A     2.700          0.320          <DTMF-2>           52.078
1     A     3.020          1.030          none               0.000
```

```
1     A     4.050          0.320          <DTMF-3>             52.085
1     A     4.370          1.040          none                 0.000
1     A     5.410          0.310          <DTMF-4>             53.31
```

From left to right, the columns in this file contain:

- The channel number, usually `1`.
- A fixed field, `A`.
- The time in seconds from the beginning of the audio to the boundary between the dial tone and other audio.
- The duration in seconds of the dial tone or the period of audio between dial tones.
- The segment classification–if IDOL Speech Server detects a dial tone, it identifies the tone.
- The score for the tone identification.

Run Audio Analysis

As an alternative to using the different individual audio preprocessing options, you can use the `audioAnalysis` task to combine a number of the key audio preprocessing tasks that are supported by the `audiopreproc` module in a single task. The `audioAnalysis` task includes the following functions:

- Audio classification, as provided in the `speechSilClassification` task.
- Overall and segmented signal-to-noise ratio (SNR), as provided in the `SNRCalculation` task.
- Clipping detection, as provided in the `clippingDetection` task.
- Audio quality summary, which is unique to the `audioAnalysis` task.

To run audio analysis

- Send an `AddTask` action to IDOL Speech Server, and set the following parameters:

  File The audio file to process.

  Out The XML file to write the audio analysis summary results to.

For example:

```
http://localhost:13000/action=AddTask&Type=AudioAnalysis&File=C:\data\Sample.wav&Out=SampleAnalysis.xml
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to perform audio analysis on the `Sample.wav` file and to write the results to the `SampleAnalysis.xml` file.

The `AudioAnalysis` task also produces an additional audio classification `.ctm` file. By default, this file has the same name as the task token. You can use the `GetResults` action with the `label` parameter set to **`class`** to retrieve this file.

For more information, see the *IDOL Speech Server Reference*.

Chapter 18: Postprocess Results

This section describes the postproc (postprocessing) module, which allows you to modify results from other audio processing tasks, such as speech-to-text, language identification, and so on.

- Postprocessing Operations 213
- Filter Vocabulary 213
- Recombine Word Fragments 214
- Configure Audio Postprocessing Tasks 215

Postprocessing Operations

The postproc module provides two postprocessing operations:

- Word filtering–removing inappropriate or unwanted words from results
- Recombining word fragments to produce complete words–for example, Hebrew speech-to-text produces word fragments that need to be recombined

These operations are available as modes within the postproc module. To include postprocessing operations in a task, include the postproc module in the task schema. You must specify the mode for the module to run in. You can also combine modes to perform both word filtering and word fragment recombination in a single operation. For more information about how to configure the postproc module, see the *IDOL Speech Server Reference.*

Related Topics

- Configure Audio Postprocessing Tasks, on page 215

Filter Vocabulary

If you enable the vocabulary filtering mode, the postproc module checks against a list of words that are barred from appearing in the results. The words list might include expletives, names, or any other vocabulary that you want to exclude from the final results. If these words exist in the results, IDOL Speech Server replaces them with a specified term. By default, this term is "<BLEEP>"; to change the term, set the BarredTerm parameter in the postproc module section of the tasks configuration file.

In addition, the text filtering process also supports specific replacement words. Instead of specifying a single replacement word to replace all instances of any word that appears in the barred words list, you can now map different words to different terms by adding a second word to each line in the list. For example:

```
ANE A&E
rubbish
dreadful poor
```

In the first line, ANE maps to the correct word A&E. The banned word in the second line maps to the default replacement word (as defined by the BarredTerm parameter), because there is no specific alternative mapping. In the third line, dreadful maps to the specified alternative mapping poor.

> **NOTE:**
> You cannot map a single word to multiple replacement words; each line in the list can have a maximum of two entries.

To create a vocabulary filter list

1. Create a list that contains the words to filter out. Each word or pair of words is a separate list entry.

   For more information about IDOL Speech Server's list manager, see Create and Manage Lists, on page 109.
2. In the `postproc` module section of the tasks configuration file, set the `BarredList` parameter to the list name. For example, `BarredList=ListManager/barred`.

Recombine Word Fragments

Certain speech-to-text language packs, such as Hebrew (`HBIL`), are based on vocabulary that has been broken down into its smallest parts to maintain a feasible vocabulary size. Using one of these language packs to perform speech-to-text on audio can produce results that contain word fragments that need to be joined back together in the final results. The `postproc` module can recombine these fragments into complete words. If adjacent words in the results file include hyphens as an indication of a word break, the module treats these as prefixes or suffixes and joins them to the stem of the word. For example, the `postproc` module could receive the following word sequence (shown in CTM format):

```
1     A     0.000          0.351          a                    0.513
1     A     0.351          0.194          pre-                 0.325
1     A     0.545          0.419          exist                0.457
1     A     0.964          0.140          -ing                 0.621
1     A     1.104          0.855          condition            0.369
```

The module would combine the prefix “pre-“, the stem “exist”, and the suffix “-ing”, as shown in the following example:

```
1     A     0.000          0.351          a                    0.513
1     A     0.351          0.753          preexisting          0.457
1     A     1.104          0.855          condition            0.369
```

Speech-to-text results can contain errors, potentially leading to word fragments that would combine to form invalid words. To avoid producing invalid words, you can supply the `postproc` module with a list of all valid words for a language (this list file is provided in the language pack). The module then combines word fragments only if they form words that are in the list. The other word fragments are left uncombined.

To specify valid words for a language

- In the `postproc` module section of the tasks configuration file, set the `RcmpValidList` parameter to the name of the .wds file supplied in the language pack.

If you do not specify a valid words list, the module combines word fragments wherever indicated by hyphens, without attempting to validate the resulting words.

Configure Audio Postprocessing Tasks

When you configure a task that uses the `postproc` module, you must use the appropriate mode for the task that you want to perform. The following modes are available:

`B`	Vocabulary filtering
`R`	Word fragment recombination

You can use one or more of these modes in a single schema. To use multiple modes, type the letter for each mode in the schema, without spaces. For example:

```
w2 ← postproc(BR,w1)
```

This example uses both of the `postproc` module modes to process a results file.

For examples and more information on how to configure tasks, see Sample Task Schemas, on page 245.

Part 3: Create Custom Tasks

This section describes how to create your own speech processing tasks.

- Understanding Tasks
- Configure Custom Tasks
- Sample Task Schemas

Chapter 19: Understanding Tasks

This section describes how tasks are set up in IDOL Speech Server and describes the schemas for some of the standard tasks that IDOL Speech Server provides.

- Overview 219
- Modules and Data Streams 219
- Check Module License Information 220
- Syntax for Module and Stream Variables 221
- Schemas for Standard Tasks 225

Overview

Out of the box, IDOL Speech Server provides several standard tasks that allow you to run individual processing operations on audio data–for example, speech-to-text or speaker identification.
IDOL Speech Server also allows you to run multiple operations in a single task. For example, you might want to perform both speaker identification and speech-to-text on a single audio file. You can create a single task that includes both speech-to-text and speaker identification instead of using two separate actions.

Modules and Data Streams

To build your own tasks, you need to understand the core concepts that IDOL Speech Server is designed around. IDOL Speech Server contains two main sets of components:

- *Processing modules* (also referred to as modules). `stt` and `speakerid` are examples of processing modules.
- *Data streams* (also referred to as streams). Audio sample buffers and recognition results are examples of data streams.

These are the fundamental operational units in IDOL Speech Server. They form the basis of all audio processing tasks, whether standard or user-defined.

The following general rules govern modules and data streams:

- Each module performs one type of function, such as speech-to-text.
- Each module can accept multiple data streams and produce multiple data streams.
- Each audio processing task requires individual instances of modules and streams.
- Every instance of a module must be referenced by a unique name. There is a strict notation for naming these instances; for more information, see Name a Module Instance, on page 222.
- Every instance of a module must have its own section in the IDOL Speech Server tasks configuration file, where you specify its inputs and outputs.
- Each output stream instance is created by a single module (two modules cannot create the same stream instance, and a stream must be created by a module). However, this stream can either feed

into one, several, or no subsequent modules. If a stream is not used by any module, IDOL Speech Server might generate a warning.

- You must assign a unique name to each instance of a stream. There is a strict notation on how these variables can be named; for more information, see Name a Data Stream Instance, on the next page.
- Connections between module instances and stream instances are expressed in 'functional programming' syntax; for more information, see Specify Module Inputs and Outputs, on page 225.
- Every instance of a module used in an active task counts toward the total count allowed by the license key. Only active modules are counted for licensing purposes.

Check Module License Information

To check how many concurrent instances of each module you currently use, send a `GetStatus` action. This action also displays information on any limits on the numbers of each module imposed by your license.

To retrieve module licensing information, send a `GetStatus` action with the `ShowModules` parameter set to `True`. For example:

```
http://localhost:13000/action=GetStatus&ShowModules=True
```

This action returns the `GetStatus` page of the server on the local machine that uses ACI port `13000`. The `GetStatus` page returns additional information about module use. For example:

```
<modules>
	<module>
		<name>stt</name>
		<usage>0</usage>
		<maxCount>5</maxCount>
	</module>
	<module>
		<name>frontend</name>
		<usage>2</usage>
		<maxCount>unlimited</maxCount>
	</module>
	<module>
		<name>langid</name>
		<usage>2</usage>
		<maxCount>5</maxCount>
	</module>
</modules>
```

Related Topics

- Customize Logging, on page 69
- GetStatus, on page 82

Syntax for Module and Stream Variables

The following sections specify the syntax to use for naming stream and module instances, and specifying module inputs and outputs.

Name a Data Stream Instance

The general syntax for naming a data stream instance is:

StreamX

where:

Stream	is the prefix for the type of data stream.
X	is the instance number. The instance number cannot begin with the digit `0`. The number is optional–an instance without an assigned number is still treated as unique and different from other instances that have numbers.

For example, `w1` represents the instance '1' of the word label stream. `f3` represents the instance '3' of the speech frame stream.

This table contains a complete list of the available data stream types.

Data stream type	Prefix
Audio sample–mono	`a`
Audio sample–left channel	`l`
Audio sample–right channel	`r`
Feature frame–unnormalized feature	`f`
Feature frame–normalized feature	`nf`
Word tag (for example, recognition result)	`w`
Language identification feature	`lf`
Language identification result	`lid`
Language identification boundary point information **DEPRECATED:** The `lb` data stream type is deprecated in IDOL Speech Server version 11.5 and later. Language identification boundary information is available in the `lid` output in boundary mode language identification.	`lb`
Speaker identification output results	`sid`
Audio fingerprint feature data	`fp`

Data stream type	Prefix
Audio fingerprint identification data (identified track records)	`tr`
Phone data for phrase matching	`p`
Score data for phrase matching	`s`
File label boundaries	`fl`
Time shift data	`ts`

Name a Module Instance

The general syntax for naming a module instance is similar to the syntax for naming stream instances. For each module instance, the syntax is:

ModuleTypeX

where:

ModuleType	is the module name.
X	is the instance number. The instance number cannot begin with the digit `0`. The number is optional–an instance without an assigned number is still treated as unique and different from other instances that have numbers.

For example, `stt1` represents the first instance of the `stt` (speech-to-text) module. `wout4` is the fourth instance of the `wout` module, which is used for handling word labels.

This table contains a complete list of available processing modules.

Module	Description
`afp`	Audio fingerprint identification module
`afpedit`	Audio fingerprint database administration module
`afpfeature`	Audio fingerprint feature extraction module
`afpfile`	Audio fingerprint feature file module
`afpout`	Audio fingerprint identification output module
`align`	Transcription alignment module
`amadaptadddata`	Acoustic model adaptation (data generation) module **DEPRECATED:** The `amadaptadddata` module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the newer Deep Neural Network (DNN) techniques, rather than GMM-based acoustic models.

Module	Description
	This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.
amadaptend	Acoustic model adaptation module **DEPRECATED:** The amadaptend module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the newer Deep Neural Network (DNN) techniques, rather than GMM-based acoustic models. This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.
audio	Audio file or stream input module
audiopreproc	Audio preprocessing module
audiosec	Audio security module
audiotemplatedevel	Audio template development score module
audiotemplateedit	Speaker set file module
audiotemplatescore	Audio template identification module
audiotemplatetrain	Speaker template training module
ctm	Word input module
filter	Stream filter module
fmdcombiner	Audio phrase search data file combiner module
frontend	Speech feature extraction module
langid	Language identification module
lbout	Language identification boundary output module **DEPRECATED:** The lbout module is deprecated in IDOL Speech Server version 11.5 and later. Boundary mode language identification provides the language boundary information in the main output file. This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.
lfout	Language identification feature output module

Module	Description
lidfeature	Language identification feature extraction module
lidoptimizer	Language identification classifier optimization module
lidout	Language identification output module
lidtrain	Language identification language training module
lmbuild	Custom language model build module
lmtool	Language model query module
mixer	Combines words from multiple channels into a single timeline
normalizer	Audio normalization module
phrasematch	Audio phrase match search module
phraseprematch	Audio phrase match preprocessing module
plh	Audio feature file processing module
postproc	Postprocessing module
segment	Speaker identification segmentation module
sidout	Speaker identification output module
speakerid	Speaker identification module
stream	Audio stream input module **DEPRECATED:** The stream module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the audio module instead. This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.
stt	Speech-to-text module
textnorm	Text normalization module
transcheck	Transcription checker module
wav	Audio file input module **DEPRECATED:** The wav module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the audio module instead. This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted

Module	Description
	in future.
wout	Word output module

Specify Module Inputs and Outputs

Every module instance must have a section in the IDOL Speech Server tasks configuration file, where you specify its inputs and outputs.

You must use the following syntax to specify the inputs and outputs of each module instance:

```
{outputStreams} <- module (mode,{inputStreams})
```

For example, the following configuration specifies that the `stt1` module instance accepts an input stream of normalized frames (`nf1`) and produces the word output instance `w3`:

```
w3 <- stt1 (_, nf1)
```

where _ specifies the default operational mode for the `stt1` module instance.

Schemas for Standard Tasks

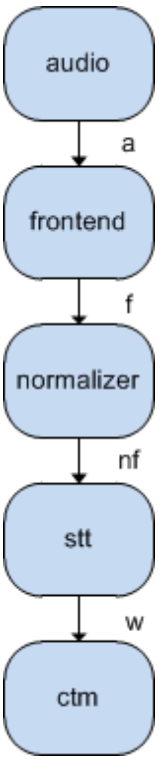
Before you configure your own tasks, it is useful to look at how the standard tasks are configured.

Speech-to-Text

The process flow for speech-to-text from the audio data point of view is:

1. The audio signal stream is broken into short overlapping windows.
2. The overlapping windows are analyzed to produce a vector time series of short-term feature vectors.
3. The feature vectors are then range-normalized to eliminate or reduce systematic variations.
4. The speech-to-text engine matches the feature vectors against the language pack contents to produce a series of word outputs.

IDOL Speech Server contains a separate module for each processing step. The following diagram shows how you can implement speech-to-text from the processing modules available in IDOL Speech Server.



The `audio` module reads the audio file and prepares windowed data.

a is the audio window series.

The `frontend` module takes each window of samples and converts it to a feature vector.

f is the feature vector series.

The `normalizer` module adjusts the feature vectors to produce normalized feature vectors.

nf is the normalized feature vector series.

The `stt` (speech-to-text) module converts the feature time series into a sequence of recognized words.

w is the output time-marked word series.

The `wout` module prepares the output words for storage and result reporting.

You must create this processing sequence in the IDOL Speech Server tasks configuration file to represent a single action. In this case, the sequence results in the following configuration section.

```
[MySpeechToText]
0 = a, ts ← audio (MONO, input)
1 = f ← frontend (_, a)
2 = nf ← normalizer (_, f)
3 = w ← stt(_, nf)
4 = output ← wout(_, w, ts)
```

The notation comes from the functional programming style.

- The arrows indicate the direction in which to read the code: in this instance, from right to left. The output is to the left of the arrow.
- In each line, the first two terms (for example, `0 =`) are redundant and indicate line numbering only.
- The next term is the name of the processing module, for example: `audio`, `frontend`, `normalizer`.
- The terms in parentheses are the mode and the input for the module, in the form `(MODE, INPUT(s))`. So `(_, w)` specifies the default mode,"_", and the input word stream, `w`.
- `a`, `f`, `nf`, and `w` all refer to instances of different types of data series.

So line `0` specifies that the `audio` module operates in `MONO` mode, receiving an input file and producing `a` as the output data stream (`a` represents mono audio data–see Name a Data Stream Instance, on page 221 for data stream types). Line `1` specifies that the `frontend` module operates in default mode, receiving `a` as the input data stream and producing `f` as the output data stream.

You must also configure each of the processing modules in the sequence. The following example configures the `audio` module.

```
[audio]
SampleFrequency = 16000
File = C:\Audio\Speech.wav
```

This example configures the `audio` module to:

- Receive data at 16 kHz (16000 Hz)
- Get the data from the audio file `Speech.wav`.

The following settings configure the `frontend` module to operate at 16 kHz.

```
[frontend]
SampleFrequency = 16000
```

The following settings configure the `normalizer` module to use a parameter file that is usually available with the language pack. For more information about the `IanFile` parameter, see the *IDOL Speech Server Reference*.

```
[normalizer]
IanFile=$Stt.Lang.NormFile
```

The following settings configure the `stt` module to use the UK English language pack, turn on run-time diagnostics, and set the running mode to `fixed`, with a mode value of `4`. You must also ensure the language pack section for `ENUK` is configured (see Language Configuration, below).

```
[stt]
Lang = ENUK
Diag = True
DiagFile = diag.log
Mode = fixed
ModeValue = 4
```

Finally, the `wout` module is configured to write the results in CTM format to the `output.ctm` file.

```
[wout]
Format = ctm
Output = output.ctm
```

Language Configuration

The language pack section needs to be set up once, or is set up by the installer. The entries in this section change only when a new language pack is installed.

The default configuration is:

```
[ENUK]
PackDir = U:\\lang\\ENUK
Pack = ENUK-5.0
SampleFrequency = 16000
DNNFile = $params.DNNFile
```

By default, IDOL Speech Server picks up the value of the `DNNFile` parameter from `Pack` and `PackDir` like other parameters. Alternatively, you can specify another `DNNFile` to use at the command line or in the task configuration file. For example, in fixed mode, you might want to use the `*-fast`. DNN file included in each language pack. This faster version is generally necessary for live or relative mode

where processing speed is critical. In this case, it is used automatically and does not need to be explicitly selected.

For information on how to configure the language pack section, see Configure Language Packs, on page 62.

> **NOTE:**
> HPE recommends (and for 7.0+ versions of language packs, it is compulsory) that you include the following lines in the configuration file for the `[frontend]` and `[normalizer]` modules, so that IDOL Speech Server can access the header to determine the quantity and nature of the extracted acoustic feature vectors:
>
> ```
> DNNFile = $stt.lang.DNNFile
> DNNFileStd = $stt.lang.DNNFileStd
> ```
>
> For more information, see the *IDOL Speech Server Reference*.

The complete configuration file section for the speech-to-text function is shown below. You must declare all schemas and language packs above this section in the tasks configuration file.

```
[TaskTypes]
0 = MySpeechToText

[Resources]
0 = ENUK

[MySpeechToText]
0 = a, ts ← audio (MONO, input)
1 = f ← frontend (_, a)
2 = nf ← normalizer (_, f)
3 = w ← stt(_, nf)
4 = output ← wout(_, w, ts)

[audio]
SampleFrequency = 16000
File = Speech.wav

[frontend]
SampleFrequency = 16000
DNNFile = $stt.lang.DNNFile
DNNFileStd = $stt.lang.DNNFileStd

[normalizer]
IanFile = $stt.Lang.NormFile
DNNFile = $stt.lang.DNNFile
DNNFileStd = $stt.lang.DNNFileStd

[stt]
Lang = ENUK
Diag = True
DiagFile = diag.log
Mode = fixed
ModeValue = 4
```

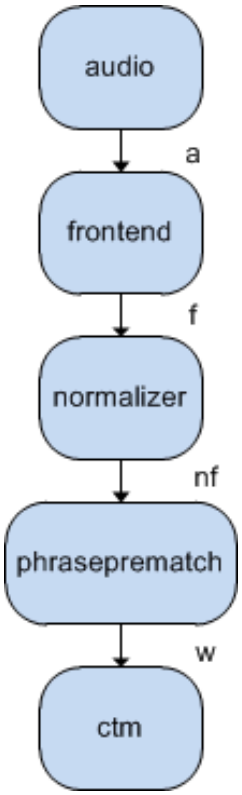
```
[wout]
Format = ctm
Output = output.ctm

[ENUK]
PackDir = U:\\lang\\ENUK
Pack = ENUK-5.0
SampleFrequency = 16000
```

The action command that runs this speech-to-text task is

```
http://SpeechServerhost:ACIport/action=AddTask&Type=MySpeechToText
```

Phonetic Phrase Search

The following diagram shows the modules in IDOL Speech Server that enable phonetic search in a single action.



The `audio` module reads the audio file and prepares windowed data.

a is the resulting audio window series.

The `frontend` module takes each window of samples and converts it to a feature vector.

f is the feature vector series.

The `normalizer` module adjusts the feature vectors to produce normalized feature vectors.

nf is the normalized feature vector series.

The `phraseprematch` module searches the feature vector time series and creates phrase identification positions, including confidence scores.

w is the output time-marked word series.

The `wout` module prepares the output phrase labels and time positions for storage and result reporting.

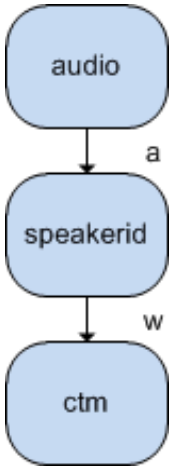
The schema that implements this feature is:

```
[MyPhraseSearch]
a, ts ← audio (MONO, input)
f ← frontend (_, a)
nf ← normalizer (_, f)
```

```
w ← phraseprematch (_, nf)
output ← wout (_, w, ts)
```

Speaker Identification

The following diagram shows the modules in IDOL Speech Server that enable speaker identification in a single step.



The `audio` module reads the audio file and prepares windowed data.

a is the audio window series.

The `speakerid` module takes each window of samples, analyzes it and identifies samples that match the speakers in the speaker database.

w is the output time-marked word series.

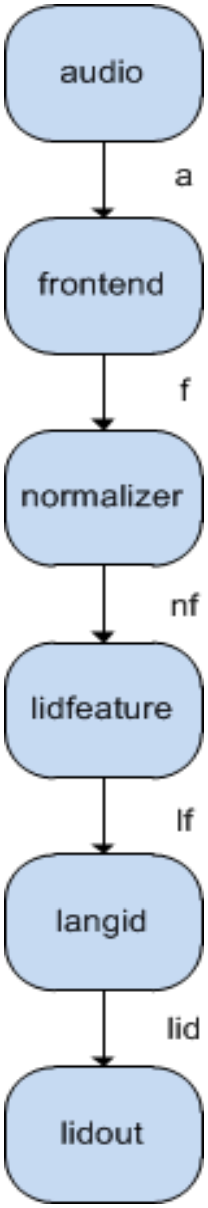
The `wout` module prepares the output phrase labels and time positions for storage and result reporting.

The schema that implements this feature is:

```
[MySpeakerId]
a, ts ← audio (MONO, input)
w ← speakerid (_, a)
output ← wout (_, w, ts)
```

Spoken Language Identification

The following diagram shows the modules in IDOL Speech Server that enable spoken language identification in a single step.



The `audio` module reads the audio file or stream and prepares windowed data.

a is the audio window series.

The `frontend` module takes each window of samples and converts it to a feature vector.

f is the feature vector series.

The `normalizer` module adjusts the feature vectors to produce normalized feature vectors.

nf is the normalized feature vector series.

The `lidfeature` module searches the feature vector time series to create phoneme-based language identification features.

lf is the language identification feature.

The `langid` module analyzes the features and determines the identified languages in the audio data.

lid is the output time-marked language identification data.

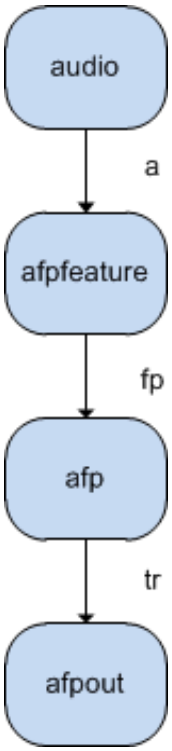
The `lidout` module prepares the output language labels and time positions for storage and result reporting.

The schema that implements this feature is:

```
[MyLangId]
a, ts ← audio (MONO, input)
f ← frontend (_, a)
nf ← normalizer (_, f)
lf ← lidfeature (_, nf)
lid ← langid (_, lf)
output ← lidout (_, lid, ts)
```

Audio Fingerprint Identification

The following diagram shows the modules in IDOL Speech Server that create audio fingerprints in a single step.



The `audio` module reads the audio file and prepares windowed data.

a is the audio window series.

The `afpfeature` module takes each window of samples and converts it to a time series of AFP features, based on significant audio events.

fp is the feature vector series.

The `afp` module matches the AFP features against tracks stored in the fingerprint database feature vectors.

tr is the recognized track labels and time positions.

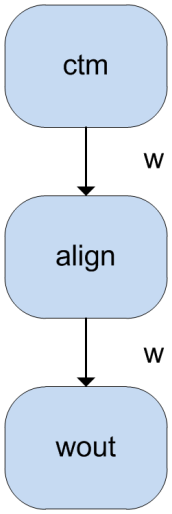
The `afpout` module prepares the output audio track labels and time positions for storage and result reporting.

The schema that implements this feature is:

```
[MyAfpMatch]
a,ts ← audio (MONO, input)
fp ← afpfeature (_, a)
tr ← afp (_, fp)
output ← afpout (_, tr, ts)
```

Transcript Alignment

The `align` module compares speech-to-text output with a normalized transcript file to produce a transcript that contains a timestamp for each word.



The `ctm` module reads the speech-to-text output file.

w is the time-marked word series.

The `align` module takes each word record and aligns it against the preloaded, normalized transcript and outputs time-coded transcript words.

w is the output time-marked word series.

The `wout` module prepares the output phrase labels and time positions for storage and result reporting.

The schema that implements this feature is:

```
[MyTranscriptAlign]
w ← ctm (READ, input)
w2 ← align(SCORE, input)
output ← wout(_, w2)
```

Chapter 20: Configure Custom Tasks

You can create your own custom tasks in the IDOL Speech Server tasks configuration file (`speechserver-tasks.cfg`). You must enter the task configurations into the configuration file before you start the server.

Setting up a new task involves creating a task schema:

- Selecting the modules that the task uses
- Determining the module inputs and outputs

For more information on available modules and instructions on schema syntax, see Syntax for Module and Stream Variables, on page 221. After you create the task schema, you can configure the settings for the task and the individual modules.

- General Task Configuration 235
- General Module Configuration 236
- Run the Configured Task 236
- Configure Variable Parameters 237
- Load a Language Pack Manually 240
- Unload a Language Pack 242

General Task Configuration

When you set up a task, there are some general properties that you can configure.

Convert Data Streams Between Permitted Types

You can readily convert streams that are of the same intrinsic data type from one type to the other. For example, you can convert between the three variants of the audio sample data type. The general syntax for in-place type conversion is *`NewType:OldType`*.

For example, if the input is a stereo audio file but you want to process the left channel only, set:

```
[MySpeechToTextOnLeft]
l, r ← audio (STEREO, input)
f ← frontend (_, a:l)
...
```

Disable Time Slicing

IDOL Speech Server uses a scheduler that uses time slicing to distribute the CPU cycles across all active tasks and modules. Time slicing saves power, but might lead to a slower running speed for some tasks. HPE recommends that you disable time slicing for modules that do not process audio data, such as `lmbuild` and `align`.

To disable time slicing for a task

- Add the following parameter and value to the task schema:

```
[MyTask]
...
DisableTimeSlicing = True
```

General Module Configuration

You can configure individual modules within a task schema. Each module has several parameters that you can set to modify its behavior. For configuration information about each module, see the *IDOL Speech Server Reference*.

For example, the `stt` module has the following parameters in the IDOL Speech Server tasks configuration file:

```
[stt]
Lang = $params.Lang=ENUK
DiagFile = $params.DiagFile=$outfile
Diag = $params.Diag=False
Mode = $params.Mode=fixed
ModeValue = $params.ModeValue = 4
DiagFileLabel = Diag
EnableConfidence = $params.Conf
DNNScale = $params.DNNScale
```

You can configure a default output for every module. For more information, see Configure Default Outputs, below.

Configure Default Outputs

You can optionally set a default output for each module. This is useful when you want to set a default output for the `GetResults` action for a task. For example, to use the `GetResults` action to return the speech-to-text outputs, you can configure the `wout` module so that its output is labeled as 'out'. The `GetResults` action returns the labeled file–in this case, `output.ctm`.

```
[wout1]
Output = output.ctm
...
OutputLabel = out
```

Run the Configured Task

To run configured tasks, send an `AddTask` action to IDOL Speech Server. You must set the `Type` action parameter to the name of the configuration section that defines the task. You must also set any parameters required for the task to run.

For example:

```
http://localhost:13000/action=AddTask&Type=SpeechToText&File=Speech.wav&Out=Text1.ctm&Lang=ENUS
```

IDOL Speech Server uses the task configuration to run all the required modules and to produce the required output files.

Related Topics

- Start and Stop Tasks, on page 85

Configure Variable Parameters

To improve the usability of tasks, you can change some of the configuration parameters into action parameters that you specify when you send an action. To do this, set the task configuration parameters to built-in variable values.

These variables can either:

- Create action parameters that allow you to specify the configuration parameter value when you run the task
- Refer similar configuration parameters to the one configuration parameter where a standard value is set

Variable parameters give you complete flexibility with modules. For example, if you have a single task that requires two `stt` modules (`stt` and `stt1`), you can ensure that each `stt` module uses different action parameters in the `AddTask` action.

Another example of how you might use variable values is if you want to create temporary files. You can use the `$token` built-in variable. Because this is the same as the unique token ID returned for each task, you can also use it as a file name associated with that task for temporary file purposes.

> **NOTE:**
> You must use UTF-8 to encode action parameters stored on disk, for example in the configuration file.

Generate Output File Names

You can use the `$outfile` built-in variable if you want IDOL Speech Server to automatically generate a unique file name for an output file, instead of specifying the file name yourself. For example, the default configuration for the `[wout]` module for the `SpeechToText` task is shown below:

```
[wout]
format = ctm
output = $params.out = $outfile
outputLabel = out
```

If you specify an output file name as part of the action (for example, `Out=test.ctm`), IDOL Speech Server uses that name. If you do not specify a file name in the action, IDOL Speech Server automatically generates a name based on a combination of the task token, the name of the module that generates the output file, and the name of the parameter used to specify the output file name. For example, in the case above, the file name might be `tmp_MTkyLjE2OC45NS4yNDox_wout_output`.

IDOL Speech Server writes the file to the standard directory for the output type (`TempDir`, `SpeakerIDDir`, and so on).

You can send a `GetStatus` action with the task token to determine the names of the output files generated by a task. The output files are listed in the response.

Configure Action Parameters

For task, module, and language configurations, you can define the configuration parameters as a variable of the form:

`ConfigParam``=$params.`*`ActionParam`*

where:

`ConfigParam`	is the name of the configuration parameter set by the value of the action parameter.
`ActionParam`	is the name of the action parameter.

For example, in the `audio` module, you can set the `File` configuration parameter to the name of an action parameter.

```
File=$params.File
```

If you configure a `WavRead` task that uses the `audio` module, you can then send an `AddTask` action to create a new `WavRead` task.

```
http://localhost:13000/action=AddTask&Type=WavRead&File=Speech.wav
```

The `File` parameter in this action sets the `File` configuration parameter in the `[audio]` module configuration section to `Speech.wav` for the duration of the task.

Action parameter names have the following restrictions.

- Action parameter names can contain only the characters `A–Z`, `a–z`, `0–9`, and underscore (_).
- Parameter names must not start with a numerical value (for example, `Param1` is valid, but `1Param` is not).

You cannot use the following reserved names as new action parameters:

- `Token`
- `TempDir`
- `CustomLmDir`
- `Type`

You can use these names in a configuration parameter (for example `$params.Token`), where they take on the following predefined values.

`Token`	The current task token.
`TempDir`	The temporary directory.
`CustomLmDir`	The custom language model directory.
`Type`	The type of task as specified in the `AddTask` action.

Set a Default Value

For some configuration parameters, it might be appropriate to set a default value.

ConfigParam=$params.*ActionParam=DefaultValue*

where:

ConfigParam	is the name of the configuration parameter set by the value of the action parameter.
ActionParam	is the name of the action parameter.
DefaultValue	is the default value that the parameter takes if the action parameter is not specified.

For example:

```
Lang=$params.Lang=ENUK
```

The `Lang` configuration parameter is set by the `Lang` action parameter. If no `Lang` action parameter is specified in the action, it uses `ENUK`.

Configure Simple References

If you want one configuration parameter to use the value of another configuration parameter, you can set a variable for the configuration parameter value.

ConfigParam=$*ConfigSection.AnotherParam*

where:

ConfigParam	is the name of the configuration parameter set by the value of the referenced configuration parameter.
ConfigSection	is the name of the configuration section where the referenced configuration parameter appears.
AnotherParam	is the name of the configuration parameter in the specified *ConfigSection* whose value the *ConfigParam* parameter takes. This value can either be a fixed value or an action parameter variable.

For example:

```
[audio]
SampleFrequency=$ENUK.SampleFrequency
```

In this example, the `SampleFrequency` configuration parameter in the `[audio]` module configuration section uses the value of `SampleFrequency` in the `[ENUK]` language configuration section.

Configure Complex References

You can set parameter values indirectly, as follows.

ConfigParam=$*ConfigSection.VarParam.AnotherParam*

where:

ConfigParam	is the name of the configuration parameter set to the value of the referenced configuration parameter.
ConfigSection	is the name of the configuration section where the referenced variable configuration parameter appears.
VarParam	is the name of the configuration parameter in the *ConfigSection* section, which references another configuration section.
AnotherParam	is the name of the configuration parameter in the configuration section specified by the value of *VarParam*, whose value the *ConfigParam* parameter takes.

For example, if an action parameter allows you to choose which language model to use for a particular action, all configuration parameters in the used modules must reference the correct language configuration.

```
[stt]
Lang=$params.Lang=ENUK
```

In this case, you can choose a language pack by using the `Lang` action parameter to specify the configuration section where the language options are set.

All configuration parameters that can change between language models must refer to the correct language configuration section. For example, you can set the `SampleFrequency` configuration parameter for modules as well as languages. To use the correct value, you can set `SampleFrequency` in all modules to refer to the `Lang` configuration parameter in a single module.

```
[audio]
SampleFrequency=$stt.Lang.SampleFrequency
```

In this example, the `SampleFrequency` parameter in the `audio` module refers to the `Lang` parameter in the `stt` module. In the `stt` section, the `Lang` configuration parameter uses the `Lang` action parameter to determine which language configuration section to use.

Load a Language Pack Manually

By default, IDOL Speech Server automatically loads and unloads language packs depending on the requirements of server actions. For example, if you submit a French audio file to IDOL Speech Server for speech-to-text processing with the appropriate parameters, IDOL Speech Server automatically loads the French language pack.

Holding language packs in memory requires a lot of RAM. It also takes time to load language packs into memory. As a result, IDOL Speech Server actively manages the loading and unloading of language packs. The `MaxLangResources` configuration parameter defines the maximum number of language packs that can be loaded in IDOL Speech Server simultaneously. For more information about this configuration parameter, see the *IDOL Speech Server Reference*.

You can override automatic loading and unloading of language packs and perform these tasks manually. Loading a language pack manually ensures that the model required for a task is ready for use, which reduces startup time for tasks that you add later. Manually loaded language packs remain loaded until you manually unload them.

> **NOTE:**
> The exact task names and action parameters to use depend on the configuration in the IDOL Speech Server tasks configuration file.

To manually load a language pack

- Send a `LoadLanguage` action with the `Name` parameter set to the name of the configuration file section where the language pack that you want to load is defined.

  If you want to add new words to the language model, or increase the weighting of specific words, you can specify a text file that contains the relevant words and weights as the value of the `ClassWordFile` parameter. You can set this parameter for any IDOL Speech Server task that uses language resources.

  For example:

  ```
  http://localhost:13000/action=LoadLanguage&Name=ENUS&ClassWordFile=D:/WordWeightings.txt
  ```

  This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to load the `ENUS` language to IDOL Speech Server, using the additional words and weightings specified in the `WordWeightings.txt` class word file.

  You can also add new pronunciations or edit existing pronunciations at language load time. To do this, specify a text file that contains the pronunciations that you want to add or edit as the value of the `PronFile` parameter. You can set this parameter for any IDOL Speech Server task that uses language resources.

  For example:

  ```
  http://localhost:13000/action=LoadLanguage&Name=ENUK-6.3&PackDir=ENUK&PronFile=T:\Pronunciations.txt
  ```

  This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to use `Pronunciations.txt` to update the pronunciations for the `ENUK-6.3` language pack in the `ENUK` directory.

  For more information on the format of the pronunciations file, see the *IDOL Speech Server Reference*.

> **NOTE:**
> If you load a language pack with a class word file or a pronunciations file, you cannot add new entries to the file by editing the file.To add new entries, you must create a new file in the same format and rerun the action. When you do this, IDOL Speech Server loads a new language pack. Note that this applies to automatically loaded language packs as well as manually loaded language packs. For more information, see the *IDOL Speech Server Reference*.

To manually load a custom language model

- Send a `LoadLanguage` action:
  1. Set the `Name` parameter to the name of the language pack section in the configuration file.
  2. Set the `DctFile` parameter to the name of the custom dictionary file.
  3. Specify the custom language files as the value of the `CustomLM` parameter.
  4. Specify a list of .tlm language files and weights separated by colons (:), in the form

file:*weight*.

For example, if you use two files, `autn1.tlm` and `autn2.tlm`, and you want to set the weights to `0.3` and `0.4` respectively, use:

```
CustomLM=autn1:0.3:autn2:0.4
```

For example:

```
http://localhost:13000/action=LoadLanguage&Name=ENUS&CustomLM=hpe1:0.3:hpe2:0.4&Dct
File=hpe
```

This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to load the `ENUS` base language model, the `hpe1.tlm` custom language model, and the `hpe2.tlm` custom language model. The models are weighted using the ratio `0.3`:`0.3`:`0.4`. IDOL Speech Server also loads the `hpe` dictionary file.

To determine the weight of the base language model, subtract the weights of the custom language models from 1.0. In this case, the base language model has a weight of `0.3`. All weights must be greater than `0` and less than `1.0`.

Related Topics

- Create Custom Language Models, on page 129

Unload a Language Pack

When you manually load a language pack, it remains in memory until you unload it, by using the `UnloadLanguage` action. You can specify language packs by name or ID.

> **NOTE:**
> IDOL Speech Server does not always unload the language immediately when you send an `UnloadLanguage` action. By default, the resource is marked to unload and is automatically unloaded at some point determined by the server.
>
> To ensure that IDOL Speech Server unloads the language immediately, you can set the `Force` parameter to **`True`**.

To unload a language pack by ID

- Send an `UnloadLanguage` action with the ID parameter set to the ID of the language that you want to unload.

  For example:

  ```
  http://localhost:13000/action=UnloadLanguage&ID=22b6832f37d05ec
  ```

  This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to unload the language pack that has the ID `22b6832f37d05ec`.

  > **TIP:**
  > You can find the resource ID in the response to the `GetStatus` action.

To unload a language pack by name

- Send an `UnloadLanguage` action with the `Name` parameter set to the name of the language that you want to unload.

  For example:

  ```
  http://localhost:13000/action=UnloadLanguage&Name=ENUS
  ```

  This action uses port `13000` to instruct IDOL Speech Server, which is located on the local machine, to unload the `ENUS` language pack.

  > **NOTE:**
  > If you unload the language by `Name`, you must also specify any additional parameters that you used when you loaded the language (either when you sent the `LoadLanguage` action, or when you used an `AddTask` action that loaded a language).
  >
  > For example, if you specify `CustomLm` and `DctName` in the `LoadLanguage` action, you might send the following `UnloadLanguage` action:
  >
  > ```
  > http://localhost:13000/action=UnloadLanguage&Name=ENUS&CustomLM=hpe1:0.3:hpe2:0.4&DctFile=hpe
  > ```

Chapter 21: Sample Task Schemas

This section contains example schemas that allow you to perform different sequences of operations in IDOL Speech Server.

- Stereo Speech-to-Text Conversion 245
- Audio Speech-to-Text Conversion 246
- Speech-to-Text with Word Filtering 247
- Language Identification 247
- Language Identification Feature Extraction 248
- Language Identification Optimization 249
- Transcript Alignment 249
- Build Language Models 250
- Cluster Speech 250
- Train Templates for Speaker Identification 251
- Perform Speaker Identification Using Templates 251
- Audio Fingerprint Identification 252
- Add a Track to an Audio Fingerprint Database 252
- Remove a Track from an Audio Fingerprint Database 253
- Audio Security 253
- Preprocess Audio Files for Phonetic Phrase Search 253
- Combine Phonetic Phrase Search Data Files 254
- Run a Phonetic Phrase Search 254
- Combine Output from Multiple Channels 254
- Categorize Audio 255
- Assess Audio Quality 255
- Identify DTMF Dial Tones 256
- Word Recompounding 256

Stereo Speech-to-Text Conversion

To perform speech-to-text conversion on stereo audio input data, each channel can be processed separately. For example:

```
[SpeechToText]
0 = l,r ← audio(STEREO, input)
1 = f1 ← frontend1(_, a:l)
2 = nf1 ← normalizer1(_, f1)
3 = w1 ← stt1(_, nf1)
4 = output ← wout1(_, w1)
```

```
5 = f2 ← frontend2(_, a:r)
6 = nf2 ← normalizer2(_, f2)
7 = w2 ← stt2(_, nf2)
8 = output ← wout2(_, w2)
```

0 The audio module processes the input stereo audio file as left and right audio data.

1 The frontend1 module converts left audio channel (l) into speech front-end frame data. In this step, the variable form a:l represents the change of name for the left channel audio data (type l) to audio data (type a).

2 The normalizer1 module normalizes the frame data from 1 (f1).

3 The stt1 module converts the normalized frame data from 2 (nf1) into text.

4 The wout1 module writes the recognized words resulting from 3 (w1) to the output file.

5 The frontend2 module converts right audio channel (r) into speech front-end frame data. In this step, the variable form a:r represents the change of name for the right channel audio data (type r) to audio data (type a).

6 The normalizer2 module normalizes frame data from 5 (f2).

7 The stt2 module converts the normalized frame data from 6 (nf2) into text.

8 The wout2 module writes the recognized words resulting from 7 (w2) to the output file.

Audio Speech-to-Text Conversion

IDOL Speech Server can perform real-time speech-to-text conversion on a file or audio stream. The following sample configuration is for such a task.

```
[SpeechToText]
0 = a,ts <- audio(MONO, input)
1 = f <- frontend(_, a)
2 = nf <- normalizer(_, f)
3 = w1 <- stt(_, nf)
4 = w2  <- postproc(_, w1)
5 = output <- wout(_, w2, ts)
DefaultResults=Out
```

0 The audio module processes stream audio data.

1 The frontend module converts audio data into speech front-end frame data.

2 The normalizer module normalizes frame data from 1 (f).

3 The stt module converts normalized frame data from 2 (nf) into text.

4 The postproc module runs any post processing tasks on the results from 3 (w1).

5 The wout module writes the recognized words resulting from 4 (w2) to the output file.

You can also use the [SpeechToTextFilter] schema to combine audio stream-to-text conversion with speech classification in a single step so that you can then remove sections classified as music or noise from the resulting output file.

Speech-to-Text with Word Filtering

The following schema describes a speech-to-text operation that is followed by a postprocessing operation that filters the results to replace any inappropriate words with a specified term (by default, this term is "<BLEEP>").

```
[SpeechToTextFilter]
0 = a,ts <- audio(MONO, input)
1 = f1 <- frontend1(_, a)
2 = nf1 <- normalizer1(_, f1)
3 = w1 <- audiopreproc1(A, a, nf1)
4 = f2 <- frontend2(_, a)
5 = nf2 <- normalizer2(_, f2)
6 = w2 <- stt2(_, nf2)
7 = w3  <- postproc2(_, w2)
8 = w4 <- mixer(_, wa:w1, wb:w3)
9 = output <- wout(_, w4, ts)
DefaultResults=out
```

0 The audio module processes the mono audio.

1 The frontend1 module converts the audio data from 1 into speech front-end frame data.

2 The normalizer1 module normalizes the frame data from 2.

3 The audiopreproc1 module in audio classification mode processes the audio (a) and normalized frame data (nf1).

4 The frontend2 module converts the audio data from 0 (a) into speech front-end frame data.

5 The normalizer2 module normalizes the frame data from 4.

6 The stt2 module converts the normalized frame data into text.

7 The postproc2 module can apply punctuation to the text produced by 6 (w2).

8 The mixer module combines word outputs from 3 (w1) and 7 (w3) into a single timeline.

9 The wout module writes the filtered words resulting from 8 to file.

Language Identification

The following schema describes language identification in CUMULATIVE mode.

```
[langId]
0 = a,ts <- audio(MONO, input)
1 = f1  <- frontend1(_, a)
```

```
2 = nf1 <- normalizer1(_, f1)
3 = w   <- audiopreproc(A, a, nf1)
4 = f2  <- frontend2(_, a)
5 = nf2 <- normalizer2(_, f2, w)
6 = lf1 <- lidfeature(_, nf2)
7 = lf2 <- filter(LF_INCLUSIVE, lf1, w)
8 = lid <- langid(_, lf2)
9 = output <- lidout(_, lid, ts)
DefaultResults=out
```

`0` The `audio` module processes the mono audio data.

`1` The `frontend1` module converts audio data into speech front-end frame data.

`2` The `normalizer1` module normalizes frame data from `1` (`f`).

`3` The `audiopreproc` module in audio classification mode processes the audio (`a`) and normalized frame data (`nf1`).

`4` The `frontend2` module converts the audio data from `0` into speech front-end frame data.

`5` The `normalizer2` module normalizes the frame data from `4`, adapted according to the audio classification from `3` (`w`).

`6` The `lidfeature` module converts normalized frame data from `5` (`nf2`) into language identification feature data.

`7` The `filter` module filters the output from `6` (`lf1`) to include only language features that occur in segments that contain speech.

`8` The `langid` module processes the language identification feature data from `7` (`lf2`) to identify the language.

`9` The language identification information (`lid`) is written to the output file.

Related Topics

- Identify Languages in Audio, on page 171

Language Identification Feature Extraction

The following schema describes the creation of language identification feature files that you can use to create and optimize language identification classifiers.

```
[langIdFeature]
0 = a <- audio(MONO, input)
1 = f1   <- frontend1(_, a)
2 = nf1  <- normalizer1(_, f1)
3 = w <- audiopreproc(A, a, nf1)
4 = f2 <- frontend2(_, a)
5 = nf2 <- normalizer2(_, f2, w)
6 = lf1 <- lidfeature(_, nf2)
```

```
7 = lf2 <- filter(LF_INCLUSIVE, lf1, w)
8 = output <- lfout(_, lf2)
DefaultResults=out
```

0 The `audio` module processes the mono audio data.

1 The `frontend1` module converts the audio data from `0` into front-end frame data.

2 The `normalizer1` module normalizes the frame data from `1`.

3 The `audiopreproc1` module runs audio classification on the audio (`a`) and the normalized frame data from `2` (`nf1`).

4 The `frontend2` module converts the audio data from `0` into front-end frame data.

5 The `normalizer2` module normalizes the frame data from `4`.

6 The `lidfeature` module converts the normalized frame data from `5` (`nf2`) into language identification feature data.

7 The `filter` module filters the output from `6` (`lf1`), using the audio classification data (`w`), to include only language features that occur in segments that contain speech.

8 The `lfout` module writes the language identification feature data from `7` (`lf2`) to the output file.

Language Identification Optimization

The following schema uses the `lidoptimizer` module to optimize language identification classifier files.

```
[LangIDOptimize]
0 = output ← lidoptimizer(_,input)
DisableTimeSlicing=True
```

0 The `lidoptimizer` module processes, optimizes, and outputs the input data.

Transcript Alignment

The following schema describes transcript alignment.

```
[transcriptAlign]
0 = w ← ctm(READ, input)
1 = w2 ← align(ALIGN, w)
2 = output ← wout(_, w2)
DefaultResults = out
```

0 The `ctm` module reads a CTM file that contains the transcript as produced by speech-to-text.

1 The `align` module aligns the text transcript by comparing it to the output from a previous speech-to-text operation.

2 The wout module writes the aligned transcript to the output file.

Build Language Models

The following schema describes building language models.

```
[LanguageModelBuild]
0 = output ← lmbuild(_, input)
DisableTimeSlicing=True
DefaultResults=log
```

0 The lmbuild module builds the language model.

Cluster Speech

The following schema describes how to segment an audio file into speaker clusters labeled Cluster_0, Cluster_1, and so on.

```
[ClusterSpeech]
0 = a <- audio(MONO, input)
1 = f1 <- frontend1(_, a)
2 = nf1 <- normalizer1(_, f1)
3 = w1 <- audiopreproc(A, a, nf1)
4 = f2 <- frontend2(_, a)
5 = nf2 <- normalizer2(_, f2, w1)
6 = w2 <- segment(_, nf2)
7 = w3 <- splitspeech(_, ws:w2, wc:w1, nf2)
8 = output <- wout(_, w3)
```

0 The audio module processes audio data.

1 The frontend1 module converts audio data (a) into speech front-end frame data.

2 The normalizer1 module normalizes frame data from 1 (f1).

3 The audiopreproc module processes the audio (a) and normalized frame data (nf) into Music, Speech, or Silence.

4 The frontend2 module converts the audio data from 0 (a) into speech front-end frame data.

5 The normalizer2 module normalizes frame data from 4 (f2).

6 The segment module finds short homogeneous acoustic segments.

7 The splitspeech modules forms the acoustic segments into speaker clusters.

8 The wout module writes the audio speaker clusters to a file.

Train Templates for Speaker Identification

The following schema describes how to train the speaker identification module with speakers.

```
[ivSpkIdTrainAudio]
0 = a <- audio(MONO, input)
1 = f1 <- frontend1(_,a)
2 = nf1 <- normalizer1(_, f1)
3 = w <- audiopreproc(A, a, nf1)
4 = f2 <- frontend2(_,a)
5 = nf2 <- normalizer2(BLOCK, f2)
6 = nf3 <- filter(FEAT_INCLUSIVE, nf2, w)
7 = output <- ivfile(CREATE_STREAM, nf3)
```

0 The `audio` module processes the mono audio data.

1 The `frontend1` module converts the audio data from `0` into front-end frame data.

2 The `normalizer1` module normalizes the frame data from `1`.

3 The `audiopreproc1` module in audio classification mode processes the audio (`a`) and normalized frame data (`nf1`).

4 The `frontend2` module converts the audio data (`a`) into front-end frame data.

5 The `normalizer2` module normalizes the frame data from `4`.

6 The `filter` module filters the output from `5` (`nf2`) to include only frames that occur in segments that contain speech.

7 The `ivfile` module trains the iVector feature files.

Perform Speaker Identification Using Templates

The following schema describes how to run speaker identification given a set of templates.

```
[ivSpkId]
0 = a,ts <- audio(MONO, input)
1 = w1 <- speakerid(GENDER_DETECT, a)
2 = f1 <- frontend1(_,a)
3 = nf1 <- normalizer1(_, f1)
4 = w2 <- audiopreproc(A, a, nf1)
5 = f2 <- frontend2(_,a)
6 = nf2 <- normalizer2(SEGMENT_BLOCK, f2, w1)
7 = nf3 <- filter(FEAT_INCLUSIVE, nf2, w2)
8 = sid <- ivScore(SEGMENT, nf3, w1)
9 = output <- sidout(_, sid, ts)
```

0 The `audio` module processes the mono audio data.

1 The speakerid module takes the audio data (a) and outputs speaker turn segments.

2 The frontend1 module converts the audio data from 1 into front-end frame data.

3 The normalizer1 module normalizes the frame data from 2.

4 The audiopreproc1 module in audio classification mode processes the audio (a) and normalized frame data (nf1).

5 The frontend2 module converts the audio data from 0 into front-end frame data.

6 The normalizer2 module normalizes the frame data from 5.

7 The filter module filters the output from 6 (nf2) to include only frames that occur in segments that contain speech, by using the audio classification data from 4 (w2).

4 The ivscore module takes audio features from 7 (nf3) and speaker segment information (w1), and produces a set of iVector speaker scores for each segment.

9 The sidout module takes the speaker ID score information (sid) and writes this information into a results file.

Audio Fingerprint Identification

The following schema describes how to define an audio fingerprint (AFP) identification task.

```
[afpMatch]
0 = a,ts <- audio(MONO, input)
1 = fp <- afpfeature(_, a)
2 = tr <- afp(_, fp)
3 = output <- afpout(_, tr, ts)
DefaultResults=out
```

0 The audio module processes the mono audio.

1 The afpfeature module converts audio data into audio fingerprint features.

2 The afp module matches features against a database of known tracks.

3 The afpout module writes the identification results to file.

Add a Track to an Audio Fingerprint Database

The following schema describes how to add an audio track to an AFP database.

```
[afpAddTrack]
0 = a <- audio(MONO, input)
1 = fp <- afpfeature(_, a)
2 = output <- afpedit(ADD,fp)
```

0 The audio module processes the mono audio.

1 The `afpfeature` module converts the audio data into AFP features.

2 The `afpedit` module adds the features to an AFP database.

Remove a Track from an Audio Fingerprint Database

The following schema describes the task of removing an audio task from an AFP database.

```
[AfpRemoveTrack]
0 = output ← afpedit(FINGERPRINT_REMOVE, input)
```

0 The `afpedit` module removes the specified audio track from the database.

Audio Security

The following schema describes how to use the `audiosec` module for audio security processing.

```
[AudioSecurity]
0 = a,ts <- audio(MONO, input)
1 = f <- frontend(_, a)
2 = nf <- normalizer(_, f)
3 = w <- audiosec(_, a, nf)
4 = output <- wout(_, w,ts)
```

0 The `audio` module processes mono audio.

1 The `frontend` module converts audio data into speech front-end frame data.

2 The `normalizer` module normalizes frame data from `1` (`f`).

3 The `audiosec` module processes the audio (`a`) and normalized frame data (`nf`).

4 The `wout` module writes the security issue label information to the output file.

Preprocess Audio Files for Phonetic Phrase Search

The following schema describes the preprocessing of an audio file to create a phoneme time track file required for phonetic phrase search.

```
[createFmd]
0 = a, ts <- audio(MONO, input)
1 = f <- frontend(_, a)
2 = nf <- normalizer(_, f)
3 = output <- phraseprematch(WRITE, nf, ts)
```

0 The `audio` module processes the mono audio.

1 The `frontend` module converts the audio data into speech front-end frame data.

2 The `normalizer` module normalizes the frame data from `1` (`f`).

3 The `phraseprematch` module processes the normalized frame data from `2` (`nf`) to generate phone-track information, which is written to the .fmd file for phonetic phrase search.

Combine Phonetic Phrase Search Data Files

The following schema describes a task that combines multiple phoneme time track files (each produced from a single audio file) into a combined file.

```
[CombineFMD]
0 = output ← fmdcombiner(_, input)
```

0 The `fmdcombiner` module combines individual data files listed in a text file into a single .fmd file.

Run a Phonetic Phrase Search

The following schema describes the process for phonetic phrase searching.

```
[SearchFMD]
0 = w, fl, s, p, nf <- phraseprematch(READ, input)
1 = w2 <- phrasematch(_, w, p, s, nf)
2 = output <- wout(_, w2, fl)
DefaultResults = out
```

0 The `phraseprematch` module reads the preprocessed data file.

1 The `phrasematch` module searches the audio phrase data from `1` (`w`, `p`, `s`, `nf`).

2 The `wout` module writes the recognized phrase locations resulting from `1` (`w2`) and the file labels from `0` (`fl`) to the output file.

Combine Output from Multiple Channels

The following schema describes how to use the `mixer` module to combine word output from multiple modules into a single timeline.

```
[StereoSpeechToText2]
0 = l,r,ts ← audio(STEREO, Input)
1 = f1 ← frontend1(_, a:l)
2 = nf1 ← normalizer1(_, f1)
3 = w1 ← stt1(_, nf1)
4 = f2 ← frontend2(_, a:r)
5 = nf2 ← normalizer2(_, f2)
6 = w2 ← stt2(_, nf2)
7 = w3 ← mixer(_, wa:w1, wb:w2)
8 = output ← wout(_, w3, ts)
```

0 The audio module processes the input stereo audio file as left and right audio data.

1 The frontend1 module converts the left audio channel (l) into speech front-end frame data. In this step, the variable form a:l represents the change of name for the left channel audio data (type l) to audio data (type a).

2 The normalizer1 module normalizes the frame data from 1 (f1).

3 The stt1 module converts the normalized frame data from 2 (nf1) into text.

4 The frontend2 module converts the right audio channel (r) into speech front-end frame data. In this step, the variable form a:r represents the change of name for the right channel audio data (type r) to audio data (type a).

5 The normalizer2 module normalizes the frame data from 5 (f2).

6 The stt2 module converts the normalized frame data from 6 (nf2) into text.

7 The mixer module combines the recognized words resulting from 3 (w1) and from 6 (w2) into a single word output timeline (w3).

8 The wout module writes the recognized words resulting from 7 (w3) to the output file.

Categorize Audio

The following schema describes how to use the audiopreproc module to identify regions of silence, speech, and non-speech in an audio file.

```
[AudioNet]
0 = a, ts ← audio(MONO, input)
1 = f1 <- frontend(_, a)
2 = nf1 <- normalizer(_, f1)
3 = w <- audiopreproc(A, a, nf1)
2 = output ← wout (_, w, ts)
```

0 The audio module processes the mono audio.

1 The frontend module converts the audio data from 0 into front-end frame data.

2 The normalizer module normalizes the frame data from 1.

3 The audiopreproc module in audio classification mode (A) processes the audio (a) and normalized frame data (nf).

4 The wout module writes the audio classification information (w) to the output file.

Assess Audio Quality

The following schema describes how to use the audiopreproc module to assess audio for clipping and to measure the signal-to-noise ratio.

```
[AudioAnalysis]
0 = a,ts <- audio(MONO, input)
1 = f <- frontend(_, a)
2 = nf <- normalizer(_, f)
3 = w <- audiopreproc(ACS, a, nf)
4 = output <- wout(_,w,ts)
```

0 The audio module processes the mono audio.

1 The frontend module converts the audio data from 0 into front-end frame data.

2 The normalizer module normalizes the frame data from 1.

3 The audiopreproc module, set to use audio classification (A), clipping detection (C) and signal-to-noise ratio calculation (S), processes the audio (a) and normalized frame data (nf).

4 The wout module writes the results to an output file.

Identify DTMF Dial Tones

The following schema describes how to use the audiopreproc module to identify DTMF dial tones.

```
[DtmfAudioNet]
0 = a, ts ← audio(MONO, input)
1 = f <- frontend(_, a)
2 = nf <- normalizer(_, f)
3 = d ← audiopreproc(T, a, nf)
4 = output ← wout (_, w:d, ts)
```

0 The audio module processes the mono audio.

1 The frontend module converts the audio data from 0 into front-end frame data.

2 The normalizer module normalizes the frame data from 1.

3 The audiopreproc module in DTMF identification mode (T) processes the audio (a) and normalized frame data (nf).

4 The wout module writes the DTMF label information (d) to the output file.

Word Recompounding

Some languages use word compounding, which means that typical words are too long for optimal speech recognition. Hungarian, Turkish. and Hebrew are typical examples. In these cases, you must break words down into constituent word fragments (or morphemes) for best performance.

The following example schema takes a CTM file that contains word fragments, and recompounds them into words.

```
[Recompound]
0 = w1 <- ctm(READ, input)
```

```
1 = w2 <- postproc(R, w1)
2 = output <- wout(_,w2)

[ctm]
file = $params.in
format = ctm

[postproc]
rcmpAllowSuffix = true
rcmpValidList = $params.validList

[wout]
file = $params.out
format = ctm
```

0 The `ctm` module reads the word fragment labels (with prefix and suffix hyphens indicating fragments).

1 The `postproc` module (mode `R`) adds prefixes to the following word and suffixes to the preceding word.

2 Output to a new CTM file.

Part 4: Appendixes

- Input and Output Files
- IDOL Speech Server File Types
- Audio Quality Guidelines
- Audio Transcript Requirements
- Compatibility Notes

Appendix A: Input and Output Files

This appendix contains tables that describe the files that IDOL Speech Server handles. Where appropriate, they include the following information.

Section	The configuration section that contains the parameters.
Configuration parameter	The configuration parameter that defines a file name.
Mode	The module mode.
Input/Output type	Whether the file is an input or an output file.
File type	The type of file, for example a log file or configuration file. This field specifies the file extension. For more information, see IDOL Speech Server File Types, on page 275.
Default directory	The default directory location for the file. If you do not specify a file path, IDOL Speech Server defaults to this directory.

Specify Files

When you specify a file in a parameter, you must include the file name. The file path is optional. If you do not include the path, IDOL Speech Server uses a default directory. You can specify either an absolute or relative file path; relative paths are regarded as being relative to the default directory. Use forward slashes (/) for paths on Linux and backslashes (\) for paths on Windows.

Input Files

If an input file name has a file extension, you must include it when you specify the file. The only exception is when you use the `CustomLm` parameter in the language resource section of the configuration files to specify a custom language model. In this case, you can supply up to nine custom language models, separated by one of the following separator characters: `:;@^|`

For example:

```
&customLM=testOK^0.08^testOK^0.08
&customLM=test OK|0.08|test OK|0.08
&customLM=test:OK;0.08;test:OK;0.08
```

> **NOTE:**
> you must use the same separator character within a single `customLM` declaration.

IDOL Speech Server checks the number of language models that you specify, and reports an error if there are more than nine.

Output Files

Output files have extensions added automatically if the AddOutputExtensions parameter is set to True in the [Server] section of the configuration file. Otherwise you must include the file extensions for output files.

Configuration File

The following sections describe the files specified in the speechserver.cfg file.

Logging Sections

This table lists files specified by parameters in logging configuration sections.

Section	Configuration parameter	Input/output type	File type	Default directory
ActionLogStream	LogFile	Output	LOG	LogDirectory
ApplicationLogStream	LogFile	Output	LOG	LogDirectory
SoftsoundLogStream	LogFile	Output	LOG	LogDirectory

[Server] Section

This table lists files specified by parameters in the [Server] configuration section.

Configuration parameter	Input/output type	File type	Default directory
TasksConfig	Input	CFG	Server

Tasks Configuration File

The following sections describe files specified in the tasks configuration file (speechserver-tasks.cfg).

[*LangPack*] Section

This table lists files specified by parameters in the [*LangPack*] configuration section.

Configuration parameter	Input/output type	File type	Default directory
CustomDct	Input	DCT	CustomLmDir
CustomLm	Input	TLM	CustomLmDir

Configuration parameter	Input/output type	File type	Default directory
AmFile	Input	AM	TrainedAmDir

[afpedit] Section

This table lists files specified by parameters in the [afpedit] configuration section.

Configuration parameter	Mode	Input/output type	File type	Default directory
Output	FINGERPRINT_INFO	Output	XML	TempDir

[afpfile] Section

This table lists files specified by parameters in the [afpfile] configuration section.

Configuration parameter	Mode	Input/output type	File type	Default directory
AfpFile	FINGERPRINT_READ	Input	AFP	TempDir
AfpFile	FINGERPRINT_WRITE	Output	AFP	TempDir

[afpout] Section

This table lists files specified by parameters in the [afpout] configuration section.

Configuration parameter	Input/output type	File type	Default directory
Output	Output	CTM	TempDir

[align] Section

This table lists files specified by parameters in the [align] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DctFile	Input	DCT	Server
PgfFile	Input	PGF	Server
ScoreOutFile	Output	TXT	TempDir
TxtFile	Input	TXT	DataDir
WarningFile	Output	LOG	TempDir

[amadaptadddata] Section

> **DEPRECATED:**
> The `amadaptadddata` module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the newer Deep Neural Network (DNN) techniques, rather than GMM-based acoustic models.
>
> This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.

This table lists files specified by parameters in the `[amadaptadddata]` configuration section.

Configuration parameter	Input/output type	File type	Default directory
`AccFile`	Output	`ACC`	`TempDir`
`AmFile`	Input	`AM`	`TrainedAmDir`
`DataListFile`	Input	`LIST`	`Server`
`DctFile`	Input	`DCT`	`Server`
`DiagFile`	Output	`TXT`	`TempDir`
`PgfFile`	Input	`PGF`	`Server`

[amadaptend] Section

> **DEPRECATED:**
> The `amadaptend` module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the newer Deep Neural Network (DNN) techniques, rather than GMM-based acoustic models.
>
> This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.

This table lists files specified by parameters in the `[amadaptend]` configuration section.

Configuration parameter	Input/output type	File type	Default directory
`AccListFile`	Input	`LIST`	`Server`
`InAmFile`	Input	`AM`	`TrainedAmDir`
`OutAmFile`	Output	`AM`	`TrainedAmDir`

[audio] Section

This table lists files specified by parameters in the `[audio]` configuration section.

Configuration parameter	Input/output type	File type	Default directory
File	Input	WAV	DataDir

[audiotemplatedevel] Section

This table lists files specified by parameters in the [audiotemplatedevel] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DataFile	Input	ATV	DataDir
DataListFile	Input	LIST	Server
DevFile	Input / Output	ATD	SpeakerIdDir
DevListFile	Input	LIST	Server
DiagFile	Output	DIAG	TempDir
LabFile	Input	LAB	DataDir
OutTemplate	Output	ATF	SpeakerIdDir
TemplateFile	Input	ATF	SpeakerIdDir
TemplateListFile	Input	LIST	Server
UbmFile	Input	ATF	SpeakerIdDir

[audiotemplateedit] Section

This table lists files specified by parameters in the [audiotemplateedit] configuration section.

Configuration parameter	Input/output type	File type	Default directory
OutputLog	Output	LOG	TempDir
OutTemplate	Output	ATF	SpeakerIdDir
OutTemplateSet	Output	ATS	SpeakerIdDir
TemplateFile	Input	ATF	SpeakerIdDir
TemplateListFile	Input	LIST	Server
TemplateSet	Input/Output	ATS	SpeakerIdDIr
UbmFile	Input	ATF	SpeakerIdDir

[audiotemplatescore] Section

This table lists files specified by parameters in the [audiotemplatescore] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DiagFile	Output	DIAG	TempDir
TemplateListFile	Input	LIST	Server
TemplateSet	Input	ATS	SpeakerIdDIr
UbmFile	Input	ATF	SpeakerIdDir

[audiotemplatetrain] Section

This table lists files specified by parameters in the [audiotemplatetrain] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DataFile	Input	ATV	DataDir
DataListFile	Input	LIST	Server
DiagFile	Output	DIAG	TempDir
LabFile	Input	LAB	DataDir
OrgTemplate	Input	ATF	SpeakerIdDir
OutTemplate	Output	ATF	SpeakerIdDir
UbmFile	Input	ATF	SpeakerIdDir

[audiopreproc] Section

This table lists files specified by parameters in the [audiopreproc] configuration section.

Configuration parameter	Input/output type	File type	Default directory
OutputLog	Output	TXT	TempDir
SNRFile	Output	TXT	TempDir

[audiosec] Section

This table lists files specified by parameters in the [audiosec] configuration section.

Configuration parameter	Input/output type	File type	Default directory
Al_TemplateListFile	Input	LIST	Server

[ctm] Section

This table lists files specified by parameters in the [ctm] configuration section.

Configuration parameter	Mode	Input/output type	File type	Default directory
File	READ	Input	CTM	TempDir

[filter] Section

This table lists files specified by parameters in the [filter] configuration section.

Configuration parameter	Input/output type	File type	Default directory
Filter	Input	TXT	Server

[fmdcombiner] Section

This table lists files specified by parameters in the [fmdcombiner] configuration section.

Configuration parameter	Input/output type	File type	Default directory
FmdFileOut	Output	FMD	TempDir
FmdListFile	Input	LIST	Server

[ivdevel] Section

This table lists files specified by parameters in the [ivdevel] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DataFile	Input	IVP	DataDir
DataListFile	Input	LIST	Server
DevFile	Input/Output	IVD	SpeakerIdDir
DevListFile	Input	LIST	SpeakerIdDir
DiagFile	Output	TXT	TempDir
LabFile	Input	SID	TempDir

Configuration parameter	Input/output type	File type	Default directory
OutTemplate	Output	IV	SpeakerIdDir
TemplateFile	Input	IV	SpeakerIdDir
TemplateListFile	Input	LIST	SpeakerIdDir

[ivedit] Section

This table lists files specified by parameters in the [ivedit] configuration section.

Configuration parameter	Input/output type	File type	Default directory
OutputLog	Output	TXT	TempDir
TemplateFile	Input/Output	IV	SpeakerIdDir
TemplateListFile	Input	LIST	SpeakerIdDir
TemplateSet	Input/Output	IVS	SpeakerIdDir

[ivfile] Section

This table lists files specified by parameters in the [ivfile] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DataFile	Input	IVP	DataDir
DataListFile	Input	LIST	Server
DiagFile	Output	TXT	TempDir
LabFile	Input	SID	TempDir
OutputIv	Output	IV	SpeakerIdDir

[ivscore] Section

This table lists files specified by parameters in the [ivscore] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DiagFile	Output	TXT	TempDir
TemplateListFile	Input	LIST	SpeakerIdDir
TemplateSet	Input	IVS	SpeakerIdDir

[langid] Section

This table lists files specified by parameters in the [langid] configuration section.

Configuration parameter	Input/output type	File type	Default directory
ClList	Input	LIST	Server

[lbout] Section

> **DEPRECATED:**
> The lbout module is deprecated in IDOL Speech Server version 11.5 and later. Boundary mode language identification provides the language boundary information in the main output file.
>
> This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.

This table lists files specified by parameters in the [lbout] configuration section.

Configuration parameter	Input/output type	File type	Default directory
Output	Output	CTM	TempDir

[lfout] Section

This table lists files specified by parameters in the [lfout] configuration section.

Configuration parameter	Mode	Input/output type	File type	Default directory
File	READ	Input	LIF	TempDir
Output	WRITE	Output	LIF	TempDir

[lidoptimizer] Section

This table lists files specified by parameters in the [lidoptimizer] configuration section.

Configuration parameter	Input/output type	File type	Default directory
ClList	Input	LIST	Server
DataListFile	Input	LIST	Server
Output	Output	TXT	TempDir
OutputLog	Output	XML	TempDir

[lidout] Section

This table lists files specified by parameters in the [lidout] configuration section.

Configuration parameter	Input/output type	File type	Default directory
Output	Output	CTM	TempDir

[lidtrain] Section

This table lists files specified by parameters in the [lidtrain] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DataListFile	Input	LIST	Server
NewClassifier	Output	LCF	TempDir
OutputLog	Output	LOG	TempDir

[lmbuild] Section

This table lists files specified by parameters in the [lmbuild] configuration section.

Configuration parameter	Input/output type	File type	Default directory
BaseDictionary	Input	DCT	Server
BaseLanguageModel	Input	TLM	Server
DataListFile	Input	TXT	Server
DropList	Input	TXT	Server
KeepList	Input	TXT	Server
NewDictionary	Output	DCT	CustomLmDir
NewLanguageModel	Output	TLM	CustomLmDir
OutputLog	Output	LOG	TempDir
PgfFile	Input	PGF	Server

[lmtool] Section

This table lists files specified by parameters in the [lmtool] configuration section.

Configuration parameter	Input/output type	File type	Default directory
BaseLanguageModel	Input	TLM	Server
CustomLanguageModel	Input	TLM	CustomLmDir
OutputLog	Output	TXT	TempDir
TextFile	Input	TXT	DataDir

[normalizer] Section

This table lists files specified by parameters in the [normalizer] configuration section.

Configuration parameter	Input/output type	File type	Default directory
IanFile	Input	IAN	Server

[phraseprematch] Section

This table lists files specified by parameters in the [phraseprematch] configuration section.

Configuration parameter	Mode	Input/output type	File type	Default directory
File	READ	Input	TXT	TempDir
File	WRITE	Output	TXT	TempDir
PhraseList	Not applicable	Input	TXT	Server

[plh] Section

This table lists files specified by parameters in the [plh] configuration section.

Configuration parameter	Mode	Input/output type	File type	Default directory
PlhFile	READ	Input	PLH, IVP	DataDir
PlhFile	WRITE	Output	PLH, IVP	DataDir

[postproc] Section

This table lists files specified by parameters in the [postproc] configuration section.

Configuration parameter	Input/output type	File type	Default directory
BarredList	Input	TXT	Server
RcmpValidList	Input	TXT	Server

[segment] Section

This table lists files specified by parameters in the [segment] configuration section.

Configuration parameter	Input/output type	File type	Default directory
IanFile	Input	IAN	Server
OutFile	Output	CTM	TempDir
WavFile	Input	WAV	DataDir

[sidout] Section

This table lists files specified by parameters in the [sidout] configuration section.

Configuration parameter	Input/output type	File type	Default directory
Output	Output	CTM	TempDir

[speakerid] Section

This table lists files specified by parameters in the [speakerid] configuration section.

Configuration parameter	Input/output type	File type	Default directory
AstFile	Input	AST	TempDir
BaseAstFile	Input	AST	Server
DiagFile	Output	DIAG	TempDir
SigFile	Input	SIG	Server
USMFile	Input	USM	Server

[stt] Section

This table lists files specified by parameters in the [stt] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DiagFile	Output	DIAG	TempDir

[textnorm] Section

This table lists files specified by parameters in the [textnorm] configuration section.

Configuration parameter	Input/output type	File type	Default directory
PgfFile	Input	PGF	Server
TxtFileIn	Input	TXT	DataDir
TxtFileOut	Output	TXT	DataDir

[textsegment] Section

This table lists files specified by parameters in the [textsegment] configuration section.

Configuration parameter	Input/output type	File type	Default directory
PgfFile	Input	PGF	Server
TxtFileIn	Input	TXT	TempDir
TxtFileOut	Output	TXT	TempDir

[transcheck] Section

This table lists files specified by parameters in the [transcheck] configuration section.

Configuration parameter	Input/output type	File type	Default directory
DiagFile	Output	DIAG	TempDir
InputTranscript	Input	TXT	TempDir
OutputTranscript	Output	TXT	TempDir
WarningFile	Output	TXT	TempDir

[wav] Section

DEPRECATED:
The wav module is deprecated in IDOL Speech Server version 11.5 and later. HPE recommends that you use the audio module instead.

This module is still available for existing implementations, but it might be incompatible with new functionality. The module might be deleted in future.

This table lists files specified by parameters in the [wav] configuration section.

Configuration parameter	Input/output type	File type	Default directory
WavFile	Input	WAV	DataDir

[wout] Section

This table lists files specified by parameters in the [wout] configuration section.

Configuration parameter	Mode	Input/output type	File Type	Default Directory
Output	WRITE or _	Output	CTM	TempDir

Appendix B: IDOL Speech Server File Types

This appendix describes the different files that IDOL Speech Server produces and uses.

- File Types 275

File Types

IDOL Speech Server produces and uses files with the following file name extensions.

> **NOTE:**
> If FFmpeg support is enabled, IDOL Speech Server also accepts a wide range of media file formats when it uses the `audio` module to read from files.

AM	Acoustic model file
AST	Speaker ID classifier set
ATD	Audio template development file
ATF	Audio template file
ATS	Audio template set file
ATV	Audio template feature vector file
DCT.SZ	Pronunciation dictionary file
DIAG	Diagnostics file
DNN	Deep Neural Network acoustic modeling file
FMD	Phoneme time track file
FXX	Audio Fingerprint database index file
IAN	Input acoustic channel normalization file
IV	An iVector file, representing a speaker (audio) template.
IVP	An iVector parameter file, used as input for training, and so on.
IVD	An iVector development file, used in threshold estimation.
IVS	An iVector set file, containing a set of iVector templates.
LCF	Language classifier file
LIF	Language identification feature file
LIST	List file

LOG	Log file
NET	Phrase search supplementary file
NET.DPT	Phrase search supplementary file
OTD	Obfuscated training data file

> **DEPRECATED:**
> This file type is used only by the deprecated `DataObfuscation` task.

PDT.SZ	Word confidence supplementary file
PGF	Pronunciation generation file
PHI.SZ	Acoustic model language key files
PHN.SZ	Acoustic model language key files
PLM	Word confidence supplementary file
PRB	Word confidence supplementary file
SID	Segmenter model file

> **DEPRECATED:**
> This file type is used only by the deprecated `SegmentWav` task.

SIG	Signal processing description file
SPK	Individual speaker ID model file

> **DEPRECATED:**
> This file type is used only by deprecated speaker identification tasks.

SPO	Speaker ID optimization file

> **DEPRECATED:**
> This file type is used only by deprecated speaker identification tasks.

SPT	Speaker training file

> **DEPRECATED:**
> This file type is used only by deprecated speaker identification tasks.

TKI	Audio Fingerprint database index file
TLM	Language model file
TTX	Audio Fingerprint database index file
TXT	Text file
USM	Universal Speech Model file
VOC	Vocabulary list file

WAV	WAV audio file
WDS	Valid word list file (supplied in language pack)
XML	XML file

Appendix C: Audio Quality Guidelines

This appendix describes the audio properties required for accurate speech processing.

Several factors affect the recall rate (correct detection) of words or phrases:

- Signal bandwidth
- Background noise
- Speech clarity, which can be affected by factors such as whether a speaker is native
- Audio signal distortion, due to compression and storage
- Breadth of language context

For best speech processing results, ensure that your audio conforms to the following guidelines:

- The sampling frequency must be at least the sampling frequency required by the processing task. Audio files with sampling frequencies below this are upsampled, which causes severe quality issues. The minimum sampling frequencies are 8 kHz for telephony audio and 16 kHz for broadcast audio.
- The minimum SNR (signal-to-noise ratio) is 15 dB. An SNR of 25 dB or above produces the best results. This ratio is measured across the word or phrase being detected and not across the entire audio.
- Words or phrases must be articulated reasonably clearly, and largely conform to the language. Speech-to-text performance is known to be poorer for non-native speakers than for native speakers.
- Natural speech rates produce the best speech-to-text results. Speech that is faster or slower than this usually produces more errors.
- When word confidences are enabled, every recognized word is associated with an acoustic confidence value. Generally, false positives tend to have a lower acoustic confidence compared to true hits.
- Newer audio codecs offer less distortion for the same rate of compression.
- If the language context of the content is too broad, the effectiveness of the language model is reduced.

Appendix D: Audio Transcript Requirements

This appendix outlines the requirements for audio transcripts that you use to build custom language models or adapt acoustic models.

- Use plain text (.txt) files and UTF-8 character encoding.
- Transcribe all speech verbatim.
- Use upper and lower case in accordance with common usage.
- Use correct punctuation. Mark the ends of sentences with periods (.). Use other punctuation, such as question marks and exclamation marks, as required. Add a white space after punctuation at the end of a sentence; for example ". A" not ".A".
- Do not use single or double quotation marks.
- Transcribe acronyms in accordance with common usage. Do not use white space or periods between the characters; for example, “FBI”, “SNCF”, “NATO”.
- Transcribe false starts and interjections, even if the speaker pronounces only half of the word; for example, “good eve- er good evening”. Transcribe these interjections as they sound; the following examples are all acceptable: “Uh-huh”, “Er”, “Huh”, “Ah”, and “Um”.
- If you cannot clearly hear a word, use your best guess for transcribing it. Any word that sounds similar to what you can hear and is appropriate to the sentence is acceptable.
- Spell proper nouns in accordance with common usage.
- Insert a paragraph break on speaker changes. No other indication of speakers is required.
- If you are not sure of the spelling of a name, use your best guess; for example, “Now over to our correspondent Christopher Balgarick”.
- Where speakers talk over each other or the speech is too loud, mark it using your own discretion.

Manually Normalize Text

IDOL Speech Server can normalize text in many languages (see Supported Languages, on page 139) but not all. For languages for which text normalization is not supported, you must manually normalize the files.

To manually normalize files

- Change digits to words. For example, replace "2" with "two", and "1997" with "nineteen ninety seven".
- Write individually pronounced character sequences as spaced characters; for example, “the word rules is spelled R U L E S”. Write pronounced punctuation as it sounds; for example, “Tony at BBC dot co dot UK.
- For all sentence breaks, replace periods (.) with <s>.

Appendix E: Compatibility Notes

IDOL Speech Server 10.8 introduced Deep Neural Network (DNN) Acoustic Modeling. DNN modeling uses files with the extension `.dnn` in a language pack. These files were introduced in IDOL Speech Server language pack version 6.0 or later. In general, recent IDOL Speech Server releases before 10.8 can still use these Language Packs, but they ignore the `.dnn` files and use only the underlying AM file for speech-to-text.

IDOL Speech Server releases 10.10, 10.11, and 11.5 also introduce additional support for new Language Pack Neural Network technologies. The following table describes the minimum version of IDOL Speech Server that you need to run each version of the language pack.

Language Pack Version	Minimum Server Version
6.x	10.8
7.x	10.10
8.x	10.11
9.x	11.5

If you use an older version of IDOL Speech Server, the server either reports a compatibility error, or runs without the Neural Network technology.

All versions of the server are backward compatible with pre-existing language packs.

In addition, language packs with version 6.0 or later use a newer format of the Language Model (version 3). This change means that IDOL Speech Server version 10.3 and earlier are not compatible with these newer language packs.

Glossary

A

ACI (Autonomy Content Infrastructure)
A technology layer that automates operations on unstructured information for cross-enterprise applications. ACI enables an automated and compatible business-to-business, peer-to-peer infrastructure. The ACI allows enterprise applications to understand and process content that exists in unstructured formats, such as email, Web pages, Microsoft Office documents, and IBM Notes.

ACI Server
A server component that runs on the Autonomy Content Infrastructure (ACI).

ACL (access control list)
An ACL is metadata associated with a document that defines which users and groups are permitted to access the document.

acoustic model
A statistical model that captures the frequency patterns of speech.

action
A request sent to an ACI server.

active directory
A domain controller for the Microsoft Windows operating system, which uses LDAP to authenticate users and computers on a network.

B

bandwidth
The frequency range in audio.

beam width
A parameter associated with how the search for recognized word sequences is performed. It is similar to beam search in dynamic processing.

broadband language pack
A 16 kHz language pack, generally used for high-quality audio.

C

Category component
The IDOL Server component that manages categorization and clustering.

change point detection
Detection of the point at which an audio property changes its value. Such a property can be the speaker or language.

Community component
The IDOL Server component that manages users and communities.

connector
An IDOL component (for example File System Connector) that retrieves information from a local or remote repository (for example, a file system, database, or Web site).

Connector Framework Server (CFS)
Connector Framework Server processes the information that is retrieved by connectors. Connector Framework Server uses KeyView to extract document content and metadata from over 1,000 different file types. When the information has been processed, it is sent to

an IDOL Server or Distributed Index Handler (DIH).

Content component
The IDOL Server component that manages the data index and performs most of the search and retrieval operations from the index.

D

DAH (Distributed Action Handler)
DAH distributes actions to multiple copies of IDOL Server or a component. It allows you to use failover, load balancing, or distributed content.

DIH (Distributed Index Handler)
DIH allows you to efficiently split and index extremely large quantities of data into multiple copies of IDOL Server or the Content component. DIH allows you to create a scalable solution that delivers high performance and high availability. It provides a flexible way to batch, route, and categorize the indexing of internal and external content into IDOL Server.

downsampling
The process of reducing the sampling rate.

I

IDOL
The Intelligent Data Operating Layer (IDOL) Server, which integrates unstructured, semi-structured and structured information from multiple repositories through an understanding of the content. It delivers a real-time environment in which operations across applications and content are automated.

IDOL Proxy component
An IDOL Server component that accepts incoming actions and distributes them to the appropriate subcomponent. IDOL Proxy also performs some maintenance operations to make sure that the subcomponents are running, and to start and stop them when necessary.

Intellectual Asset Protection System (IAS)
An integrated security solution to protect your data. At the front end, authentication checks that users are allowed to access the system that contains the result data. At the back end, entitlement checking and authentication combine to ensure that query results contain only documents that the user is allowed to see, from repositories that the user has permission to access. For more information, refer to the IDOL Document Security Administration Guide.

K

KeyView
The IDOL component that extracts data, including text, metadata, and subfiles from over 1,000 different file types. KeyView can also convert documents to HTML format for viewing in a Web browser.

L

Language code
A four or five letter abbreviation that denotes the language and country for a language pack.

language model
A statistical model that captures word sequence patterns and probabilities.

language pack
A set of files that contain the acoustic model, one or more language models, and other

components to perform speech recognition on a language as spoken in a given country or region.

LDAP
Lightweight Directory Access Protocol. Applications can use LDAP to retrieve information from a server. LDAP is used for directory services (such as corporate email and telephone directories) and user authentication. See also: active directory, primary domain controller.

License Server
License Server enables you to license and run multiple IDOL solutions. You must have a License Server on a machine with a known, static IP address.

M

module
A set of internal operations that Speech Server performs. You can combine modules to create tasks or processing schemas that Speech Server can run to perform various audio processing operations.

O

OmniGroupServer (OGS)
A server that manages access permissions for your users. It communicates with your repositories and IDOL Server to apply access permissions to documents.

P

phoneme
The basic unit of speech used in speech recognition.

primary domain controller
A server computer in a Microsoft Windows domain that controls various computer resources. See also: active directory, LDAP.

processing schema
A task created in Speech Server to process audio using a series of configured processing modules.

Pulse Code Modulation (PCM)
PCM is the most frequently used analog-to-digital conversion technique, and is the standard form of digital audio in digital telephony and other digital audio applications. PCM converts analog signals to binary values that are represented as a series of pulses.

R

real-time scaling
The rate at which an audio signal plays relative to real time. For example, if the audio plays at twice real time, its real-time scaling value is 2.

S

sampling rate
The rate at which an analog audio signal is obtained during conversion to digital.

schema
A network of modules connected by their inputs and outputs. This network specifies the modules to use and the order, for an audio analysis operation.

Simple Object Access Protocol (SOAP)
An XML-based protocol that specifies a method that web service applications can use to exchange information over networks, regardless of the operating system, platform, or programming language that they are using.

SOAP messages can be exchanged by using a variety of internet protocols, including SMTP, MIME, and HTTP.

speaker identification
The process of identifying known speakers in audio.

speech-to-text
The process of converting speech into text through speech recognition.

spoken language identification
The process of identifying the language or languages being spoken in the audio.

T

telephony language pack
An 8 kHz language pack, generally used for low quality audio.

transcript
A text file that contains the words spoken in a piece of audio.

transcript alignment
The process of matching a known sequence of words to the audio.

U

upsampling
The process of increasing the sampling rate. HPE does not recommend upsampling data, because it does not improve audio quality.

V

View
An IDOL component that converts files in a repository to HTML formats for viewing in a Web browser.

W

Wildcard
A character that stands in for any character or group of characters in a query.

WIndows Media Audio (WMA)
A proprietary Microsoft file format for encoding digital audio files. WMA offers faster compression than MP3, and WMA files can be compressed to match different connection speeds or bandwidths.

X

XML
Extensible Markup Language. XML is a language that defines the different attributes of document content in a format that can be read by humans and machines. In IDOL Server, you can index documents in XML format. IDOL Server also returns action responses in XML format.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Administration Guide (IDOL Speech Server 11.5)

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to AutonomyTPFeedback@hpe.com.

We appreciate your feedback!