IBM ViaVoice™ Telephony Tools
Programmer's Reference

IBM ViaVoice™ Telephony Tools
Version 1.2
Programmer's Reference

Printed in the U.S.A.

Note: Before using this information and the product it supports, be sure to read the general information under Legal Notices.

Fourth Edition (August 1999)

This edition applies to the IBM ViaVoice Telephony Tools and to all subsequent versions and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 1997, 1998, 1999. All rights reserved. Note to U.S. government users -- Documentation related to restricted rights -- Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.
Legal Notices

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States or in other countries.

- IBM
- ViaVoice

Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

Dialogic, D/240SC-T1 and D/41H are trademarks or registered trademarks of Dialogic Corporation.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Natural Microsystems, Alliance Generation-T1 and Alliance Generation - 8/80 are trademarks or registered trademarks of Natural Microsystems Corporation.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.
Table of Contents

About This Guide and Reference

- Who Should Read this Guide and Reference
- Prerequisites
- How this Guide and Reference is Organized
- What's New in this Version
- Related Documentation

Part 1. Introduction to ViaVoice Run Time and Tools

Introduction

- What is Phone Recognition?
- Examples of Phone Recognition Applications
- What is the Phone Recognition Engine?
- Phone Recognition Engine Architecture
- ViaVoice Telephony Run Time Content
- ViaVoice Telephony Tools Content

Installation

- Required Hardware and Software
- The ViaVoice Telephony Run Time
- Installing the ViaVoice Telephony Run Time
- ViaVoice Telephony Run Time Directory Structure
- Removing the ViaVoice Telephony Run Time
- The ViaVoice Telephony Tools
- Installing the ViaVoice Telephony Tools
- ViaVoice Telephony Tools Directory Structure
Part 2. Defining the Telephony Environment

ViaVoice Telephony Processes

- Process Manager (pm)
- TCL Channel Process
- Engine Process
- Logger Process
- Statlog Process

Via Voice Telephony in a Client-Server Configuration (Remote Recognition Engine)

- Installing the ViaVoice Telephony Run Time on a Server
- Installing the ViaVoice Telephony Run Time on a Client Machine
- Defining Client-Server Configuration Resources
- Defining Processes (procman.cfg)
- Configuring the Client VTTDefaults File for Remote Speech Reco
- Client Side Configuration of procman.cfg
- Configuring the Server VTTDefaults File for Remote Reco
- Configuring the Server Procman.cfg File for Remote TTS

Specifying Shared Access

ViaVoice Telephony Resources

- Process Manager (procman.cfg)
- Prompts(promptConfig)
- VTTDefaults Resources Overview
- VTTDefaults:Directories
- VTTDefaults:Logging
- VTTDefaults:Process Manager Resources
- VTTDefaults:Prompts
Part 3. Developing Grammars

Phone Recognition Grammars Overview

- About Vocabularies
- What is a Phone Grammar?
- Examples of Simple Phone Grammars
- Why is a Phone Grammar Necessary?
- Dynamic Command Vocabularies
- Acceptance or Rejection of Utterances
- Guidelines for Designing Grammars

Developing Phone Grammars

- Developing Grammars Using Lists
  - Using the makeApp Utility
  - Preparing Words and Phrases for Phone Recognition
  - The makeApp Utility
- Developing Grammars Using SRCL
  - Introduction to SRCL
  - Preparing SRCL Specifications
  - Creating the Pronunciation Dictionary
  - Creating the Pronunciation Pool

Detailed Information on the SRCL Command Language
• Defining Common Words and Phrases with Nonterminal Symbols
• Defining Optional Words and Phrases
• Defining Repeated Words and Phrases
• Grammar Annotations -- A Post Parsing Aid
  • Defining Annotations
  • The Kiosk Example
• SRCL Syntax
  • Language Definition
  • Language Elements
  • Grammar Rules
• Extensions to SRCL - Specifying Weights
• Using the Translation Facility
  • Working with the Details
  • Translation Example

Developing Grammars using the Administration Tools
• Building your Application Grammars
  • Selecting Your Application
• Generating Grammars
  • Generating Grammars from subscriber.db files
  • Generating Grammars from BNF files
  • Generating Baseforms
  • Editing Baseforms
  • Phonetic Pronunciation
  • Displaying Status
  • Displaying Build Information
• Grammar Development Tools
• The makeGrammar Utility
• The makeBsfms Utility
• The makePol Utility

• The Grammar Compiler - VTBNFC
  • Using VTBNFC from the Command Line
  • Compiling a Grammar
  • Using the Grammar Compiler API
  • Grammar Compiler Error Messages

• Generating Baseforms From Vocabularies

• Testing Your Phone Grammars
  • Tools for Testing Grammars
    • fsgenum
    • fsgprint
    • fsgtest

Part 4. Developing Telephony Applications

The Tcl Extensions

• The Extended Tcl Shell (tclch)
  • Using the tclch Program
  • GetResource
  • LogEvent
  • SetDebug

• The ernestine Library
  • AddRecoCmd
  • AddTelCmd
  • AddVoiceCmd
  • CopyCmd
• DelCmd
• Dispatch
• ListCmds
• LogStat
• ModCmd
• PrintCmd
• Error Returns

• Tcl Extensions for Version 1.2
  • Logging Commands
    • GetResource
    • LogEvent
    • LogStat
  • Speech Recognition Commands
    • DisableVocab
    • EnableVocab
    • LoadGrammar
    • Recognize
    • StartRecognize
    • StopRecognize
    • UnloadGrammar
  • Telephony
    • AnswerCall
    • BlockCalls
    • Dial
    • MakeCall
    • Prompt
• TerminateCall
• TransferCall
• UnblockCalls
• User Event Handling
• Error Returns

C++ Telephony Classes

The Audio Class
  • aud_write()
  • AudioInit()
  • getParams()
  • setParams()

The libBase Class Library
  • VttMemoryBuffer Class
    • VttMemoryBuffer()
    • getSize()
    • refCount()

The libTsm Class Library
  • TsmDelete()
  • TsmDisableVocab()
  • TsmEnableVocab()
  • TsmEndAppi()
  • TsmGetAnno()
  • TsmGetLastError()
  • TsmGetNbest()
  • TsmGetNbestAnno()
  • TsmGetText()
• TsmInit()
• TsmLoadContext()
• TsmRetrieveSynth()
• TsmSaveContext()
• TsmSendText()
• TsmStartAppl()
• TsmStartReco()
• TsmStopReco()
• TsmTerm()
• TsmTrain()
• TsmUnloadContext()
• TsmWait()
• TsmWordData()

• The Resource Manager Class
  • ResourceMgr() Constructor
  • ~ResourceMgr() Destructor
  • doLog()
  • finish()
  • hexDump()
  • init()
  • initialize()
  • resRequestBool()
  • resRequestFloat()
  • resRequestInt()
  • resRequestStr()
  • resRequestStrArray()
- resRequestUStr()

- **The vttTts Class**
  - vttTts() Constructor
  - ~vttTts() Destructor
  - ttsInit()
  - ttsSetPitchBaseline()
  - ttsSetPitchFluctuation()
  - ttsSetSpeed()
  - ttsSetVolume()
  - ttsStop()
  - ttsText()

- **Writing the Application Interface**

- **The Preliminary Tasks**

- **Basic Phone Recognition Tasks**

- **Programming Tasks for TCL Applications**
  - Enabling and Disabling Vocabularies
  - Directing the Engine to Process Speech
  - Processing Recognized Speech

- **Programming Tasks for C++ Applications**
  - Header files < .hused by the Grammar Compiler API
  - Connecting to the Engine
  - Enabling and Disabling Vocabularies
  - Directing the Engine to Process Speech
  - Processing Recognized Speech
  - Disconnecting from the Speech Engine

- **Sample Programs**
• `appletTester.tcl`
• `ndsample.tcl`
• `appletTester Script Source`
• `ndsample Script Source`
• `C++ Sample:recotest`
  • `Building recotest`
  • `Running recotest`
  • `recotest Source`
• `Sample Grammars`
  • `Sample Grammar Vocabularies - U.S. English`
    • `Banking`
    • `Commands`
    • `Dates`
    • `Digits and digComp`
    • `Dollars`
    • `Natural Numbers`
    • `Spelling`
    • `Times`
    • `Voice Activated Network Control`
    • `Voice Mail`
    • `Yes/No`
  • `Sample Grammar Vocabularies - U.K. English`
    • `Banking`
    • `Commands`
    • `Dates`
    • `Digits and digComp`
• Spelling
• Times
• Voice Activated Network Control
• Voice Mail
• Yes/No

• Sample Grammar Vocabularies - French
  • Banking
  • Commands
  • Voice Activated Network Control
  • Voice Mail
  • Yes/No

• Sample Grammar Vocabularies - German
  • Banking
  • Commands
  • Dates
  • Digits and digComp
  • Spelling
  • Times
  • Voice Activated Network Control
  • Voice Mail
  • Yes/No

• The Grammar Parser
  • Dates
  • Times

Utilities
• The DialWatch Utility
About this Reference

This Reference provides detailed information on developing Windows NT** speech-aware telephony applications using the IBM ViaVoice Telephony Tools.

Who Should Read This Reference

Read this Reference if you are a programmer interested in writing Windows NT telephony applications that use ViaVoice for speech recognition.

Prerequisites

Readers of this Reference should be familiar with the following programming concepts and techniques:

- Telephony
- Speech recognition
- Object-oriented programming
- C++
- Perl
- Tcl/Tk

How This Reference is Organized

This Reference provides information on the following:

**Part 1. Introduction**

*Introduction* describes phone recognition application programming using the ViaVoice Telephony Tools. It explains how to use the Tools to develop a phone application that incorporates speech recognition.

*Installation* lists the hardware and software requirements for installing and using the Tools. It also explains the resulting directory structure when the Tools are installed.

**Part 2. Defining the Telephony System Environment**

*ViaVoice Telephony Processes* describes the Run Time processes that are provided in a ViaVoice Telephony system.

*ViaVoice Telephony Resources* describes the configurable resources provided for a ViaVoice
Telephony application.

**Part 3. Developing Grammars**

*Phone Recognition Grammars Overview* describes a phone grammar, provides examples of phone grammars, and describes handling interruptions in speech. It also provides guidelines for designing grammars.

*Developing Phone Grammars* explains how to use the ViaVoice Telephony Tools to develop grammars for use with phone recognition applications. It covers identifying grammars, creating grammar files, compiling grammars, building a pronunciation dictionary, and testing grammars.

**Part 4. Developing Telephony Applications**

*Tcl Extensions* describes the extensions that were made to the standard Tcl library for ViaVoice Telephony.

*C++ Telephony Classes* provides an overview of the C++ classes and methods provided with the Telephony Tools. It describes in detail, the classes and their supported member functions.

*Writing the Application Interface* presents the basic functions that phone recognition applications should support.

*Sample Programs* describes the two sample programs that are provided with the Tools along with the sample applet grammars that are provided for each language. These samples can be used as a reference to develop phone recognition applications.

*Utilities* describes several commands and utilities available to help you develop and test your phone recognition application.

**What’s New in this Version**

There are several new features provided in Version 1.2 of the ViaVoice Telephony Run Time:

- The Run Time is available in multiple languages (U.S. English, U.K. English, French and German).
- Multiple language versions of the Run Time can be installed on a single system.
- Additional telephony adapters are supported.
- The text-to-speech engine is supported in a client/server configuration.
- Topics are supported.
- For U.S. English only, two new acoustic models are provided: one that is optimized for the cellular environment and one that is optimized for small- to medium-sized vocabularies.

Additionally, the following new features are provided in Version 1.2 of the Tools:

- The Tools are available in multiple languages (U.S. English, U.K. English, French and German).
• Multiple language versions of the Tools can be installed on a single system. Note that a particular language version of the Tools can only be installed if the same language version of the Run Time is installed.

• A new sample (ndsample.tcl) that demonstrates both telephony and speech recognition concepts.

• Enhancements to the Tsm APIs.

• New grammar test tools.

• Improvements to the Programmer’s Reference.

• The AppletTester is now being installed with the Tools software. (NOTE: Because it is being installed with the Tools (and not the Run Time), the software has no way of knowing whether the installation is for a client or server; consequently, the appletTester is no longer automatically started. To automatically start the appletTester, you must create an entry in the procman.cfg file.)

Related Documentation

• IBM ViaVoice™ Telephony Natural Language Tools Developer’s Guide

• VVDetective Control Guide, IBM ViaVoice™ SDK for Windows®

• SAPI (Speech Application Programming Interface) Reference, IBM ViaVoice™ SDK for Windows®

• SMAPI Developer’s Guide, IBM ViaVoice™ SDK for Windows®

• SMAPI (Speech Manager Application Programming Interface) Reference, IBM ViaVoice™ SDK for Windows®

• ActiveX™ Developer’s Guide, IBM ViaVoice™ SDK for Windows®

• VVEngine Control Guide, IBM ViaVoice™ SDK for Windows®

• CT Access Developers Manual (from Natural Microsystems)

• CT Access Function Reference Manual (from Natural Microsystems)

• Voice Software Reference for Windows NT, Vol. II (from Dialogic Corporation)

Books on Tcl/Tk programming such as:

• Ousterhout, John K. Tcl and the Tk Toolkit. USA: Addison-Wesley Publishing Co., 1994


Books on Perl programming, such as:
Part 1. Introduction

Introduction to ViaVoice Telephony Run Time and Tools

Version 1.2 of the IBM ViaVoice* Telephony Run Time and Tools for Windows NT** provides programmers with tools to develop phone recognition applications. The ViaVoice Telephony Run Time provides the speech recognition platform over which phone recognition applications run. The ViaVoice Telephony Tools provide a set of utilities for developing and testing phone recognition applications. The Tools provide extensions to the Tcl/Tk scripting language to make application development and customization easier. Also included is a set of C++ classes and member functions that enable the programmer to develop object-oriented solutions. The Tools include two sample applications that can help programmers understand how to integrate speech recognition into telephony applications. Finally, there are several utilities that can be used to develop, test, and manage phone recognition applications.

What is Phone Recognition?

Phone recognition is an extension of speech recognition. Speech recognition refers to using a microphone (or other input device directly attached to a computer) to speak to a computer. The computer recognizes what was said and the associated application takes appropriate action. Phone recognition, on the other hand, refers to talking through a telephone over a telephone line to a computer. As with speech recognition, the computer recognizes what was said and the telephony application takes appropriate action.

Speech recognition can augment and even replace keyboard and mouse interaction with the computer. For example, instead of typing text, the user can dictate text to the computer. Or, the user can control applications through voice commands instead of using the mouse. Similarly, phone recognition can augment or replace the phone keypad. The caller can speak a request, rather than having to use the phone keypad to interact with the phone application. This is particularly useful in Voice Response Unit (VRU) applications, where a complex VRU menu can become cumbersome for the end user, or where alphabetic rather than numeric information needs to be entered.

For more information on speech recognition in general, refer to the IBM ViaVoice Developer Tools Speech Programmer's Guide.

Examples of Phone Recognition Applications

There are many ways that speech recognition can be incorporated into telephony applications to enhance usability, productivity, and effectiveness. One example is a voice-based phone operator that could be employed throughout an enterprise. Instead of speaking to a person, the caller speaks the name of a person ("Lou Gerstner"), and the phone application handling the call recognizes what was spoken and transfers the caller to the correct phone number.

Using speech recognition in telephony applications that provide financial services is also beneficial. Many of these applications require several key presses to enter, verify, and complete a transaction. Wouldn't it
be easier to say "Transfer five hundred dollars from savings to checking?"

**What is the Phone Recognition Engine?**

A ViaVoice Telephony application uses the standard speech recognition engine provided by ViaVoice. Some additional data files optimized for telephony are included.

**Phone Recognition Engine Architecture**

A phone recognition system is implemented as a dedicated server using Windows NT 4.0 Workstation. The phone recognition server utilizes the standard speech recognition engine provided by ViaVoice. Refer to the *ViaVoice Developer Tools Speech Programmer’s Guide* for more information on the speech recognition engine architecture.

The architecture of the phone recognition server is illustrated and described below:

**ViaVoice Telephony Architecture**

The processes that comprise the phone recognition server are:

**Process Manager:** This process is responsible for the startup of system resources in a configurable fashion. It is capable of running on separate machines and managing just the resources for that machine. In addition, it sequences the startup of the service.

**Channel Process:** This process is responsible for carrying out the call flow. For each channel supported, a channel process is created. A channel process is connected to only one recognition engine at a time.
Note that the channel process uses an Interactive Voice Response (IVR) script to manage its call flow. This script is implemented in the Tcl scripting language. Therefore, throughout this Programmer's Reference, a channel process is also called a tclchp process. The Tcl functions are provided in a DLL.

**Engine:** This process is an instance of the ViaVoice continuous speech recognition engine. For each channel supported, an engine process is created.

In addition to the above processes, there are two other processes that are integral to the ViaVoice Telephony system. These processes are:

**Logger:** This process is responsible for collecting log events from the other telephony processes. This information is available to utilities (such as DialWatch) that may be monitoring the system.

**StatLog:** This process is responsible for collecting statistics from each of the channel processes and producing reports.

For more detailed descriptions of each of these processes, refer to [ViaVoice Telephony Processes](#).

### ViaVoice Telephony Run Time Content

The ViaVoice Telephony Run Time provides the platform over which phone recognition applications are developed and run. It contains the following software components:

- Speech recognition task and data
- ViaVoice Telephony processes
- ViaVoice Telephony resources
- Run Time libraries for Tcl and C++
- Grammar development tools
- Baseform dictionaries
- Text2pcm utility
- Sample application grammars (FSGs) and baseforms
- Sample Tcl script
- System utilities to monitor and administer an application
The ViaVoice Run Time also includes several software packages that enable developers to write phone recognition applications in a web-centric environment. The complete set of software that is installed with the Tools is:

<table>
<thead>
<tr>
<th>Software Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViaVoice 98 Speech Recognition Run Time</td>
<td>Provides the speech recognition engine, optimized for telephony.</td>
</tr>
<tr>
<td>ViaVoice Outloud Run Time</td>
<td>Provides the text-to-speech engine.</td>
</tr>
<tr>
<td>Netscape FastTrack Server</td>
<td>Enables management of telephony applications through the web.</td>
</tr>
<tr>
<td>Netscape Communicator</td>
<td>Enables web-based documentation and administrative tools.</td>
</tr>
<tr>
<td>Perl</td>
<td>Provides the Perl scripting environment which is used for the grammar development tools.</td>
</tr>
<tr>
<td>Dialogic software</td>
<td>Device software to support the installed Dialogic adapter.</td>
</tr>
<tr>
<td>Natural Microsystems software</td>
<td>Device software to support the installed Natural Microsystems adapter.</td>
</tr>
</tbody>
</table>

If the installation process detects that any of the software components already exist on the system, it provides an option to skip the installation for those components.

**ViaVoice Telephony Tools Content**

The ViaVoice Telephony Tools provide several mechanisms for incorporating speech recognition into telephony applications. The functions provided with the Tools are:

<table>
<thead>
<tr>
<th>Grammar Development Tools</th>
<th>Utilities that generate grammars and baseforms for telephony applications.</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Using word lists for simple vocabularies</td>
<td></td>
</tr>
<tr>
<td>• Using the Administration Tools GUI</td>
<td></td>
</tr>
<tr>
<td>• Using the Command Line Grammar Development Tools</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TCL Extensions</th>
<th>Extensions to the TCL scripting language through which a telephony application manages its call flow.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ Telephony Classes</td>
<td>Object-oriented, C++ interfaces to the speech recognition</td>
</tr>
<tr>
<td><strong>Utilities</strong></td>
<td>Commands and utilities to monitor the activity on a call line.</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Samples</strong></td>
<td>A sample TCL script and a sample C++ program. Also, several sample grammars are also provided.</td>
</tr>
</tbody>
</table>
ViaVoice Telephony Installation

This section describes the hardware and software required to install and use ViaVoice Telephony. It also describes how to install the products and provides details about the directory structures that result from installation.

ViaVoice Telephony consists of two products:

- ViaVoice Telephony Run Time
- ViaVoice Telephony Tools

The two products are packaged on a single CD-ROM and can be installed separately. Note that the Tools requires that the Run Time is installed; therefore, if you try to install the Tools without installing the Run Time first, the Tools will install the Run Time.

Required Hardware and Software

To use the ViaVoice Telephony Run Time and the ViaVoice Telephony Tools, you need:

- A personal computer with a Pentium Pro 200MHz or faster microprocessor with at least 128MB RAM (256MB if running as a telephony server)
- Approximately 150MB of free disk space
- Microsoft** Windows NT Workstation Operating System V4.0 with Service Pack 4
- A telephony adapter. The following are supported:
  - Dialogic** D/41ESC
  - Dialogic D/41H
  - Dialogic D/240SC-T1
  - Dialogic D/300SC-E1
  - Natural Microsystems** AG-8
  - Natural Microsystems AG-E1
  - Natural Microsystems AG-T1
  - Natural Microsystems QX2000-100/4L.

Optionally, you need a sound card if you want to listen to the text-to-speech pronunciation of baseforms from the Administration Tools.

The following software is optional and depends on how you plan to implement your telephony applications:
• Microsoft Visual C++ Version 5.0

The ViaVoice Telephony Run Time

The ViaVoice Telephony Run Time includes the following software components:

• Install
• Uninstall
• ViaVoice Telephony
  • Telephony tasks and data
  • Telephony processes (Process Manager, StatLog, Logger, tclchp, tsm, tsmRouter, ttsp)
  • Pronunciation dictionaries (baseforms files)
  • Sample grammars (FSGs) and their associated pronunciation (baseforms) files
  • System utilities (DialWatch, listres, setdbg, logtail, text2pcm, tsmcon)
  • Grammar development tools (vtbnfc, makeApp.pl, makeGrammar.pl, makeBsfms.pl, makePol.pl)

The ViaVoice Telephony Run Time includes several other software packages that enable a web-centric, efficient development and Run Time environment. The following software components are installed with the ViaVoice Telephony Run Time:

• ViaVoice Speech Recognition Run Time V5.3
• ViaVoice Outl oud Run Time V4.0
• Netscape** FastTrack Server 3.01
• Netscape Communicator 4.08
• Perl** for Win32 5.005.02
• Dialogic DNA 3
• Natural Microsystems Natural Access V2.11

If the installation process detects that any of the software components already exist on the system, it provides an option to skip the installation for those components.
Installing the ViaVoice Telephony Run Time

Follow these steps to install the ViaVoice Telephony Run Time:

**IMPORTANT:** If you have previous versions of ViaVoice, or VoiceType products, or ViaVoice Telephony Run Time installed on your system, uninstall them before you install the current version of the ViaVoice Telephony Run Time. It is not necessary to re-install Windows NT.

Note also that you must have administrator privileges to install the ViaVoice Telephony Run Time.

Insert the ViaVoice Telephony CD-ROM in the appropriate drive:

1. On the Windows NT Task Bar, click **Start**, and then click **Run**.
2. Type cd x:\vvphone_runtime\setup and click **OK**. For **x**, substitute the letter of the drive containing the ViaVoice Telephony CD-ROM.
3. Follow the setup procedures to install the ViaVoice Telephony Run Time.

Upon completion of the Run Time installation, a folder named *ViaVoice Telephony Run Time* is created. The Start Menu items for this folder are:

- DialWatch
- Uninstall ViaVoice Telephony Run Time

ViaVoice Telephony Run Time Directory Structure

The installation procedure creates the following directory structure in the destination directory specified:

**Netscape** - Contains the Netscape Fast Track software.

**Perl** - Contains the Perl software.

**Program Files\Netscape** – Contains the Netscape Communicator software.

**tts** - Contains the ViaVoice Outloud Run Time.

**ViaVoice** – Contains the ViaVoice speech recognition engine.

**ViaVoicePhone** – Contains the ViaVoice Telephony Run Time files. The following subdirectories are also created:

- **app** – Contains sample grammar FSGs and baseforms files.
- **appbld** – Contains the IBM dictionary of pronunciations (baseforms).
- **bin** – Contains the ViaVoice Telephony programs, DLLs, utilities, and processes.
- **log** – Contains output files from the logger process.
- **pcm** – The default directory where speech recordings are stored.
- **stat** – Contains output files from the StatLog process.

Multiple language versions of the ViaVoice Telephony Run Time can be installed on the same system. Language-specific files are installed in subdirectories within the directory structure described above and identified by the language code (for example, `app\<Ll_LL>`), where Ll_LL is:

- **En_US** - U.S. English
- **En_UK** - U.K. English
- **Fr_FR** - French
- **Gr_GR** - German

### The ViaVoice Telephony Tools

The ViaVoice Telephony Tools provides developers with the necessary tools to write telephony applications that use ViaVoice for speech recognition. The following software components are installed with the Tools:

- Tcl library extensions and headers
- C++ interfaces
- Web-based grammar development and administration tools
- Grammar test tools
- Two sample Tcl scripts
- Sample C++ program
- Sample grammar BNFs
- ViaVoice Telephony Programmer's Reference in HTML and PDF format
Installing the ViaVoice Telephony Tools

Follow these steps to install the ViaVoice Telephony Tools:

Insert the ViaVoice Telephony CD in the CD-ROM drive.

1. On the Windows NT Task bar, click **Start**, and then click **Run**.

2. Type `cd x:\vvphone_tools\setup` and click **OK**. For `x`, substitute the letter of the drive containing the ViaVoice Telephony Tools CD.

3. Follow the setup procedure to install the ViaVoice Telephony Tools.

Upon completion of the Tools installation, a folder named **ViaVoice Telephony Tools** is created. The Start menu items for this folder are:

- Administration
- DialWatch
- Programmer's Reference
- Read This First
- Uninstall ViaVoice Telephony Tools
- ViaVoice Options
- Vocabulary and Topic Installer
- Vocabulary Expander
- Vocabulary Manager

ViaVoice Telephony Tools Directory Structure

The IBM ViaVoice Telephony Tools adds files to the **ViaVoicePhone** directory as follows:

- `app\<ff\ll>\<appname>\grammars` – Adds sample grammar BNFs.
- `bin` – Adds the grammar test tools.
- `doc\pgmref` – Contains the ViaVoice Telephony Tools Programmer’s Reference in HTML format. A PDF file is also provided (pgmref.pdf) so that you can print a copy of the documentation.
- `inc` - Contains the header files for C++ program development.
- `lib` - Contains the library files needed for C++ program development.
prompts<LI_LL> – Contains ULW format files to use for call prompting.
samples\recotest - Contains the source code for the C++ recotest sample.
scripts\samples – Contains the appletTester and ndsample TCL scripts.
stat – Contains output files from the statLog process.

Note that LI_LL specifies the language version of the Tools for which the files are being installed.

Verifying Your Installation

After installing all required hardware and software, make certain that both the VTTDefaults file and the procman.cfg file have been properly configured. To verify that your system is operating correctly, try the following:

1. Copy the following text and save it as <VTT_HOME\scripts\samples\helloworld.tcl>:

```tcl
puts " "
puts "Executing script [info script]"

# load the telephony TCL extensions library
load ernestine.dll

#Define the playback prompt
AddVoiceCmd Play -prompt [list "hello world. Welcome to IBM ViaVoice Telephony"]

#Answer the call
puts "Waiting for this channel to be called ..."
AnswerCall
puts "Call answered."

#Play prompt
puts "Playing the prompt..."
Play
puts "Prompt complete."
```
TerminateCall
puts "Hang-up"

2. Type the following:

tclch scripts\samples\helloworld.tcl -c0

You should see the following messages:

Executing script scripts\samples\helloworld.tcl
Waiting for this channel to be called...

3. Next, call the phone number to which this channel is associated. Listen for the prompt to play and watch the screen.

4. Hang up.

To verify that speech recognition is working, run one of the samples that are included with the Tools:

- **AppletTester**: Verifies basic speech recognition function and telephony call handling (i.e., playing prompts).
- **ndsampe.tcl**: Verifies speech recognition functions and more advanced telephony call handling such as transferring a call to another number and handling DTMF.
- **recoTes test sample**: Verifies speech recognition capability independent of telephony.

**Removing the ViaVoice Telephony Run Time**

To remove the ViaVoice Telephony Run Time from your system, click **Uninstall ViaVoice Telephony** from the ViaVoice Telephony Start menu.

**Removing the ViaVoice Telephony Tools**

To remove the ViaVoice Telephony Tools from your system, simply click **Uninstall ViaVoice Telephony Tools** from the ViaVoice Telephony Tools start menu.
Part 2. Defining the Telephony Environment

ViaVoice Telephony Processes

The ViaVoice Telephony Run Time system consists of several processes. These processes are:

<table>
<thead>
<tr>
<th>Process</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td>The ViaVoice speech recognition engine. For each channel in the system, an engine is created.</td>
</tr>
<tr>
<td>Logger</td>
<td>Collects log messages from all telephony processes that can be used for application-specific purposes.</td>
</tr>
<tr>
<td>Process Manager</td>
<td>Responsible for launching, monitoring, restarting, and stopping all telephony processes.</td>
</tr>
<tr>
<td>StatLog</td>
<td>Collects statistics from all telephony processes that can be used for application-specific purposes.</td>
</tr>
<tr>
<td>Tcl Channel</td>
<td>Handles a single voice channel. For each channel in the system, a Tcl channel process is created.</td>
</tr>
<tr>
<td>tsmp</td>
<td>Provides the capability to perform remote speech recognition.</td>
</tr>
<tr>
<td>tsmRouter</td>
<td>Monitors and tracks the status of tsmp and ttsp processes and routes connection requests to the appropriate engine.</td>
</tr>
<tr>
<td>ttsp</td>
<td>Provides the capability to perform remote text-to-speech.</td>
</tr>
</tbody>
</table>

The integration and interaction of these processes in the ViaVoice Telephony system are illustrated in the section Introduction - Phone Recognition Architecture and Using ViaVoice Telephony in a Client-Server Configuration.

The Process Manager Process (pm)

The Process Manager (pm) process is responsible for managing the interaction of all the telephony processes in the system. To do this, pm can function as a command, a service, or a daemon. The options that are passed to pm at startup determine how it functions.

The format for specifying how pm functions is:

pm [options] [command] [command argument]

The command line parameters are:
options – Specifies the scope of the command:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-v</td>
<td>Lists the values of the resources defined in the procman.cfg file.</td>
</tr>
<tr>
<td>-c filename</td>
<td>Specifies the configuration file that is read at startup. The default configuration file, procman.cfg, can be overridden with this option.</td>
</tr>
<tr>
<td>-h hostname</td>
<td>The name of the host for which the command will run.</td>
</tr>
</tbody>
</table>

command - A string that indicates the command to be processed.

command argument - Optional parameters passed to the command.

When the pm process is started, it first parses and analyzes the command line to determine the appropriate behavior. It communicates any parsing errors to the console.

**pm as a Command**

The pm process runs as a command mode if it is started with a command argument.

If applicable, a command may take an optional argument specifying a process or a group of processes (denoted below as `[processname[,cloneld]]` where `processname` is the name of the process and `cloneld` denotes the highest number instance of the process. Instances are numbered sequentially beginning with 0. By default, actions apply to all of the processes under control of pm.

The commands that are supported by pm are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Forces re-read of the configuration file and start/restart of pm.</td>
</tr>
<tr>
<td>exit</td>
<td>Forcefully stops all telephony processes and then exits or stops pm.</td>
</tr>
<tr>
<td>install</td>
<td>Installs pm as a service if it is not yet installed.</td>
</tr>
<tr>
<td>list</td>
<td>Returns a formatted list of all of the processes under pm control.</td>
</tr>
<tr>
<td>list [processname[,cloneld]]</td>
<td>Returns a formatted listing of the status of the specified instance of a process.</td>
</tr>
<tr>
<td>quit</td>
<td>Gracefully stops all telephony processes. Note that quit does not stop pm.</td>
</tr>
<tr>
<td>remove</td>
<td>Removes pm as a service.</td>
</tr>
<tr>
<td>restart [processname[,cloneld]]</td>
<td>Stops the specified processes and their dependents and then restarts them in reverse order.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>run</td>
<td>Starts pm as a service.</td>
</tr>
<tr>
<td>start [processname[.cloneId]]</td>
<td>Starts the specified processes.</td>
</tr>
<tr>
<td>stop [processname[.cloneId]]</td>
<td>Gracefully stops the specified running processes. If the time expires without response, this will stop all processes forcefully. pm itself does not exit.</td>
</tr>
</tbody>
</table>

**pm as a Daemon**

If pm is started without any command specified, it operates as a daemon. It first starts as a command, checks to make sure a pm process is not already running (only one pm process per host is allowed). Next, it parses its configuration file (procman.cfg) and validates prerequisites (primarily the presence of various directories). If these actions are successful, pm is launched in the background using options appropriate for a daemon. At this point, the foreground pm exits, and the pm daemon creates a lock (a semaphore which uses the path name of the executable file as its name). The pm daemon process then parses the rest of its configuration file.

If any command is present, pm operates as a command. Some of the commands may launch a second pm with a new set of options (actually starting pm as a service). For these commands, the configuration file is read and validated before launching the background process.

**pm** keeps track of each process that registers with it. It keeps a list of process objects (one entry for each process under control of pm). If the AutoStart resource is set in the procman.cfg file, pm launches these processes. pm also listens for commands submitted by another (local or remote) instance of pm that is running as a command.

**pm as a Service**

The pm process can be installed and run as a Windows NT service. As such, pm retains all of the function described in the previous section. In addition, pm wraps NT service functions. To start pm as a service, enter:

```
pm install
```

where:

*username* - The Administrator ID on the system.

*password* - The Administrator's password.

**Tcl Channel Process (tclchp)**

A Tcl Channel process (tclchp, pronounced "tickle chip") manages a single voice channel. A separate tclchp process is started for each voice channel supported.

A tclchp process is the call flow interpreter for the telephony application. It is the interface between the voice channel and the following:
• Call Flow
• Telephony
• Speech Recognition
• Audio Services (prompts, text-to-speech)

Refer to Tcl Extensions for more information on the specific functions provided by the ViaVoice Telephony Tools.

Tcl Scripts

A Tcl script controls an application's call flow. As the user (caller) sees it, the behavior of the telephony system is controlled by the Tcl script. The tclchp process finds the name of the Tcl script to use from the VTTDefaults file, and passes this script to the Tcl interpreter.

The tclchp Process

The tclchp process creates a framework in which the Tcl script executes. Customization of the process for a specific application is accomplished by overriding ViaVoice telephony resources.

The Engine Process

The ViaVoice engine performs speech recognition. For each channel supported, the corresponding tclchp process creates an instance of the ViaVoice engine. It is important to note that for a local engine configuration, an engine process is associated with one and only one tclchp process.

The Logger Process

The logger process collects messages from all of the other processes running under control of the pm process and writes these messages to a log file named current.log. The logger also maintains a registry of system classes and shared memory segments that specify the debug level for each registered class.

The log file is named current.log and is stored in the <VTT_HOME>\log directory.

All processes that are under the control of pm access the logger process as clients. Clients deliver messages to the logger through the debug level that is associated with each <process>.<cloneId>.<className> category. When the first instance of a specific class is constructed in the calling process, a registration message is sent to the logger. The logger then provides an index into a shared memory segment where the debug level for this category is stored. Any subsequent instantiation of a registered class will obtain the same index. Note that the shared memory segment is read-only by other processes.

The debug level is used to filter messages within the client space (messages marked with a severity lower than debug level aren't delivered to logger). A client specifies which messages are logged by using the following format:

<severity> <time stamp> <sender> <message body>
The severity field is a three-letter code indicating the type of message to log. The type of message indicates the amount of information that is logged in the message. The following table lists the codes in the order of most to least information logged.

<table>
<thead>
<tr>
<th>CODE</th>
<th>TYPE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRC</td>
<td>Trace</td>
<td>Messages used for debugging purposes. This is the most detailed debug level. It generates a large volume of output.</td>
</tr>
<tr>
<td>DBG</td>
<td>Debug</td>
<td>Messages used for debugging purposes. This level is similar to trace but generates less output.</td>
</tr>
<tr>
<td>INF</td>
<td>Informational</td>
<td>Messages used for debugging purpose. This level produces a moderate amount of debug information.</td>
</tr>
<tr>
<td>WRN</td>
<td>Warning</td>
<td>Messages which describe errors from which the system automatically recovers.</td>
</tr>
<tr>
<td>ERR</td>
<td>Severe Error</td>
<td>Errors where recovery is possible but not guaranteed.</td>
</tr>
<tr>
<td>ADV</td>
<td>Advisory</td>
<td>Messages which recommend changes to system settings.</td>
</tr>
<tr>
<td>FAT</td>
<td>Fatal Error</td>
<td>Errors which are not recoverable. The reporting process must be restarted.</td>
</tr>
</tbody>
</table>

The time stamp field is controlled through resources and may include milliseconds. The time stamp is attached to the message in client code (so that the time reported is as close to the actual reported event as possible).

The sender field contains a <process nickname>.<clone>Id>.<class nickname> string in parentheses. The nickname is used instead of classname at the discretion of the developer only for the purpose of keeping the line length within a reasonable limit.

The message body field contains the actual message to be logged. Messages are specified by the client application. There are no restrictions on the format of a message body - the client may insert line feeds at its discretion.

The Statlog Process

The statlog process is similar to the logger process. The key difference is that the statlog process maintains the history of events so that statistics can be returned to the application. The statlog process also maintains a record of the last state for each channel. When a monitor process (such as DialWatch) accesses the statlog process, it receives the current state of all channels. The statlog process writes status to the file current.stat in the <VTT_HOME>\stat directory.
The tsmp Process

The tsmp process enables remote speech recognition. In a client-server configuration, the tsmp process provides communication between the recognition engine (on a server) and the tsm object (on a client). For more information, refer to Using ViaVoice Telephony in a Client-Server Configuration.

The tsmRouter Process

The tsmRouter process monitors and tracks the status of tsmp processes and routes connection requests to the appropriate recognition engine. The tsmRouter process exists on the server system. For more information, refer to Using ViaVoice Telephony in a Client-Server Configuration.

The ttsp Process

The ttsp process enables remote text-to-speech. In a client-server configuration, the ttsp process provides communication between the text-to-speech (on a server) and the tsm object (on a client). For more information, refer to Using ViaVoice Telephony in a Client-Server Configuration.
Using ViaVoice Telephony in a Client-Server Configuration

The ViaVoice Telephony Run Time provides both speech recognition and text-to-speech. These features may be used locally or remotely; that is, your application can perform speech recognition and/or text-to-speech on the same system on which it runs, or it can perform these functions on a system that is separate and perhaps even remote. In the latter, your application becomes the client to a speech recognition or text-to-speech server.

**NOTE:** The ViaVoice Telephony Run Time also supports using Natural Language understanding in a client-server configuration. Please refer to the IBM ViaVoiceTelephony Natural Language Tools Developer's Guide and Reference for more information.

In the ViaVoice Telephony client-server environment, each client system instantiates a tsm object for communication with a remote speech recognition server. The TsmRouter is installed on the server machine. A tsm process, along with a middleware-like ViaVoice Telephony mechanism, RPCMailbox, provides communication between the recognition engine (on a server machine) and the tsm object (on the client machine). The tsmRouter process monitors and tracks the status of the tsm processes and routes connection requests to the appropriate engine.
Remote Engine Architecture

To use a remote speech recognition or text-to-speech engine, you:

1. Install the ViaVoice Telephony Run Time on the server machine(s) and the client machine(s).
2. Define specific configuration resources on both the server(s) and the client(s) machine.
3. If desired, specify shared access to server data.

Installing the ViaVoice Telephony Run Time

The ViaVoice Telephony Run Time can be installed on a server machine or a client machine. The following section describes each type of installation.

Installing on Server Machines

When you install the ViaVoice Telephony Run Time on a server machine, be sure to click Yes on the dialog box that asks you if this will be a server installation. This installs support for the TsmRouter and other processes required on the server. For more information, refer to Installing the ViaVoice Telephony Run Time.

* An IVR can have multiple TSMs.
Installing on Client Machines

When you install the ViaVoice Telephony Run Time on a client machine, be sure to click No on the dialog that asks you if this will be a server installation. For more information, refer to Installing the ViaVoice Telephony Run Time.

Defining Client-Server Configuration Resources

Several resources need to be specified prior to configuring the client and server machines to perform remote speech recognition. The procman.cfg file defines the configuration of each process or process group in the system. The VTTDefaults file defines the resources used to configure the system.

Defining Processes (procman.cfg)

The ViaVoice Telephony Run Time consists of the following processes:

- Process Manager
- Tcl Channel
- Engine
- Logger
- Statlog
- tsmp
- tsmRouter
- ttsp

The Process Manager and Tcl Channel system processes are integral parts of and are predefined to the system. To set the environment for remote recognition or text-to-speech, define a procman.cfg file entry for the following processes:

<table>
<thead>
<tr>
<th>Process</th>
<th>procman.cfg Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Define an entry for the server.</td>
</tr>
<tr>
<td></td>
<td>Note that you can centralize the logger for a network of servers by specifying the VTTDefaults loggerHost resource as:</td>
</tr>
<tr>
<td></td>
<td><code>*:.*.loggerHost:&lt;nameOfCentralLoggerHost&gt;</code></td>
</tr>
<tr>
<td>statlog</td>
<td>Define an entry for the server.</td>
</tr>
</tbody>
</table>
Note that you can centralize statlog for a network of servers by specifying the VTTDefaults statLogHost resource as:

```
.*.*.statLogHost:<nameOfCentralStatlogHost>
```

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine</td>
<td>Define an entry for each voice channel supported. Note that each engine process is associated with a corresponding Tcl Channel (tclch) process.</td>
</tr>
<tr>
<td>tsm</td>
<td>Define an entry (as an instance of tsm) for each engine supported. A maximum of 64 tsm processes can be active at one time on one server.</td>
</tr>
<tr>
<td>tsmRouter</td>
<td>Define an entry for a server.</td>
</tr>
<tr>
<td>ttsp</td>
<td>Defines an entry for each text-to-speech channel supported</td>
</tr>
</tbody>
</table>

NOTE: When you change the number of engines and tsm processes in the procman.cfg file, you must also change the number of user and temp directories.

### Configuration for Remote Speech Reco

#### Configuring the Client VTTDefaults File for Remote Speech Reco

On the client system, you will also need a VTTDefaults file. This file defines system resources and default information which may have been configured during the product installation. Other default information is also included. The following resources must be specified in the client's VTTDefaults to perform remote speech recognition:

- routerHost
- useLocalTsmProc
- grammarPrefix

On the client system, the `routerHost` resource is used to identify the hostname(s) that are running the remote engine and telephony routers. You can specify multiple hosts as routers. The list of hosts is searched from left to right to find the engine required by an application. Refer to the resource `routerHost`. You can optionally specify how the workload is spread among all engines of an engineType. Refer to the `roundRobin` resource.

In the following example, there are two speech recognition hosts, eric and chloe, and the following entry would be placed in each client's VTTDefaults file:

```
chp.*.RPCMailbox.routerHost: "eric chloe"
```

Note that the host names specified in the list must be separated by blanks, not commas or underscores, and the list must be contained within quotes if more than one host name is specified.

You must also specify the `useLocalTsmProc` resource to indicate to the tsm object that the engine is remote and to use the telephony router to find the engine required by the application:
You also need to specify the grammarPrefix resource in the client VTTDefaults. This resource should point to the path on the server where the grammar files are found. For example, if the grammar files required for speech recognition on channel 3 are located in the d:\ViaVoicePhone\app\En_US directory on the server, the grammarPrefix resource would be set to:

chp.3.CallFlow.grammarPrefix: d:\ViaVoicePhone\app\En_US

**Client Side Configuration**

As mentioned earlier, the client portion of remote speech recognition uses a separate procman.cfg and VTTDefaults file. The following example shows a typical client side procman.cfg file. In this example, the server defines three engine connections or processes (defines two log process, and gives basic parameters for the tcl script called "appletTester").

```plaintext
Process{
    progname = logger;
    restartDelay = 1000;
}

Process{
    progname = statlog;
}

Process{
    tclname=samples/appletTester;
    cloneId=0;
    restartCount=0;
    stopDuration=100000;
    restartDelay = 1000;
}
```

**Configuring the Server VTTDefaults File for Remote Reco**

On server systems, the following two speech recognition resources must be specified:

- engineType
- useLocalTsmProc

If your application selects various engine types based on criteria, use the engineType resource to name the types. Your application then specifies the engine type to be selected as the parameter of the
The **TsmStartAppl** function. For example, the VTTDefaults entry on the server would be:

```
tsm.*.*.engineType: <engine-type>
```

You must also specify the **useLocalTsmProc** resource to indicate to the tsm object whether the engine is local or remote (0 for remote, and 1 for local):

```
chp.*.Tsmapi.useLocalTsmProc: 1
```

### Configuration Files for Remote Text-to-Speech

Before configuring your files to allow Remote Text-to-Speech (Remote TTS), there are several things to bear in mind:

1. The TTS channel and the Speech Engine channel **cannot** have the same channel ID.

2. "Channel 25" is the factory default local/remote TTS channel number (one channel higher than the last channel of a T-1 card). You can change this number if you like.

3. Before starting the remote TTS, make certain that your procman.cfg file has been properly configured and that all the parameters are set for the attributes. Have a copy of your procman.cfg file handy when configuring the remote TTS portion of your VTTDefaults file.

Remote TTS is composed of a client side and a server side. Each of these components have their own procman.cfg and VTTDefaults configuration files. The following sections show how to configure both the server and the client machines for remote speech recognition.

The following client side VTTDefaults file shows a typical client VTTDefaults file for a single channel (Channel 0). Note the highlighted areas. These items must be manually added to the **bottom** of your VTTDefaults file to indicate the name(s) of the host systems. In this example, the client recognition engine is running on the host emile2, as is the text-to-speech engine. (Of course, multiple host names could be specified for either the speech recognition host or the text-to-speech host).

```
//client reco engine
chp.*.RPCMailbox.routerHost: emile2

//client tts engine
chp.*.tts.routerHost: emile2
chp.*.tts.Language: En_US

//Add this setting if your app uses the remote tts tcl channel [tclch] [ernestine library]
chp.*.useTts: 1
```
Server Side Configuration

Several VTTDefaults resources and procman.cfg configuration parameters must be specified to support remote speech recognition. The following sections describe the resources that need to be specified on the server system.

Configuring the Server Procman.cfg File for Remote TTS

The top portion of the server side procman.cfg file contains default parameters added by the system. Verify that the channel IDs are correct (indicated in green in the following example). The bottom portion of the server side procman.cfg file contains remote tts parameters that you must add manually. In this example, "8" is the default tts channel. This number must match the default channel number shown in the server side VTTDefaults file. Of course, you can designate a different default tts channel. Whatever channel you designate, the same channel number should be specified in the server VTTDefaults file.

```plaintext
// added by default
Process{progname = logger; restartDelay = 1000; }
Process{
    progname = statlog;
}
Process{
    progname = tsmrouter;
}

// reco engine component added by default
Process{
    progname = engine;
    cloneId = 0;
    restartCount = 100;
    binDir = D:\ViaVoice\Bin\;
    options = "SPCH_DIR=D:\ViaVoice\Temp\5500@EMILE2";
    restartDelay = 1000;
    startDuration = 1000;
    quitDuration = 1000;
    stopDuration = 1000;
}

Process{
    progname = tsm;
    cloneId = 0;
    restartCount = 100;
    restartDepend = engine.0;
    restartDelay = 1000;
    startDuration = 1000;
    quitDuration = 1000;
    stopDuration = 1000;
}
```
NOTE: When you change the number of engines and tsm processes in the procman.cfg file, you must also change the number of user and temp directories.

The following graphic shows the server side VTTDefaults file for remote Text-to-Speech. In the following example, only channel 0 is activated for the recognition engine. The upper two-thirds of the file contain parameters added by the system. The bottom of the file contain parameters that you must manually configure.
Notice that some of the configuration data displayed in this graphic is obtained from the Procman.cfg file. The "useLocalTsmProc" parameter can have either of the following two settings:

1. "0" for a remote engine
2. "1" for a local engine

After configuring all necessary files, you can test the remote functionality by:

1. Start the server.
2. Enter "pm run" on the command line. (This starts the process manager.)
3. Enter "pm list" to make certain that all necessary text-to-speech engines are running.
4. Or, you can enter "tsmcon" to get a diagrammatic list of all engines.
5. Start the client side: "tclch scripts/samples/applettester.tcl -c0". (Where -c0 indicates the channel ID number you're using.)

This section describes only those resources that are essential to performing speech recognition on a remote machine. There are other speech recognition resources not discussed here that you may need to set for your application. Refer to VTTDefaults Speech Recognition Resources.

**Specifying Shared Access**

If you are building applications to run in a client-server environment, the application must be accessible to each client and server. You can do this by:

1. Building and copying the application to each system.

2. Designating a shared drive on which the application resides. Use the Windows NT drive mapping facility or Sun Microsystems' Network File System (NFS) to provide shared access to the application.
ViaVoice Telephony Resources

A ViaVoice Telephony application can be customized through the use of several system configuration files. All configuration information is stored in plain text files. Three configuration files are provided with the ViaVoice Telephony Tools:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>procman.cfg</td>
<td>Defines the behavior of the Process Manager, the ViaVoice Telephony component that controls the system.</td>
</tr>
<tr>
<td>promptsConfig</td>
<td>Defines the prompts to be used by the application.</td>
</tr>
<tr>
<td>VTTDefaults</td>
<td>Defines all of the characteristics of your application and its environment.</td>
</tr>
</tbody>
</table>

Each section describes in detail the format, content, and use of each of these configuration files.
Process Manager (procman.cfg)

The Process Manager (pm) process is responsible for managing, launching, stopping, and restarting all other telephony processes. It can be installed and run as a service or as a regular application.

The manner in which pm performs its supervisory job is defined in the configuration file procman.cfg. This file is located in the directory defined by the environment variable $(VTT_HOME).

The procname.cfg file contains entries (one for each process or process group) using the following format:

```
Process{
  <progname | tclname> = <process name>;
  group = <group name>;
  restartDepend = <list>;
  cloneId = <0..98 | range>;
  binDir = <path>;
  options = <option string>;
  restartCount = <0..N>
  restartDelay = <0..T>;
  stopDuration = <0..T>;
  startDuration = <0..T>;
}
```

Each process is defined in a separate process{} block. Within each process block is a series of tag-value pairs that define a particular process. There are also tags that can be specified outside the scope of a process block and have global scope. A description of these tag-value pair follows.

Note that all tags are case sensitive. Their associated values depend on usage. White space characters (blank, tab, carriage return and line feed) can be placed deliberately according to following rule:

```
{W}tag{W}={W}value{W};
```

where W can be \t, \n, or \r.

For example, the following commands are legal:

```
programe = myproc;

// Used to mark the beginning and end of a string
options = -a -b ; // maps into ^-a -b^, ^

// The following maps in ^ -a -b ^
options = " -a -b " ;

// The following maps into ^"ab"^
options = "ab" ;

Notes:

- White characters are permitted in value fields.
- Enclosure within quotes is only needed when the value field includes quotes.
- Specify all timers in milliseconds.
- // marks indicate a comment to the end of the line except when the // appears in the middle of a value field.
- A process list is a sequence of tokens containing group name(s) and/or process name(s).

**Process Block Tags**

The following tags are used within the scope of a process block:

**binDir**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>binDir</td>
<td>binDir = &lt;path&gt;;</td>
<td>resource <em>.</em>.*.binDir</td>
</tr>
</tbody>
</table>

Remarks:

The Process Manager builds a fully-qualified path name for each process. Normally, this path name is derived from $VTT_HOME; however, the binDir tag, if present, provides an alternative location.

**cloneld**

The cloneld tag is used to distinguish multiple instances of the same executable. It maps into a string ("-c <cloneld>") that is passed to each process.

If any Process{} block defines a cloneld using a range (a mixture of '-' and ',' is permitted, such as cloneld = 0-2,5;), the cloneld refers to a group of processes. By default, all of the processes in the group are started and stopped simultaneously (that is, the start/stop command is issued to all of the processes and only one timer applies).

**group**

The group tag is an arbitrarily assigned name. It allows processes to be referred to with different names when defining dependency lists or start/stop sequences.

**options**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>options = &lt;option string&gt;;</td>
<td>-c &lt;cloneld&gt;</td>
</tr>
</tbody>
</table>

Remarks:
The value assigned to the **options** tag is passed as is to the started process. If the options tag-value pair are present in the configuration file, the "-c cloneld" must be included in the value (The Process Manager does not supply it).

**proname or tclname**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>proname or tclname</td>
<td>proname = &lt;process name&gt;; or tclname = &lt;Tcl script name&gt;;</td>
<td>No default</td>
</tr>
</tbody>
</table>

Remarks:

The value following this tag should specify just the process name (for example, tclchp) or Tcl script name (for example, appletTester). If the full path is specified, it is stripped. For **proname**, pm retrieves binaries from the directory specified by the `binDir` resource and it appends `.exe` to the proname. For **tclname**, the resource-defined tclShell specified by the `tclDir` resource is executed and `.tcl` is appended to the tclname.

**quitDuration**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>quitDuration</td>
<td>quitDuration = &lt;0..t&gt;;</td>
<td>2000</td>
</tr>
</tbody>
</table>

Remarks:

The **quitDuration** tag defines how long pm waits for a process to exit in response to a quit command (This is a graceful exit and is executed immediately. For channel processes, all active calls are abandoned). If a process being stopped does not exit within the quitDuration time, it is forcefully terminated.

**restartCount**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>restartCount</td>
<td>restartCount = &lt;0..n&gt;;</td>
<td>10</td>
</tr>
</tbody>
</table>

Remarks:

The **restartCount** tag defines how many times pm tries to restart a process that has abnormally terminated. If restartCount is set to 0, the aborting process is not restarted automatically. Restarting pm or restarting the terminated process manually resets the count of restarts.

**restartDelay**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>restartDelay</td>
<td>restartDelay = &lt;0..t&gt;;</td>
<td>12000</td>
</tr>
</tbody>
</table>

Remarks:
The **restartDelay** tag defines how long *pm* should wait after a process exits before starting it again. When a group of processes is restarted, the longest timer from the process list is used. The value is specified in milliseconds.

The default value is 12000 milliseconds. If the speech recognition engine is launched from the tclchp process, it has enough time to notice when the tclchp exits. The engine must exit before the tclchp process is restarted.

**restartDepend**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>restartDepend</td>
<td>restartDepend = &lt;list&gt;;</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

**Remarks:**

The **restartDepend** tag defines the list of processes and/or groups which are to be stopped and started when a process with this entry is restarted (either manually or as a result of an abort). For example, given the following entries:

```
Process{proname = a; restartDepend = b c d.0;}
Process{proname = b; cloneId = 0-2;}
Process{proname = c;}
Process{proname = d; cloneId = 0,2;}
```

If process *a* restarts, processes *b.0, b.1, and b.2* are stopped, followed by process *c.0*, and finally *d.0*. Starting process *a* proceeds in the reverse order.

**startDuration**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>startDuration</td>
<td>startDuration = &lt;0..t&gt;;</td>
<td>2000</td>
</tr>
</tbody>
</table>

**Remarks:**

The **startDuration** tag defines how long *pm* should wait for the notification from a process that it has completed initialization. The *pm* process expects each process to notify it of successful initialization. Only after a process notifies *pm* of its completion can a consecutive process (or process group) be started. If the timer expires, a warning is issued and *pm* proceeds with launching other processes.

**stopDuration**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>stopDuration</td>
<td>stopDuration = &lt;0..t&gt;;</td>
<td>90000</td>
</tr>
</tbody>
</table>

**Remarks:**

The **stopDuration** tag defines how long *pm* waits for a process that was stopped with a stop command. Channel processes should wait for active calls to complete before issuing a forceful exit. The value is
specified in milliseconds. The default is a generous 90 seconds. It should be decreased for processes not
involved in call processing.

Global Tags

In addition to the tags used within process blocks, the following tags are permitted outside the scope of a
process block. As such, these tags specify overall system behavior rather than process behavior.

**AutoStart**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoStart</td>
<td>AutoStart = &lt;on</td>
<td>off</td>
</tr>
</tbody>
</table>

Remarks:

The **AutoStart** tag tells **pm** whether all processes are to be started at startup (or at service installation).
The value is a boolean.

**StartSequence**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartSequence</td>
<td>StartSequence = &lt;process name&gt;;</td>
<td>No default</td>
</tr>
</tbody>
</table>

Remarks:

The **StartSequence** tag tells **pm** the order in which to start processes. Normally, processes are started
by **pm** in the sequence in which they appear in the configuration file.

**StopSequence**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>StopSequence</td>
<td>StopSequence = &lt;process name...&gt;;</td>
<td>No default</td>
</tr>
</tbody>
</table>

Remarks:

The **StopSequence** tag tells **pm** the order in which to stop processes. Normally, process shutdown is
performed in the reverse order from which the processes were started.

The value is specified as a process list. The only requirement is that the value list defines all processes.
List elements are either a group name or a `<progname>.<cloneId>`. 
Examples

In the following example, three processes are defined to pm with cloneIds (p1.0, p1.1, and p1.2). These processes are started in parallel. Each process is stopped one-by-one.

```
Process{proname = p1; cloneId = 0;}
Process{proname = p1; cloneId = 1;}
Process{proname = p1; cloneId = 2;}
StartSequence = p1;
StopSequence = p1.0 p1.1 p1.2;
```

In this example, four process are defined to pm with cloneIds (p1.0, p1.1, p1.2, and p2.0). A process group (p1group) is also defined which contains the p1.0, p1.1, and p1.2 processes. Each of the p1group processes are started in parallel, and then pm starts process p2.0. Process shutdown is performed in reverse order.

```
Process {proname = p1; group p1group;}
Process {proname = p1; cloneId = 1; group p1group;}
Process {proname = p1; cloneId = 2; group p1group;}
Process {proname = p2;}
StartSequence = p1group;
StopSequence = p2.0 p1group;
```

NOTE: When you change the number of engines and tsmp processes in the procman.cfg file, you must also change the number of user and temp directories.
Prompts (promptsConfig)

Most persons have used voice response units (VRUs). Voice prompts are an integral part of a VRU system. They are provided by a phone application during a call to help guide the caller through the call and to confirm the caller’s selections. Examples of voice prompts are “Enter your account number followed by a pound sign” and “You chose to transfer five hundred dollars and zero cents from account one to account two. Press the pound sign if this is correct. Press star to go back.”

Voice prompts can be generated as text-to-speech, or they can be prerecorded audio waveforms.

The voice prompts that are available through ViaVoice Telephony applications are customizable. They are defined in the promptsConfig file. (The directory in which the promptsConfig file is stored is specified as a VTTDefaults resource). The promptsConfig file is a text file that uses the following format to describe individual voice prompts:

```
<prompt name>
  [text=<text of the prompt>;
  [preload=(yes|YES|Yes|1|NO|No|no|0);
  [filename=<full path>;
  }
```

A prompt is defined using a prompt {} block. Within the prompt block are a series of tag-value pairs. The only mandatory item is the prompt name. All of the tags within the process block are optional.

Indentation shown in the sample above is not mandatory; white space characters (‘ ’, ‘\t’, ‘\n’) can be placed anywhere. The “//” comment converts all characters until the first ‘\n’ into white space.

**prompt name**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>prompt name</td>
<td>[0-9_A-Za-z]+{}</td>
<td>No default</td>
</tr>
</tbody>
</table>

**Remarks:**

The prompt name is an application-defined string. It is used as a key to select a prompt for the play command in a TCL script.

**text**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>text=&lt;text of the prompt&gt;;</td>
<td>&lt;prompt name&gt;</td>
</tr>
</tbody>
</table>

**Remarks:**

The text tag specifies the text of the prompt. This text is used by the text-to-speech engine. If no prompt text is specified, the default value is the prompt name.
**preload**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>preload</td>
<td>preload=(yes</td>
<td>YES</td>
</tr>
</tbody>
</table>

Remarks:

The `preload` flag indicates whether the prompt should be loaded into shared memory or loaded from disk. For very long prompts that are played infrequently, it may be sufficient to load the prompt from disk only when it is needed. The default value is `yes`.

**filename**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Format</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>filename = &lt;full path&gt;;</td>
<td>VTTDefaults</td>
</tr>
</tbody>
</table>

Remarks:

By default, all prompts are loaded from the directory defined by a resource in the VTTDefaults file. named `<prompt name>.ulw`; this default location can be overridden with fully qualified filename assigned to `filename` tag.

**Example of a promptsConfig File**

An example promptsConfig file is provided below. For this example, the resource `*.`.Prompts.promptsDir is set to `d:\vtt\prompts`:

```plaintext
// prompt name = please
// text = "please"
// preload = yes
// filename = d:\vtt\prompts\please.ulw
please{}

// prompt name = thankYou
// text = "thank you"
// preload = yes
// filename = d:\vtt\prompts\thankYou.ulw
thankYou{text= thank you;} // text is "thank you"

// prompt name = welcome
// text = "welcome to I B M dialer"
// preload = yes
// filename = d:\vtt\prompts\welcome.ulw
welcome{text="welcome to I B M dialer";}
```
If there are multiple processes using the same promptsConfig file, the file is marked as read-only by the first process which obtains the lock on shared memory. Other processes attach the shared memory segment and access the data stored in it directly.

It is safe to edit the file at any time. The changes take effect only when all processes are stopped and started again.
VTTDefaults Resources Overview

The ViaVoice Telephony Tools define many resources through which the system is configured. Each resource has a name, a default scope, and a default value associated with it. The scope and value can be overridden by placing new resource strings in the VTTDefaults file located in the $VTT_HOME directory. Boolean default values 1, on, and true are equivalent as are 0, off, and false.

The resources contained in the VTTDefaults file can be categorized as follows:

- **Directories**
- **Logging**
- **Process Manager**
- **Prompts**
- **Speech Recognition**
- **TCL Scripts**
- **Telephony**
- **Text-to-Speech**

Each resource in the VTTDefaults file is defined using the following format:

```
<process>.<cloneId>.<classname>.<resource name>: ""<resource value>""[// comment]
```

where:

- **process** is the name of a ViaVoice Telephony Tools process.
- **cloneId** is an integer that identifies the unique instance of the process.
- **classname** is the name of the ViaVoice Telephony C++ class or Tcl script that uses the resource.
- **resource name** is the unique name of the resource.
- **resource value** is the default value of the resource.

Process, cloneId, andclassname are called resource scope qualifiers. Specification of a qualifier defines the scope to which the resource value applies. Following are the rules for coding resource scope.

- Quotes are necessary if the resource value string contains any white space (blank or tab).
- // -style comments are allowed.
- Any of the resource scope qualifiers can be omitted or replaced with the wildcard character ‘*’ (except for the case where classname and processname are the same, then either *.classname or process.*.name must be used).

The default resources are read at the end of Process Manager initialization. They do not override values supplied in the VTTDefaults file. An attempt to acquire an undefined resource causes an assertion exit (a
resource can be defined in a file without a hard-coded default value but, if an application tries to use the resource, it must be present in the file used by the system).

One process—tclchp— uses a <process> scope qualifier which differs from its process name: chp.

Note: If the default scope is narrower than the scope supplied in VTTdefaults, the default scope prevails.

The resources are initialized only at process startup. If the VTTdefaults file is modified, the changes become effective in the affected process(es) only after restart.

In the following example, the system has 2 channel processes (tclchp) and the following resource file content:

```plaintext
// Defines the prompts directory for the tclchp.0 process
chp.0.Prompts.promptsDir: /home/app/chp0

// Defines the prompts directory for all tclchp processes
chp.*.Prompts.promptsDir: /home/app/prompts

// Defines the prompts for any tclchp process
chp.*.*.promptsDir: /home/mydir

// Sets the global debug level to "warn"
*.*.*.debugLevel: warn

// Overrides the global debug level for all tclchp processes
chp.*.*.*.debugLevel: info
```
VTTDefaults: Directories

The ViaVoice Telephony Tools suggest a layout of directories at installation, but it does not enforce it. It can be overridden by the application through various system resources. If a relative path is defined, it starts from $VTT_HOME. This is necessary because each ViaVoice telephony process calls chdir($VTT_HOME) after startup.

**binDir**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>binDir</td>
<td><em>::</em></td>
<td>bin\</td>
</tr>
</tbody>
</table>

Remarks:
The directory in which all ViaVoice telephony binaries are stored. This directory is prepended to each process name before the Process Manager starts it. It is also prepended to the Process Manager itself when it is started as a daemon.

**checkDirList**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkDirList</td>
<td><em>::</em></td>
<td>binDir logDir statDir</td>
</tr>
</tbody>
</table>

Remarks:
Checks the list of directories used by the Process Manager to ensure that everything is in order before the system starts.

**dirsToClean**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirsToClean</td>
<td><em>::</em>.DiskManager</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies from which directories Process Manager should remove files when the amount of free disk space is below a specified threshold. It also specifies the age of the files to be removed. Note that if a relative path is specified, it is appended to <VTT_HOME>.

The dirsToClean resource is specified as a space-delimited string of values, where each value is comprised of a directory name, the age (in days) of the files to be removed, and a recursive flag (0 or 1, which indicates whether subdirectories will also be cleaned).
The following example specifies that files 30 days or older are to be removed from the log directory, 15 days or older from the pcm directory, and 15 days or older from the web directory and all of its subdirectories:

```
*.DiskManager.dirsToClean: log 30 0 pcm 15 0 web 15 1
```

diskRescanInterval

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>diskRescanInterval</td>
<td>*.DiskManager</td>
<td>1800</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the time (in seconds) when the Process Manager will rescan the disk for free space. This resource is used when, even after Process Manager has cleaned up the system, the amount of free disk space is still below the threshold at which aged files are removed. By default, it is set to 1800 seconds, or 30 minutes.

The following example sets the diskRescanInterval to 7200 seconds (2 hours):

```
*.DiskManager.diskRescanInterval: 7200
```

diskScanSchedule

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>diskScanSchedule</td>
<td>*.DiskManager</td>
<td>0100</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the times at which Process Manager will scan the disk for free space. Times are specified in hhmm format. The default is 0100, or 1:00 a.m.

The following example schedules Process Manager to scan the disk at 1:00 a.m. and 4:00 a.m.:

```
*.DiskManager.diskScanSchedule: 0100 0400
```

diskScanThresholds

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>diskScanThresholds</td>
<td>*.DiskManager</td>
<td>10 5</td>
</tr>
</tbody>
</table>

Remarks:

This resource defines the threshold of free disk space that triggers warning messages or cleanup. The first value specifies the warning threshold and the second value specifies the cleanup threshold. The default is set to 10% and 5%, respectively. This means that Process Manager will remove aged files
when free disk space is less than 5% but issue warnings when disk space is greater than 5% but less than 10%.

The follow example schedules Process Manager to issue warnings when free disk space is greater than 10% and less than 20%, and to remove aged files when disk space is less than 10%:

```
*.DiskManager.diskScanThresholds: 20 10
```

### dlgDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlgDir</td>
<td>*.</td>
<td></td>
</tr>
</tbody>
</table>

Remarks:

If the system is configured for Dialogic hardware, the dlgDir entry specifies the directory in which dynamically loaded Dialogic DLLs are found. If the default value is used (""), Windows NT rules for finding DLLs apply.

### eciDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>eciDir</td>
<td>*.Dll</td>
<td>&lt;VTT_HOME&gt;	tts</td>
</tr>
</tbody>
</table>

Remarks:

If the system is configured to use text-to-speech, the eciDir resource specifies the directory in which the text-to-speech engine binaries are located. By default, this is set to <VTT_HOME>	tts.

### filesToClean

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>filesToClean</td>
<td>*.DiskManager</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the files that Process Manager should remove when the amount of free disk space is below a specified threshold. Note that if a relative path is specified in dirsToClean, it is appended to <VTT_HOME>. The files are removed from the directories specified in the dirsToClean resource.

A space-delimited list of file extensions can be specified. For example, the following statement specifies the removal of *.log, *.pcm, and *.html files:

```
*.DiskManager.filesToClean: .log .pcm .html
```

61
### logDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logDir</td>
<td><em>.</em>.*</td>
<td>log\</td>
</tr>
</tbody>
</table>

**Remarks:**
The directory where the logger process writes data. The file created by logger is `current.log`. When its size exceeds a resource-defined limit, it is renamed to `MMDDYYHHMM.log` and `current.log` is started anew.

### logPcmDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logPcmDir</td>
<td>chp.<em>.</em></td>
<td>pcm\</td>
</tr>
</tbody>
</table>

**Remarks:**
The Resource Name in the VTTDefaults file that indicates where speech recordings are stored (`chp.*.*.logPcmDir: pcm\`). REC (.rec) and mulaw (.ulw) files are produced and stored in this directory.

If the logPcm resource is set to on, then *.ulw files will be produced and stored in the directory. If the logPcm resource is set to off (the default state), no *.ulw files will be produced and stored in the directory.

If the logCalls resource is set to on, then *.rec files are created and stored in this directory. If the logCalls resource is set to off, then no *.rec files are created and stored in this directory.

If both logPcm and logCalls are set to off, no speech files will be created and stored in this directory. If both logPcm and logCalls are set to on, both speech file types (.ulw nd .rec) will be created and stored in this directory.

Note that it is the application developer’s responsibility to guarantee uniqueness of the filenames.

### namePromptsDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>namePromptsDir</td>
<td><em>.</em>.*</td>
<td>prompts\LI(LL)\alpha\</td>
</tr>
</tbody>
</table>

**Remarks:**
The directory where prerecorded name prompts are located. Each string is passed as a name to play and is tokenized. For each token, a path

`<namePromptsDir><first letter of a token><last letter of a token><token>.ulw`  
is used to retrieve the recording. All recordings are then concatenated and played.
Note that LI_LL can be one of the following:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

### nmsDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsDir</td>
<td>chp.<em>.</em></td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

**Remarks:**

If the system is configured for Natural Microsystems hardware, the nmsDir entry specifies the directory in which dynamically loaded Natural Microsystem DLLs are found. If the default value is used (""), Windows NT rules for finding DLLs apply.

### promptsDir

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>promptsDir</td>
<td><em>.</em>.*</td>
<td>prompts\LI_LL\</td>
</tr>
</tbody>
</table>

**Remarks:**

The directory where the promptsConfig file is located. It also specifies the default location of prompts files. Only prompts defined in the promptsConfig file are visible to channel processes. Other files (if necessary) must be accessed in Tcl scripts or as a blob. The promptsConfig resource is set to "prompts\LI_LL\applets prompts\LI_LL\ndsampel" by the installation program, where LI_LL can be one of the following:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

Note: Whenever the promptsDir resource is set to different values for different channels, the promptsName resource must also be set differently.
**recordDir**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordDir</td>
<td><em>.</em>.*</td>
<td>users\</td>
</tr>
</tbody>
</table>

**Remarks:**
The directory where speech recognition data is written. Files are named as RECFbNNnn.vox, where NN is the adapter ID and nn is the channel ID.

**statDir**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>statDir</td>
<td><em>.</em>.*</td>
<td>stat\</td>
</tr>
</tbody>
</table>

**Remarks:**
The directory where the statlog process writes data. The file created by statlog is current.stat. When its size exceeds a resource-defined limit, it is renamed to MMDDYYHHMM.stat and current.stat is started anew.

**tclShell**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tclShell</td>
<td><em>.</em>.*</td>
<td>tclch.exe</td>
</tr>
</tbody>
</table>

**Remarks:**
The name of the Tcl shell program used by the Process Manager when launching Tcl processes. This value is used in combination with the tclDir resource.

**tclDir**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tclDir</td>
<td><em>.</em>.*</td>
<td>scripts</td>
</tr>
</tbody>
</table>

**Remarks:**
The directory used by the Process Manager to build the path to the tclShell. This resource is used to launch Tcl scripts.
**tmpDir**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmpDir</td>
<td><em>.</em>.*</td>
<td>tmp\</td>
</tr>
</tbody>
</table>

**Remarks:**

A temporary directory used by the Process Manager at startup. The last component of the path must be either `tmp` or `temp` (this is needed to prevent accidental cleanup of directories if the resource value is set incorrectly).
VTTDefaults: Logging

These resources are used by the logger process and maintained in the VTTDefaults file:

**debugLevel**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>debugLevel</td>
<td><em>:</em>:*</td>
<td>warn</td>
</tr>
</tbody>
</table>

Remarks:

The debug level that is set whenever an object registers with the logger process (usually at startup or at the time the first instance is constructed). The valid values are: trace (trc), debug (dbg), info (inf), advise (adv), warning (warn), error (err), and fatal (fat). It is recommended that you use the lowest debug level (trace) only selectively (not in a global scope), since the volume of generated messages with global trace may have an adverse effect on system performance.

**fileSizeLimit**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileSizeLimit</td>
<td>logger.*.Logger</td>
<td>1000000</td>
</tr>
</tbody>
</table>

Remarks:

The statlog process writes arriving messages to a file `<statDir>/current.stat`. When this file size exceeds `<fileSizeLimit>`, it is renamed to MMDDhhmmss.stat.

**loggerHost**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>loggerHost</td>
<td><em>:</em>:*</td>
<td>localhost</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the name of the host process that runs the logger process.

**loggerProtocol**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>loggerProtocol</td>
<td><em>:</em>:*</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>
Remarks:
This resource specifies the networking protocol to be used if the logger process is running on a remote host. The valid values are tcp and udp. Null (""), the default, indicates that the logger process is running on a local host.

### logLevelsName

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logLevelsName</td>
<td><em>.</em>.*</td>
<td>logLevels</td>
</tr>
</tbody>
</table>

Remarks:
The shared memory segment used for a particular debug level has the name "/Shmem/<logLevelsName>" and any ‘\’ character in <logShmName> will be replaced with ‘_’.

### logMilliseconds

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logMilliseconds</td>
<td><em>.</em>.*</td>
<td>off</td>
</tr>
</tbody>
</table>

Remarks:
If this resource is set to on, the logger process includes milliseconds in its time stamps. Note that the time stamp is always created in the client so it accurately describes the timing of the event being logged.

### logTimeFormat

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logTimeFormat</td>
<td><em>.</em>.*</td>
<td>%m-%d %H:%M:%S</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the format in which the logger process should write the time stamp. The value should be specified as an strftime compliant string.

### maxLogObjects

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxLogObjects</td>
<td><em>.</em>.*</td>
<td>1000</td>
</tr>
</tbody>
</table>
 Remarks:

This resource is used to size the shared memory segment maintained by the logger process. The debug levels for all registered objects are stored in this shared memory segment. Each entry is an integer, so the effective shared memory segment size is maxLogObjects*4.

repeatLimit

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeatLimit</td>
<td>logger.*.Logger</td>
<td>40</td>
</tr>
</tbody>
</table>

 Remarks:

The logger process buffers each message received. If consecutive messages have identical content, only every <repeatLimit> message is logged, along with repeat count information. Use of repeatLimit prevents clogging log files by a runaway process.

rmLogsOnStart

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmLogsOnStart</td>
<td>logger.*.Logger</td>
<td>false</td>
</tr>
</tbody>
</table>

 Remarks:

If the value of this resource is set to true, all files in the <logDir> will be removed when the logger process starts.

statLogHost

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>statLogHost</td>
<td>localhost</td>
<td>localhost</td>
</tr>
</tbody>
</table>

 Remarks:

This resource specifies the name of the host that runs the statLog server.

statLogProtocol

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>statLogProtocol</td>
<td>localhost</td>
<td>localhost</td>
</tr>
</tbody>
</table>
This resource specifies the networking protocol to be used if the statlog process is running on a remote host. The valid values are tcp and udp. Null (""), the default, indicates that the statlog process is running on a local host.

### tailSize

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tailSize</td>
<td><em>.</em>.LoggerLoop</td>
<td>50</td>
</tr>
</tbody>
</table>

**Remarks:**

The logger process maintains a circular buffer of received messages. This resource defines the size of the circular buffer (in number of messages). Monitoring utilities such as **DialWatch** receive its content at startup, so recent activity can be viewed instantly.
VTTDefaults: Process Manager

The following section describes the resources in VTTDefaults which configure the Process Manager process:

**oemServiceName**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>oemServiceName</td>
<td><em>.</em>.*</td>
<td>See Remarks</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the processes that must be started before the Process Manager process when it is started as a service. For Dialogic adapters, it should be set to "Dialogic | RpcSs." For Natural Microsystems, the resource should be set to "Agmon | RpcSs."

**postMortemDebug**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>postMortemDebug</td>
<td><em>.</em>.*</td>
<td>off</td>
</tr>
</tbody>
</table>

Remarks:

When the postMortemDebug resource is set to on, a pop-up dialog is displayed when a process controlled by Process Manager terminates on exception. The dialog contains pertinent debug information. This resource applies to all processes controlled by Process Manager (in other words, setting the scope to chp.* has no impact on tclch behavior as all processes inherit the behavior from Process Manager).

**useConsole**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>useConsole</td>
<td><em>.</em>.*</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies whether logger output is displayed on the screen when processes are launched from the command line. The default of 0 means that logger output will not be displayed on the screen.

This resource is used in conjunction with procman.cfg. The console=yes option must be specified in procman.cfg and the useConsole resource set to 1 for logger output to displayed in a window.
VTTDefaults: Prompts

The following section describes the prompt resources supported in the VTTDefaults file:

**maxNumberOfPrompts**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxNumberOfPrompts</td>
<td>*. *.Prompts</td>
<td>3000</td>
</tr>
</tbody>
</table>

Remarks:

This resource defines the maximum number of prompts which may be specified in the promptsConfig file.

**promptsName**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>promptsName</td>
<td>*. *.Prompts</td>
<td>prompts</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the name of the shared memory segment in which prompts are stored. The default value for promptsName that is provided by installation is "prompts".

Note: Whenever the promptsDir resource is set to different values for different channels, the promptsName resource must also be set differently.
VTTDefaults: Speech Recognition

The following section describes the speech recognition resources supported in the VTTDefaults file. **Note:** All Tsmapi and Audio resources have scope for chp and tsmp processes.

### appPath

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>appPath</td>
<td><em>.</em>.Tsmapi</td>
<td>&lt; empty &gt;</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource controls the base component of the application path. For example, a U.S. English application could specify a grammar file as pwd.1.fsg and set this resource to d:\<VTT_HOME>\app\En_US\pwd\grammars. The Tsm object combines these two components and creates the whole path for the grammar file. Note that if a grammar file is specified using either a colon in the path or a leading '/ ', the Tsm object assumes that the application is specifying the complete path, and the appPath resource is not used.

### audioRecordBuffers

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>audioRecordBuffers</td>
<td><em>.</em>.Audio</td>
<td>300</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies a recording duration of 20 seconds without wrap-around (300 blocks of 551 bytes each). Normally, wrap-around is handled properly; it is only an issue when using barge-in for longer than this period and then subsequently dumping the buffers for data collection purposes. In that case, only the last 20 seconds of data will be available.

### DMThresholdNoConfirm

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMThresholdNoConfirm</td>
<td><em>.</em>.Tsmapi</td>
<td>14</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource controls the threshold above which the Tsm object declares an utterance not to require confirmation. It is used to detect utterances where the recognizer is confident of the result. The useful range of DMThresholdNoConfirm is from 10 to 30. A default of 14 is used for telephony; however, developers can lower the threshold if they feel that recognition will be confident and users don't need to be asked to confirm what they said. The number should be increased if the system is accepting too many recognition errors.
DMThresholdReject

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMThresholdReject</td>
<td><em>.</em>.Tsmapi</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:
This resource controls the threshold below which the Tsm object declares an utterance to be rejected. This resource is used to reject utterances where the recognizer is unsure of the result. The useful range of DMThresholdReject is from -10 to 10. Selecting a higher threshold requires the engine be more sure of the result and should result in more rejections.

enableNbest

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableNbest</td>
<td><em>.</em>.Tsmapi</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:
This resource controls whether the engine returns multiple alternatives (the n best) for recognized text. By default, the engine only returns one alternative. Set enableNbest to 1 to return multiple (up to 10) alternatives.

enableSilenceDetection

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableSilenceDetection</td>
<td><em>.</em>.Audio</td>
<td>1</td>
</tr>
</tbody>
</table>

Remarks:
This resource controls whether silence is detected within an utterance and reported back to the application. By default, silence information is reported back to the application.

dimeType

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimeType</td>
<td><em>.</em>.*</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the engine type of this session. For example if you specify the following in
VTTDefaults:

tsm.0.*.engineType: SBU
tsm.1.*.engineType: USA
tsm.2.*.engineType: USA

Channel 0 will be selected by applications that call the function TsmStartAppl() with the appName set to "SBU".

enrollId

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enrollId</td>
<td>*.Tsmapi</td>
<td>tel14</td>
</tr>
</tbody>
</table>

Remarks:

This resource controls the enrollment ID that is used with the ViaVoice speech recognition engine. The following enrollment IDs are provided with the ViaVoice Telephony Run Time:

- tel14 - U.S. English General Purpose Telephony
- tel14I - U.S. English Cellular
- tel15 - U.S. English Fast General Telephony
- teluk0 - U.K. English General Purpose Telephony
- telfr0 - French General Purpose Telephony
- telgr0 - German General Purpose Telephony

The default enrollment ID for each language is the General Purpose Telephony enrollment ID for that language. For example, in U.S. English, the default is tel14.

grammarPrefix

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>grammarPrefix</td>
<td>*.CallFlow</td>
<td>None</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the path where the grammar files are found.

In a client-server environment, the grammarPrefix resource is set in the client's VTTDefaults file. It should be set to the path on the server where the files are found.
### initialMNRFile

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialMNRFile</td>
<td><em>.</em>.Tsmapi</td>
<td>None</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies that the cepstral mean normalization file will used. This file is generated by IBM for custom applications. The general file provided with the engine is sufficient for most situations.

### initialSilenceDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialSilenceDuration</td>
<td><em>.</em>.Audio</td>
<td>4000</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the amount of time (in milliseconds) that the speech engine waits before declaring initial silence timeout. Initial silence timeout is declared when the user has not spoken anything during the first milliseconds of audio recording. Note that each buffer represents 69 milliseconds of speech data. Use a very large number here to disable detecting initial silence timeout.

### Language

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td><em>.</em>.Tsmapi</td>
<td>En_US</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the language of the speech recognition engine. It can be one of the following:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

### maxAllocatedTime
maxAllocatedTime | *.Tsmapi | 600

Remarks:
This resource specifies the amount of time (in seconds) that the TsmRouter waits before freeing an inactive engine. Every engine has an attribute which specifies the last time it was allocated. While it is inactive, the engine is still connected to the client. For the period up to maxAllocatedTime, no other clients can connect to the engine. When maxAllocatedTime has been reached, the TsmRouter frees the engine and it is available for a client connection.

The default time duration is 10 minutes.

minSilenceDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minSilenceDuration</td>
<td>*.Audio</td>
<td>700</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the amount of time (in milliseconds) that the speech engine waits before declaring the end of an utterance. The default time duration is, therefore, 0.7 seconds.

The end of an utterance is declared when the user has not spoken anything during the last n buffers of audio recording. Each buffer represents 69 milliseconds of speech data.

minSpeechDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minSpeechDuration</td>
<td>*.Audio</td>
<td>150</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the amount of time (in milliseconds) required for the speech engine to start searching for the end of an utterance. The default value is .15 seconds.

If the user has not spoken anything for the last n buffers, the speech engine will start searching for the end of an utterance. Each buffer represents 69 milliseconds of speech data.

This resource should only be changed if the specific application requires some adjustment. Lowering the value makes the application less sensitive to speech, while increasing the value is useful if the application knows there will be a significant amount of speech before the user stops speaking.

minSpeechLevel

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**minSpeechLevel**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minSpeechLevel</td>
<td>*. *.Audio</td>
<td>150</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the energy computed for the speech signal. The energy is computed using the absolute value of the signal. The useful range of this resource is between 72-1800. This value should only be changed if there is a specific application problem detecting silence.

**minSpeechOverBeepLevel**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minSpeechOverBeepLevel</td>
<td>*. *.Audio</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the energy computed for the speech signal as the sum of the absolute values on the first frame. If it is exceeded and the resource speechOverBeep is set to true, the Audio object declares TSM_SPEECH_OVER_BEEP.

**portAtServer**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>portAtServer</td>
<td>chp.0N.Tsmapi.portAtServer or chp.NN.Tsmapi.portAtServer</td>
<td>550N@localhost or 55NN@localhost</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the port number at the host that is used for audio communication. Normally, this should NOT be modified after installation; however, if there is a conflict, a different port number can be specified using this resource. 0N through NN represents the channel number which can have the value 0 through 23.

**roundRobin**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>roundRobin</td>
<td>tsmRouter. *. Router</td>
<td>1</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies how to search for the next engine. The default setting specifies that the load is spread among all the engines of the specified engineType, searching round robin among those engines. Specifying 0 for this parameter sets the search to linear mode.
Proper use of this resource is related to the amount of physical memory (RAM) available to the application. For those systems where the amount of physical memory is greater than the total memory required by the application to support all of the channels (engines) in the system, either technique is effective. However, for those systems where the amount of physical memory available to the application is less than required (and thus all of the engine code cannot reside in RAM nd may be cached to disk), the linear search mode is preferable.

As a rough estimate, each speech recognition channel requires 10MB of memory. Of course, all pertinent processes must be considered when calculating the total memory required to support all channels of a telephony application, including the operating system, the client application, ViaVoice Telephony, and data.

### routerHost

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>routerHost</td>
<td><em>.</em>.RPCMailbox</td>
<td>localhost</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource defines which host(s) have telephony routers running. It allows a Tsm object to automatically find an engine of the desired type. The host list is a space-delimited list of host names (enclosed within quotes if more than one host name is specified). For example, if you specify the following in the VTTDefaults file:

```shell
chp.*.RPCMailbox.routerHost: "barnett vds05 vds08"
```

The Tsm class will search the tsmRouter on barnett followed by vds05 and vds08 to find the engine required for the application.

### saveAudio

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>saveAudio</td>
<td><em>.</em>.Tsmapi</td>
<td>0</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource controls whether the engine saves the PCM data it receives. This is useful in debugging, and should normally be turned off. Per utterance control is also provided using the TsmStartReco function.

### speechOverBeep

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>speechOverBeep</td>
<td>chp.*.Audio</td>
<td>1</td>
</tr>
</tbody>
</table>
Remarks:

This resource is a boolean value which controls whether to detect speech on the first frame of recording. This resource is used to determine if the user might have spoken over the beep. For barge-in applications this parameter MUST be set to 0 since there is no beep played. However, this resource can be set per utterance and then used for barge-in.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskId</td>
<td>Tsmapi</td>
<td>teletl14</td>
</tr>
</tbody>
</table>

Remarks:

This resource controls the speech recognition task that is used by the ViaVoice engine. There are several task IDs provided with the ViaVoice Telephony Run Time:

- teletl14 - U.S. English General Purpose Telephony
- telet14l - U.S. English Cellular
- teletl15 - U.S. English Fast General Telephony
- digits - U.S. English Telephony Digits
- telv1uk - U.K. English General Purpose Telephony
- telfr - French General Purpose Telephony
- telgr0 - German General Purpose Telephony

The default task ID in each language is the General Purpose Telephony task for that language. For example, for U.S. English, it is teletl14. The General Purpose Telephony task is useful for most telephony applications.

For digit-only applications (for example, those that process credit card numbers), the Digits task can be used. In conjunction to changing the taskId resource to “digits,” use the digComp applet grammar for digit recognition. The digComp applet grammar is designed for digit recognition precision needed by digit-only applications.

To use this resource for a channel process, set the taskId resource for the scope chp.*.Tsmapi. To use this resource for a remote engine process, set the taskId resource for the scope tsmp.*.Tsmapi.

The taskId resource can be set, per recognition needs, for an individual channel or line by narrowing the scope of the resource. For example, consider a system having three channels with the following resource settings:

tspm.0.taskId: digits
tspm.1.taskId: digits
tspm.2.taskId: teletl14
tsmp.0.*.engineType: DIGIT
tsmp.1.*.engineType: DIGIT
tsmp.2.*.engineType: GENERAL

The system has 2 digit-specific recognizers and 1 general purpose recognizer. Be sure to use the digComp applet with the DIGIT recognizer to get optimum digit performance. The application can then receive high performance on digits as well as high performance on standard tasks by switching to the appropriate engine task for the utterance. The above example is not applicable for those systems that use a local TsmProc since the engine can be in either mode but its setting is determined at startup. To choose which setting to use, determine whether your application is mostly digits or mostly general English and then choose the appropriate setting.

You can obtain the current value of taskId by issuing the command `listres -s chp.*.Tsmapi` for channel processes or `listres -s tsmp.*.Tsmapi` for remote engine processes.

<table>
<thead>
<tr>
<th>useEngineMNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Name</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>useEngineMNR</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies whether to use the engine MNR file or a local MNR file specified with the InitialMNRFile resource. Normally, the default value should be used.

<table>
<thead>
<tr>
<th>useLocalTsmProc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Name</td>
</tr>
<tr>
<td>useLocalTsmProc</td>
</tr>
</tbody>
</table>

Remarks:

This resource controls whether the Tsm object uses a local TsmProc object or a remote one. A value of 1 specifies a local TsmProc object; a value of 0 specifies a remote TsmProc object. For more information on the Tsm process, see ViaVoice Telephony Processes.

For distributed systems, set this resource to 0 on the client machines and 1 on the server machines.

<table>
<thead>
<tr>
<th>userid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Name</td>
</tr>
<tr>
<td>userid</td>
</tr>
</tbody>
</table>

Remarks:
This resource controls the user ID that is used by the ViaVoice engine. During installation, this resource is set to a unique value for each channel installed. Refer to ViaVoice Telephony Resources.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>voiceFormat</td>
<td>*.Audio</td>
<td>1</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies whether the system is receiving Ulaw or Alaw data. The default is Ulaw (1). Set this parameter to 0 to indicate Alaw data.
VTTDefaults: Text-to-Speech

This section describes the resources that define the behavior of the text-to-speech engine. These include voice characteristics and speaking style. Note that there are several reserved text-to-speech resources that are not presented.

allowAnnotations

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowAnnotations</td>
<td>*.Tts</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Remarks:
This resource indicates whether or not embedded annotations are supported within the text to be converted. These annotations change the characteristics of the voice (for example, from a male voice to a female voice). A value of "yes" indicates that embedded tags are allowed within the text string to be converted.

breathiness

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>breathiness</td>
<td>*.Tts</td>
<td>25</td>
</tr>
</tbody>
</table>

Remarks:
This resource sets the breathiness attribute of the text-to-speech voice. The value can range from 0 to 100.

gender

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the name of the text-to-speech engine. The default engineType for a text-to-speech engine is "tts."
Gender

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>gender</td>
<td><em>.</em>.Tts</td>
<td>male</td>
</tr>
</tbody>
</table>

Remarks:
This resource sets the gender attribute of the text-to-speech voice. The default gender is male. It can also be set to female.

Head Size

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>headSize</td>
<td><em>.</em>.Tts</td>
<td>50</td>
</tr>
</tbody>
</table>

Remarks:
This resource sets the head size attribute of the text-to-speech voice. The value can range from 0 to 100.

Log Text

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logText</td>
<td>ttsp.*.Tsmapi</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:
This resource controls whether each sentence that is sent to the text-to-speech engine is logged. This is useful for debugging purposes. The files that are created are named tts_text.<channel-number>.log and are placed in VVT_HOME. The default is 0, indicating that sentences sent to the engine for synthesis are not logged.

Pitch Baseline

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pitchBaseline</td>
<td><em>.</em>.Tts</td>
<td>60</td>
</tr>
</tbody>
</table>

Remarks:
This resource sets the baseline pitch attribute of the text-to-speech voice. The value can range from 0 to 100.

Pitch Fluctuation
Remarks:
This resource specifies how much the pitch of the text-to-speech voice can fluctuate. The value can range from 0 to 100.

quality

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>quality</td>
<td><em>.</em>.Tts</td>
<td>25</td>
</tr>
</tbody>
</table>

Remarks:
This resource sets the quality attribute of the text-to-speech voice. The value can range from 0 to 100.

roughness

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>roughness</td>
<td><em>.</em>.Tts</td>
<td>20</td>
</tr>
</tbody>
</table>

Remarks:
This resource sets the roughness attribute of the text-to-speech voice. The value can range from 0 to 100.

routerHost

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>routerHost</td>
<td><em>.</em>.Tts</td>
<td>localHost</td>
</tr>
</tbody>
</table>

Remarks:
This resource defines which host(s) have text-to-speech routers running. It allows a Tsm object to automatically find a text-to-speech engine of the desired type. The host list is a space-delimited list of host names (enclosed within quotes if more than one host name is specified). For example, if you specify the following in the VTTDefaults file:

chp.*.Tts.routerHost: "eric"

The Tsm class will search the tsmRouter on eric for a text-to-speech engine required for the application.
### speed

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed</td>
<td>*. *.Tts</td>
<td>60</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource sets the overall speed of the utterance in words spoken per minute. The speed value can range from 0 to 100.

### useTts

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>useTts</td>
<td>*. *.Tts</td>
<td>0</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource controls whether text-to-speech is available to the application. The default is 0, meaning that text-to-speech is not available to the application. This resource must be set to 1 to use text-to-speech.

### volume

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume</td>
<td>*. *.Tts</td>
<td>99</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource sets the loudness of the utterance. The valid range is 1 to 100.
VTTDefaults: Tcl Script Resources

This sections describes the Tcl script resource provided in the VTTDefaults file. Each application defines its call flow in a Tcl script. Each channel can execute a different Tcl script. Since each script is read only at process startup, changes to this resource are only activated by restarting the channel process.

**preloadGrammars**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>preloadGrammars</td>
<td>chp.*.CallFlow</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies whether grammars are to be preloaded. By default, grammars are not preloaded.
VTTDefaults: Telephony

The following section describes the telephony resources supported in the VTTDefaults file.

Some of the telephony resources are independent of the installed telephony adapter. Since the ViaVoice Telephony system supports both Dialogic and Natural Microsystems adapters, there are two other categories of telephony resources:

**Dialogic**

**Natural Microsystems**

### bargeln

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>bargeln</td>
<td><em>.</em>.*</td>
<td>0</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies whether the adapter will perform the barge-in (interrupting a prompt when speech is detected). The default value is 0, meaning that barge-in is off. Note that bargeln and hostBargeln have a mutually exclusive relationship; only one of these resources can be set.

### beepDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beepDuration</td>
<td><em>.</em>.TelLine</td>
<td>80</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies (in milliseconds) the duration of a beep during a phone call. For both Dialogic and Natural Microsystems adapters, the valid range is 0 to 65535.

### beepFrequency

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beepFrequency</td>
<td>chp.*,TelLine</td>
<td>1500</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the frequency (in hertz) at which a beep is played during a phone call. For Dialogic adapters, this value can range from 200 Hz to 2000 Hz. For Natural Microsystems adapters, this value can range from 200 Hz to 3600 Hz.
### beepLevel

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>beepLevel</td>
<td>chp.*.TelLine</td>
<td>-10</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the loudness of the beep (in decibels). For Dialogic adapters, this value can range from -40 dB to 0 dB. For Natural Microsystems adapters, the value can range from -54 dB to 3 dB.

### cleardownDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleardownDuration</td>
<td>chp.*.TelLine</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the minimum time to qualify a signal as a cleardown. Specify the time in milliseconds.

### cleardownFrequencies

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleardownFrequencies</td>
<td>chp.*.TelLine</td>
<td>350 40 440 40</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the cleardown frequencies. The value is specified as "frequency1 bandwidth of frequency1 frequency2 bandwidth of frequency2".

### cleardownLevel

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleardownLevel</td>
<td>chp.*.TelLine</td>
<td>-28</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the loudness of the cleardown tone in decibels. The value range is -40 to -0.
cleardownPulse

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cleardownPulse</td>
<td>chp.*.TelLine</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the cleardown pulse. The value is "count minimum-on maximum-on minimum-off maximum-off".

dialPauseDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialPauseDuration</td>
<td>chp.*.TelLine</td>
<td>500</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the duration of a pause in a dial string. (A pause is marked by ‘,’ in the dial string). It is specified in milliseconds.

dialtoneDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialtoneDuration</td>
<td>chp.*.TelLine</td>
<td>1000</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the minimum time to qualify a signal as a dialtone. Specify the time in milliseconds.

dialtoneFrequencies

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialtoneFrequencies</td>
<td>chp.*.TelLine</td>
<td>350 40 440 40</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the dial tone frequency. The value is specified as "frequency1 bandwidth of frequency1 frequency2 bandwidth of frequency2".

dialtoneLevel
### dialtoneLevel

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialtoneLevel</td>
<td>chp.*.TelLine</td>
<td>-28</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the loudness of the dial tone in decibels. The value range is -40 to -0.

### didLength

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>didLength</td>
<td>chp.*.TelLine</td>
<td>3</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource is applicable only to T1 trunks. It specifies the expected number of dialed digits for DID signalling.

### echoCancellation

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>echoCancellation</td>
<td>chp.*.TelLine</td>
<td>off</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies whether echo cancellation is active on the board. This resource applies only to Natural Microsystems adapters and must match the settings in the Natural Microsystems configuration file (ag.cfg). The value can be "on" or "off."

### flashDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>flashDuration</td>
<td>chp.*.TelLine</td>
<td>500</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the duration of a flash in milliseconds.
flashOnTransfer | chp.*.TelLine | 0

Remarks:
This resource specifies whether a flash should precede a call transfer. The default is 0, meaning flashOnTransfer is off.

hostBargeIn

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostBargeIn</td>
<td><em>.</em>.*</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies whether the host application will perform barge-in (interrupting a prompt when speech is detected). Use the `aud_write()` function to specify the prompt to be played on bargein and whether or not the prompt is ongoing. The default value is 0, meaning that the host application will not perform barge-in.

Note that hostBargeIn and bargeIn have a mutually exclusive relationship; only one of these resources can be set.

ignoreMissingPrompt

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ignoreMissingPrompt</td>
<td>chp.*.TelLine</td>
<td>on</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the behavior of the system when attempting to play a prompt that does not exist. If this resource is set to off, the Tcl channel process exits when a prompt is missing. An error message that contains the prompt name is always logged.

loadTest

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadTest</td>
<td>chp.<em>.</em></td>
<td>off</td>
</tr>
</tbody>
</table>

Remarks:
This resource controls whether a DTMF ‘1’ is played before recognition starts to that the system can be tested automatically by a load test computer. This resource is useful in a test environment only. In application software, it should be set to off.
### logCalls

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logCalls</td>
<td>chp.<em>:</em></td>
<td>off</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies whether the speech collected in the recognition session should be preserved in a .rec file. This resource is used to turn on/off the creation and storage of *.rec files in the \pcm directory. If the `logCalls` resource is set to on (in the vttdefaults file), REC (*.rec) files will be created in the \pcm directory. If the `logCalls` resource is set to off (the default state), no REC files are created in the subdirectory. It is the application developer’s responsibility to guarantee uniqueness of the filenames.

### logPcm

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logPcm</td>
<td>chp.<em>:</em></td>
<td>off</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies whether the speech collected during the recognition session should be preserved in a file. If the `logPcm` resource is set to on, mulaw (*.ulw) files will be created in the directory specified in the vttdefaults file (chp.*:*.*.logPcmDir:). The default directory is \pcm. If the `logPcm` resource is set to off (the default state), no *.ulw files are created in the subdirectory. It is the application developer’s responsibility to guarantee uniqueness of the filenames.

### maxResponseDelay

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxResponseDelay</td>
<td>chp.*:TelLine</td>
<td>180</td>
</tr>
</tbody>
</table>

**Remarks:**

This resource specifies the length of time to wait for completion of any action. The value is specified in seconds.

### model

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>chp.*:TelLine</td>
<td>analog</td>
</tr>
</tbody>
</table>
Remarks:
This resource specifies the type of telephony interface. Valid values are "analog", "t1", or "pri". Note that "pri" is only supported for Dialogic without call transfer.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>oem</td>
<td>chp.*.TelLine</td>
<td>Dialogic</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the type of telephony hardware used. Valid values are either "NMS", "Dialogic", or "Wave".

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>offhookWhenBlocking</td>
<td>chp.*.TelLine</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:
This resource applies to analog lines only. It specifies whether the line is placed off hook when the Tcl channel process exits. The default value is 0, meaning that the line is not placed off hook when the Tcl channel process exits.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>preWinkDuration</td>
<td>chp.*.TelLine</td>
<td>100</td>
</tr>
</tbody>
</table>

Remarks:
This resource is applicable only to T1 EM and T1 DID signalling. It specifies the time duration, in milliseconds, prior to beginning wink signalling. This resource is applicable only if wink start is enabled.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordDuration</td>
<td>chp.*.TelLine</td>
<td>300 (Dialogic)</td>
</tr>
</tbody>
</table>
Remarks:

This resource specifies the maximum size of a recording. Specify 0 for no limit to the size. The value should be specified in 100 millisecond units for Dialogic, and milliseconds for Natural Microsystems.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>recordFileSignature</td>
<td>chp.*.TelLine</td>
<td>&quot;&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Remarks:

An application-defined string can be stored as a header in recorded PCM files. By default, no header information is included in PCM files.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalling</td>
<td>chp.*.TelLine</td>
<td>FX</td>
<td></td>
</tr>
</tbody>
</table>

Remarks:

This resource is applicable only to T1 trunks. It specifies the signalling type. The value may be FX, DID, or EM.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>telSilenceDetection</td>
<td><em>.</em>.*</td>
<td>0</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies whether silence detection on the telephony adapter is used. The default value is 0, meaning that the adapter silence detection mechanism is not used.

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadPriority</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
threadPriority  chp.*.TelLine  0

Remarks:
This resource specifies the thread priority. The value "0" denotes normal priority; "1" denotes above normal priority; "2" denotes the highest priority.

transferPause

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>transferPause</td>
<td>chp.*.TelLine</td>
<td>1000</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies whether an extra pause is inserted after dialing a digit string for call transfer and before hangup.

transferTo

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>transferTo</td>
<td>chp.*.TelLine</td>
<td></td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies whether all call transfer requests are transferred to the number specified by this resource. This is useful for testing purposes.

winkDuration

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>winkDuration</td>
<td>chp.*.TelLine</td>
<td>150</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the duration of wink in milliseconds. It is applicable only to T1 EM and T1 DID signaling.
<table>
<thead>
<tr>
<th></th>
<th>winkStart</th>
<th>chp.*.TelLine</th>
<th>0</th>
</tr>
</thead>
</table>

**Remarks:**

This resource is applicable only to T1 trunks. It defines wink start signaling for EM. The value can be 0 or 1. The default value of 0 indicates that wink start is off.
VTTDefaults: Dialogic Resources

The Dialogic adapters are accessed through device drivers, the names of which are constructed from the board number and the channel number. The Dialogic resources define these two values separately for analog and digital trunk cards.

The following section describes the Dialogic resources supported in the VTTDefaults file:

**boardNumber**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boardNumber</td>
<td><em>.</em>.*</td>
<td>-1</td>
</tr>
</tbody>
</table>

Remarks:
For channels 0 through 3 set boardNumber to 1, for channels 4 through 7 set boardNumber to 2, and so on.

**channelNumber**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelNumber</td>
<td><em>.</em>.*</td>
<td>-1</td>
</tr>
</tbody>
</table>

Remarks:
Set to 1 for cloneId 0+n*4, 2 for cloneId 1+n*4 and so on.

**t1BoardNumber**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1BoardNumber</td>
<td><em>.</em>.*</td>
<td>1</td>
</tr>
</tbody>
</table>

Remarks:
Set to 1 for cloneId 0 through 23.

**t1ChannelNumber**

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1ChannelNumber</td>
<td>.<em>.</em></td>
<td>-1</td>
</tr>
</tbody>
</table>

**Remarks:**

For cloneId 0 through 23, set to 1 through 24.
The following section describes the Natural Microsystems resources supported in the VTTDefaults file. The Natural Microsystems adapters provide additional functions such as simultaneous play and record and barge-in. By default, the Natural Microsystems configuration is shared between all channels. The ViaVoice telephony software creates a private copy of ADI_RECORD_PARMS, ADI_PLAY_PARMS, ADI_ENERGY_PARMS, ADI_START_PARMS, and ADI_COLLECT_PARMS in each channel process and overrides some of the Natural Microsystems default values with the ones supplied through these resources. NMS resources are defined separately for analog and digital T-1 trunk cards).

### boardNumber

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boardNumber</td>
<td><em>.</em>.*</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Remarks:**

For channels 0 through 3 set boardNumber to 1, for channels 4 through 7 set boardNumber to 2, and so on.

### channelNumber

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelNumber</td>
<td><em>.</em>.*</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Remarks:**

Set to 1 for cloneld 0+n*4, 2 for cloneld 1+n*4 and so on.

### nmsBoardType

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsBoardType</td>
<td><em>.</em>.*</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

**Remarks:**

Specifies whether you are using a QX or AG adapter.

### nmsECAdaptTime

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
</table>
Remarks:

This resource specifies the Echo cancellation adaptation time. Valid values range 100 to 1000 milliseconds.

### nmsECFilterLength

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsECFilterLength</td>
<td><em>.</em>.TelLine</td>
<td>6</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the echo cancellation filter length. Valid values range from 2 to 20.

### nmsEnergyDeglitch

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsEnergyDeglitch</td>
<td><em>.</em>.TelLine</td>
<td>150</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the duration (in milliseconds) of a speech signal that is needed to report energy detection to the application. The valid range is from 0 to 32000.

### nmsEnergyThreshold

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsEnergyThreshold</td>
<td><em>.</em>.TelLine</td>
<td>-35</td>
</tr>
</tbody>
</table>

Remarks:

This resource specifies the minimum level of a signal triggering energy detector. This value is specified in dB, with a valid range of –51 to -15.

### nmsProtocol

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsProtocol</td>
<td><em>.</em>.TelLine</td>
<td>LPS0</td>
</tr>
</tbody>
</table>
Remarks:
This resource defines the signaling protocol. The default value is appropriate for analog cards. For T1 adapters, it should be "nocc".

### nmsRecordBufferSize

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsRecordBufferSize</td>
<td><em>.</em>.TelLine</td>
<td>4000</td>
</tr>
</tbody>
</table>

Remarks:
This resource specifies the default buffer size used in recording. Implicitly, it also defines the frequency with which data is passed between the Natural Microsystems adapter and the application. Note that two buffers are always used.

### nmsRecordStepBack

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmsRecordStepBack</td>
<td><em>.</em>.TelLine</td>
<td>350</td>
</tr>
</tbody>
</table>

Remarks:
When barge-in is enabled, recording may be started at the same time a prompt is played. The collected speech is not passed directly to the recognition engine but is buffered locally. Once the adapter reports energy detected, the engine receives data starting from the frame equal to the energy detector signal minus this resource-defined value (approximately from the beginning of speech). This resource is specified in milliseconds.

### t1BoardNumber

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1BoardNumber</td>
<td><em>.</em>.*</td>
<td>1</td>
</tr>
</tbody>
</table>

Remarks:
Set to 1 for cloneId 0 through 23.

### t1ChannelNumber

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Default Scope</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1ChannelNumber</td>
<td><em>.</em>.*</td>
<td>-1</td>
</tr>
</tbody>
</table>

Remarks:
For cloneId 0 through 23, set to 1 through 24.
Phone Recognition Grammars Overview

Phone recognition is an extension of speech recognition. Speech recognition refers to using a microphone (or other input device directly attached to a computer) to speak to a computer. The computer recognizes what was said and an associated speech-enabled application takes appropriate action. Phone recognition, on the other hand, refers to talking through a telephone, over a telephone line to a computer. As with speech recognition, the computer recognizes what was said and an associated telephony application takes appropriate action.

Speech recognition can augment and even replace keyboard and mouse interaction with the computer. For example, instead of typing text, the user can dictate text to the computer. Or, the user can control applications through voice commands instead of using the mouse. Similarly, phone recognition can augment or replace the phone keypad. The caller can speak a request, rather than having to use the phone keypad to interact with the telephony application.

Grammars provide the mechanism for defining the set of spoken words that are to be recognized.

About Vocabularies

When planning your grammar-based telephony application, you must identify all of the words, phrases and sentences that a caller can say. Each collection of words, phrases, and sentences form a vocabulary. You can have multiple vocabularies active at the same time. This helps you group and combine things the user will say to your application.

A vocabulary can be as restrictive or as flexible as your application needs to be. Of course, there is a trade-off of recognition speed and accuracy versus the size of the vocabulary. You may want to experiment with different vocabularies to validate a design that best matches the requirements and expectations of your users.

A vocabulary is defined to your application and the recognition engine through a grammar. You can specify a vocabulary in one of two ways: either as a predefined grammar or as a dynamic command vocabulary. A dynamic command vocabulary is a vocabulary that is defined by your application while it is running, is made known to the recognition engine by your application bypassing the grammar definition stage.

What is a Phone Grammar?

A phone grammar is a structured collection of words and phrases bound together by rules. A grammar defines a vocabulary to your application and to the telephony recognition engine. The telephony recognition engine uses the associated rules to match spoken words, phrases or sentences against the phone grammar and provides information to your application. Your application then uses the information to perform specific actions. Note that users of your application are constrained to speaking words, phrases, and sentences allowed by the phone grammars.
Examples of Simple Phone Grammars

Common phone grammars with which most of us have experience are those used when requesting telephone Directory Assistance. The selection of the grammar(s) to be used is pinpointed by the request; that is, by region, area code, city, and so on. Phone grammars for Directory Assistance include the published names of persons, businesses, and institutions; regions, states, and cities.

Another example of a phone grammar is one used for digit dialing. Digit dialing, useful for car phones or for elderly or handicapped persons, allows a user to place a call by saying telephone numbers instead of dialing numbers. The phone grammar for digit dialing consists of the single digits (zero, one, two, three, and so on) and possible combinations of the single digits (for example, twelve hundred, fifteen, thirty-five hundred).

Name dialing allows a person to dial a telephone number by speaking the name of a person, business, or phrase. The phone grammars for name dialing include the names, phrases and locations.

Why is a Phone Grammar Necessary?

Phone grammars guide the telephony recognition engine to what a caller may say or might say next. Consider phone grammars for a telephone Directory Assistance application. A caller is asked through voice prompts, “what state?”, “what city?” and so on. Grammars for the United States domain would include:

- The names of the 50 states (for example "Alabama", "Ohio", "Texas")
- The names of every city in each state

A caller's response of a state leads the recognition engine to the path of grammars that identify the valid cities.

Grammars provide a language model for the recognition engine, constraining the valid set of words to be considered and increasing recognition accuracy while minimizing computational requirements.

The tradeoff for speed and accuracy, however, is that the caller is also constrained to speaking commands that stay within the grammar definition. Larger grammars can be more tolerant of user variation (for example, adding alternative commands such as "the state of <state_name>"), but at the potential expense of command latency and reduced accuracy.

There is also a tradeoff with user expectations. Larger grammars might also set high user expectations, since it may be perceived that one can say "just about anything." Without real natural-language processing, the speech recognition system cannot anticipate all the different ways a user will structure a command. Also, you don't want to support so many variations that the user can't remember what to say.

Consideration of these inherent design tradeoffs is fundamental to designing robust and efficient phone grammars. See Guidelines for Designing Grammars.

Dynamic Command Vocabularies

A vocabulary consists of all the words, phrases, and sentences that a caller can say and be recognized
by the engine. A dynamic command vocabulary is defined by your application while it is running.

Dynamic command vocabularies afford a quick and easy way to incorporate basic speech recognition into your application. For example, you might want your users to be able to speak the words on buttons. To provide this capability, you can define these words within your application at run time. This is known as defining a dynamic command vocabulary.

Dynamic command vocabularies can be quite useful in this way, since you reap many of the benefits of speech recognition without the development overhead of defining and compiling grammars. Keep in mind, though, that dynamic command vocabularies support a "what-you-see-is-what-you-say" model (for example, if you defined "Add name" as a valid phrase, the user must say "Add name," not "Add a name" or "Add this name").

Dynamic command vocabularies can be used in conjunction with grammars generated for vocabularies that do not change while the application is running. This comes in quite handy when you don't know all of the words in your vocabulary at the time you're defining your grammar. Consider a telephone dialer application, where you have a list of names that aren't known to you when you're developing your grammar. These names could be read from an address book at run time, and added dynamically to your vocabulary with the appropriate calls.

Acceptance or Rejection of Utterances

Since the user may say words that are not in the vocabulary or be in an environment with background noise, the engine implements accept/reject logic for grammars. Providing the logic in the engine simplifies application development, and makes performance consistent across applications.

An utterance is any stream of speech that represents a caller's complete command. Not all utterances are accepted by the recognition engine. Acceptance or rejection is flagged on each returned phrase. The engine will reject an utterance for these reasons:

- An incomplete path through the grammar is encountered, and the timeout is exceeded. This happens if the user stops talking before completing a phrase, or if the engine incorrectly goes down the wrong path. An application has control over this by setting the incomplete timeout, minSilenceDuration, which effectively determines the tolerance for pausing within a phrase or utterance.
- The scores of the words are sufficiently low relative to the threshold setting. An Application can specify the threshold level through the DmThresholdReject resource in VttDefaults.

Guidelines for Designing Grammars

The following are suggestions for designing grammars:

- Use grammars with few phrases and a simple syntax to permit a faster and more accurate speech recognition process.
- Limit the size of the grammar so that fewer words must be matched. Note that a grammar cannot contain more than 65,000 words.
• Develop long and narrow grammars, rather than short and wide grammars, to limit the number of legal possibilities at any given point in the grammar.

• Avoid using words that are very similar in pronunciation in the same portion of your grammar. For example, if you are writing a child's game, you would not want to define a grammar "pick up the hat" and "pick up the cat." Instead, you could specify the grammars as "pick up the blue hat" and "pick up the yellow cat."

• Allowing the user multiple ways of saying a particular command can enhance usability. But, large grammars might also set user expectations unrealistically high; that is, users might perceive that they can say almost anything, which, of course, is not true. Recognition accuracy and speed can deteriorate as users speak more randomly based on their expectations.

• Don't support so many ways of saying commands that your users won't remember what to say. You should probably test your grammars with a number of potential users, and support the phrases that are most intuitive and meaningful to them. You might also want to provide a visual mechanism for reminding users what they can say at any given point.
Developing Phone Grammars

You must supply the phone grammars that the telephony recognition engine uses for speech recognition. Before beginning to develop your phone grammars, you must have planned what a user can say and how your application is to handle unusual occurrences such as pauses and mumbles. Read Phone Recognition Grammars Overview, to understand the preparations needed.

Grammar development consists of the following steps:

1. Vocabulary definition: specifying the words, phrases, and sentences to be recognized by the engine.
2. Creating source grammar (.bnf) files from the vocabulary
3. Creating compiled grammar (.fsg) files from the source grammar files
4. Creating a pronunciation dictionary, also called baseforms, using grammars or directly from a vocabulary
5. Creating the recognition engine pronunciation pool.

ViaVoice Telephony Tools provides several options for grammar development. These are the options:

- If your application requires a simple grammar, you can create a flat file containing the list of words and phrases to be recognized. The tools provide utilities that use this file to compile and generate grammars and pronunciations for your telephony application. See Developing Grammars Using Lists or see Developing Grammars Using the Administration Tools.

- The Tools provide several samples of component grammars that are useful for specific types of applications such as banking and voice-mail applications. You can use and extend these grammars to quickly and easily develop phone grammars for your applications. See Sample Grammars.

- If your application requires a complex grammar specification, you should use the Speech Recognition Control Language (SRCL). SRCL provides a formal methodology for grammar definition that imposes structure and allows more robust grammar capabilities. See Developing Grammars Using SRCL.

Pronunciation dictionary (baseforms) generation is an integral part of some options. Other options provide a separate method for baseform generation. Baseforms can be generated from grammars or from a vocabulary.

Regardless of the option used, after developing your phone grammars, you should test them. Refer to Testing Your Phone Grammars.
Developing Grammars Using Lists

A key element of your phone recognition application is the set of words and phrases that a user can say. These words and phrases are specified as grammars. Grammars can be defined in text files that follow a prescribed format. The ViaVoice Telephony Tools provide utilities that generate all of the files necessary to use these grammars for speech recognition. This method of creating grammars is recommended for applications where simple grammars are sufficient.

You can use either the administration tools or the makeApp utility to quickly and easily generate your grammars and all of the files needed for speech recognition. See either Grammar Development Using the Administration Tools or Using the makeApp Utility.

Using the makeApp Utility

To develop a grammar using simple word lists and phrases, you:

1. Prepare the words and phrases to be recognized by the Telephony Recognition Engine.
2. Create your phone grammars and pronunciation dictionary.
3. Create the recognition engine pronunciation pool.
4. Test your phone grammars.

Preparing Words and Phrases for Phone Recognition

You prepare one or two files that identify the words and phrases to be recognized and associated rules for recognition. These are:

1. A file that specifies the words and phrases that the engine can recognize. This is a simple text file that must be in a particular format, known as subscriber.db format. The file can have any name and extension; in the example below, subscriber.db is used for clarity.
2. Optionally, a file that specifies words that may be used preceding and following the target words. This is a simple text file and is known as the subscriber.wrap file. It must have the same name as the subscriber.db file and must have an extension of .wrap.

Preparing the subscriber.db file

This file specifies the words and phrases to be recognized. Create this file using a text editor of your choice. The file may reside in any directory.

The format of a grammar entry in the subscriber.db file is:

<grammarid> <annotation> <any text>
where:
<grammarid> is the identification of the grammar. It can be any text.
<annotation> is information that is passed to your application along with the recognized text.
<any text> is a valid text string of up to 100 words.

To illustrate the format, consider the following example subscriber.db file.

Control 1 CANCEL
Control 2 REPEAT
bool 1 YES
bool 0 NO

For this example, the set of allowable words that a caller can say and that can be recognized by the telephony engine is "cancel", "repeat", "yes" and "no". The annotation for "cancel" is 1, "repeat" is 2, "yes" is 1 and "no" is 0. Note that in this example, if both "Control" and "bool" are enabled at the same time then both could return the annotation "1".

Preparing the subscriber.wrap file

This file contains the words that are allowed to "wrap" the allowable words or phrases a caller can say. Create this file as well using the text editor of your choice. This file must reside in the same directory as the subscriber.db file.

INTROs specify words or phrases that can precede any word or phrase that is specified in subscriber.db. OUTROs specify words or phrases that a user can say at the end of a sentence. INTRO and OUTRO are optional so that a user is permitted to speak only the words and phrases in subscriber.db. The format of a wrapper entry is:

INTRO <any text> or OUTRO <any text>

where <any text> is a valid text string of up to 100 words.

To illustrate the format, consider the following example subscriber.wrap file.

INTRO Please give me
INTRO I would like
INTRO How about
OUTRO Please

These specifications allow a caller to say any of the following preceding a word or phrase that is specified in subscriber.db:

"Please give me"
"I would like"
"How about"

It also allows a caller to say the following at the end of an utterance:

"Please"
The makeApp Utility

The makeApp utility generates all of the files you need for speech recognition. It generates the grammars, creates the associated pronunciation file (baseforms), and makes the engine's pronunciation pool.

The utility is comprised of three component utilities:

- makeGrammars
- makeBsfms
- makePol

The following diagram illustrates the process of developing a grammar using the makeApp utility:

Before running the utility, ensure that you are logged on as the userid that installed the Telephony Tools. To run the utility, on the command line, enter:

```
makeApp.pl <appname> <location of subscriber.db file> <-l Ll_LL>
```

where:

- `<appname>` is the name of your application
- `<location of subscriber.db file>` is the complete directory path where the subscriber.db and optionally, the subscriber.wrap, files reside. If the subscriber.db file is not found, the utility looks for the file in the directory `<VTT_HOME>\app\Ll_LL\appname`. The appname directory must exist within this directory structure.
- `<-l Ll_LL>` specifies the language for which the grammar is being created. By default, the language is
U.S. English. Specify this parameter whenever the language of the application is not U.S. English. LL_LL can be one of the following:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

Note: The makeApp utility is a Perl script; therefore, it is essential that you specify the .pl extension to access and use it. You can also use the command (.cmd) file version of makeApp, which invokes Perl for the makeApp.pl script. The format of makeApp.cmd is:

```
makeApp <appname> <location of subscriber.db file> <-l LL_LL>
```

If you don't specify the .pl extension, the command file runs by default.

Upon completion, the following files are generated for your application:

1. Grammar source files (.bnf) and compiled grammar files (.fsg) are in the directory <VTT_HOME>\app\LL_LL\appname\grammars.
2. A pronunciation dictionary file (baseforms) is in the directory <VTT_HOME>\app\LL_LL\appname\baseforms. The baseforms file contains the pronunciations for all of the words and phrases in the grammars.
3. The pronunciation pool files that are required by the speech recognition engine are in the directory <VTT_HOME>\app\LL_LL\appname\baseforms\appname>.

For the example subscriber.db file:

```
Control 1 CANCEL
Control 2 REPEAT
bool 1 YES
bool 0 NO
```

The makeApp utility yields two .bnf files:

- <VTT_HOME>\app\LL_LL\appname\grammars\appname\Control.bnf
- <VTT_HOME>\app\LL_LL\appname\grammars\appname\bool.bnf

Upon completing these steps, all of the files needed for speech recognition have been generated. You should now test your grammars. Refer to Testing your Grammars.
Developing Grammars Using SRCL

Introduction to SRCL

A key element of your phone recognition application is the set of words, phrases, and sentences that a user can say. This set is specified as grammars. The Speech Recognition Control Language (SRCL) allows a grammar designer to produce very powerful and complex grammars using a prescribed methodology.

This section describes the process for developing your application's grammars using SRCL. The ViaVoice Telephony Tools provide utilities and tools that generate all of the files necessary to use these grammars for phone recognition.

To develop a grammar using SRCL, you:

- Prepare as SRCL specifications, the words, phrases, and sentences to be recognized by the Telephony Recognition Engine.
- Create your phone grammars through compilation.
- Create your pronunciation dictionary
- Create the recognition engine pronunciation pool
- Test your phone grammars

Preparing SRCL Specifications

You prepare all of the words, phrases, sentences and accompanying rules using SRCL. Use a text editor to place the SRCL specifications into a file that has the expected grammar file extension .bnf. Each specification occupies a line in the file.

Refer to Introduction to SRCL for an explanation of using the language.

After constructing your grammar file (.bnf), you then compile it. Refer to the Grammar Compiler.

Creating the Pronunciation Dictionary

Use the makeBsfms utility to generate the pronunciation dictionary for your application. Specify the directory that contains the compiled grammar files (.fsg). The utility creates the dictionary as a file named baseforms. It is placed in the directory:

<VTT_HOME>\app\<appname>\baseforms\n
where:

<VTT_HOME> is the directory into which the ViaVoice Telephony Tools were installed.
<appname> is the name of your application.
Creating the Pronunciation Pool

Use the makePol utility to generate the pronunciation pool. Specify the name of your application.

Upon completing these steps, all of the files needed for speech recognition have been generated. You should now test your grammars. Refer to Testing Your Grammars.
Introduction to SRCL

One way to represent a speech grammar is to use Backus-Naur Form, or BNF. This form of grammar representation is used to describe the syntax of a given language and its notation. Formalisms are generally understood and used throughout the world of formal language theory and processing.

This section describes the syntax of a particular type of BNF grammar that has been adapted to the task of speech recognition. It is called Speech Recognition Control Language (abbreviated SRCL and pronounced "circle"). SRCL was developed jointly between the SRAPI (Speech Recognition API) Committee and ECTF (Enterprise Computer Telephony Forum). IBM is a member of both organizations. Information on SRAPI can be found at http://www.srapi.com; likewise, information on ECTF can be found at http://www.ectf.org.

Grammars constructed using Speech Recognition Control Language (SRCL pronounced "circle") offer an organized view of the words and phrases that are part of the speech grammar, since they define a notation for identifying common phrases, optional phrases, and repeated phrases. Throughout this section, the terms BNF and SRCL refer to the SRCL command language.

A speech grammar is defined by enumerating the valid words and phrases. The general form of a SRCL grammar is as follows:

<rule> = this is the right side .

This general form is called a production rule. Every SRCL grammar must start with and contain at least one production rule. The production rule has four parts:

- **The left side (<rule>):** This part of the production rule is required, although it is not necessary to specify the string exactly as we have done. In general, any string may be used as long as it is placed between angle brackets <>, and the string formed by the angle brackets and the enclosed word is unique (that is, it is not used elsewhere in the BNF). Hence, <rule> must not appear elsewhere in the grammar.

- **The assignment operator (= or =):** This part of the production rule is required and must be written as shown.

- **The right side (this is the right side):** This part of the production rule defines all of the sentences and phrases that are valid in the speech grammar. The exact structure of these definitions is covered in the remaining parts of this section.

- **The end-of-production delimiter (period):** This part of the production rule is required and must be specified as shown. Although it is not necessary to separate the period from the sentences-and-phrases section of the rule, doing so often improves readability.

From looking at this definition, you might have noticed that the structure of the production rule is much more complex than it need be for a language that is capable of having only one such rule. In fact, a SRCL grammar can have an arbitrary number of such rules. Later in this discussion, you see how to define these grammars and how to relate their rules such that the entire speech grammar fits together into a single sophisticated speech grammar. For now, however, we will confine ourselves to those that contain only one rule.

The following is an example SRCL BNF grammar that defines menu selections for a windowed application. This simple example is shown for educational purposes.

<root> = FILE
Defining Common Words and Phrases with Nonterminal Symbols

The sentences that you use in everyday life are composed of many hundreds of common phrases and words. A SRCL grammar gives you a notation to define these common elements and to use them throughout your speech grammars. In so doing, it allows you to create sophisticated grammars out of a few sentence forms with many common words and phrases.

To see how the SRCL grammar syntax supports this, let's return to our two-phrase grammar above:

```
<root> = hello world
| hello there .
```

In the preceding discussions, the grammars have contained only terminal symbols. In our two-phrase example, the words "hello," "world," and "there" are examples of terminal symbols. Although there are numerous definitions of the term "terminal symbol" in the linguistic and programming community, when we use the term in describing a SRCL grammar, we mean words to be spoken. All of the grammars, so far, have contained only the words to be spoken arranged into collections of individual words, phrases, or both, separated by a vertical bar or "OR" symbol. However, everywhere that a terminal symbol can be used, it is possible to use a nonterminal symbol instead. To define a nonterminal symbol, you create a production rule of the form just described. In the preceding example, we could define a nonterminal symbol as:

```
<root> = hello world
| hello there .
<object> = world
| there .
```

Now, we have defined a grammar with two nonterminal symbols, <root> and <object>. The nonterminal symbol <root> is the special starting nonterminal and must be unique as previously outlined. The nonterminal symbol <object> is a simple nonterminal. Like the starting nonterminal symbol, it is a string of alphanumeric characters enclosed in angle brackets. In order to use a nonterminal symbol, place it somewhere in the right side of the production rule, either as a stand-alone word or in a phrase. Anywhere a terminal symbol can be used, a nonterminal symbol can also be used.

One way to use the nonterminal symbol is:

```
<root> = hello <object>
| hello <object> .
<object> = world
| there .
```

Notice that by using the nonterminal symbol <object> we have created a grammar with two identical
clauses. Hence, we could rewrite the grammar without the redundant clause.

<root> = hello <object> .
<object> = world
   | there .

Although this example is intentionally simple to allow each point to be explained easily, the use of nonterminal symbols simplifies the writing and maintenance of production grammars that might be both large and complex.

**Defining Optional Words and Phrases**

It is often desirable to define sentences that contain optional phrases. Many of these are in the form of an imperative preceded by a noun or pronoun of address. For example the command "May I see your passport?" might well be optionally preceded by "Sir," "Madam," or "Miss." One way to define this construct is:

<root> = <command>
| <title> <command> .
<title> = sir | madam | miss .
<command> = may I see your passport .

Although this example is a perfectly reasonable way to define such constructs, they are so common that SRCL provides the ? operator to define them. The above example is rewritten below using the ? operator.

<root> = <title>? <command> .
<title> = sir | madam | miss .
<command> = may I see your passport .

When the ? operator appears it causes the symbol to its immediate left to be defined as optional. If the symbol is a terminal symbol the operator causes it to be defined as an optional word. If the symbol is a nonterminal symbol the operator causes all of the clauses defined by that nonterminal symbol to be treated as optional. The ? operator cannot be used on the left side of a production rule.

**Defining Repeated Words and Phrases**

Another commonly seen construct in speech grammars is that of the repeated word or phrase. These are quite common in grammars that define arbitrarily long sequences of natural numbers. Phone numbers, credit card numbers, account numbers, and serial numbers fall into this category. As in the preceding example, SRCL supports this special case with a special purpose operator. We'll use a digit string grammar as an example. This grammar will cause the speech engine to recognize and accept as valid any number of digits zero through nine in any order:

<numbers> = <digit>+ .
<digit> = zero
   | one
   | two
   | three
   | four
   | five
   | six
When the + operator appears, it causes the symbol to its immediate left to be defined as one or more of. Hence, in our example, the <numbers> production rule is read: "Numbers is defined as one or more digit." The * operator is analogous to the + operator, except that it defines zero or more of the symbols to its left. Grammars should be written with the supported ?, +, and * operators, rather than with recursive rules. The use of recursion can often result in larger and less efficient grammars (than those written using the operators), and these grammars can often produce unintended, yet legal, word sequences. The grammar compiler, therefore, does not support recursive grammars.

Grammar Annotations--A Post Parsing Aid

The SRCL command language contains, in addition to the features that support defining a command grammar, a feature that reduces the complexity of parsing command grammar sentences. This feature, called grammar annotations or simply annotations, accomplishes its goal by allowing key words and phrases to be marked or annotated when the grammar is defined. Then, when the grammar is used in an application, and whenever an annotated word is recognized, both the word and the annotation are returned to the application. By choosing the annotations properly, you can simplify most parsing of command grammar sentences. This also simplifies development of grammars for multiple languages. Careful use of annotations can make the parsing independent of word order in a command, which will vary from language to language (for example, "start <program>" in U.S.English versus "<program> start" in German).

Defining Annotations

Here are some general rules for defining annotations and some general suggestions to help you get started. In the following BNF fragment we see the two major ways to define an annotation:

... tuesday:"Day_of_week"
... february:2

The first method allows a string to be used as an annotation. To use this form of annotation, place a colon after the symbol to be annotated and follow it with a quoted string that contains the data. The second method allows any decimal value in unsigned long to be used as tags. To use this form, place a colon after the symbol to be annotated and follow it with the number.

Aside from their syntax, keep the following rules and suggestions in mind when using annotations:

- Annotations can be attached to any terminal symbol.
- Annotations cannot be attached to a nonterminal symbol.
- Only one annotation per symbol is permitted.
The Kiosk Example

To illustrate the use of annotations, we will use a kiosk speech grammar. This is the type of grammar that you might find in an information kiosk at either the Olympics or an international pavilion in a major airport. The purpose of such a kiosk would be to provide automated assistance to visitors on a variety of topics. Its SRCL grammar is as follows:

\[
\begin{align*}
<kiosk> &= <greeting1>\? <greeting2>\? <sentence1> \\
&\quad | <greeting1>\? <sentence2> . \\
<greeting1> &= \text{hello} | \text{excuse me} | \text{excuse me but} . \\
<greeting2> &= \text{can you tell me} \\
&\quad | \text{I need to know} \\
&\quad | \text{please tell me} . \\
<sentence1> &= \text{where <destination1> is located} \\
&\quad | \text{where is <destination1>} \\
&\quad | \text{where am I} \\
&\quad | \text{when will <transportation> <destination2>\? arrive} \\
&\quad | \text{when <transportation> <destination2>\? will arrive} \\
&\quad | \text{what time it is} \\
&\quad | \text{the local time} \\
&\quad | \text{the phone number of <destination1>} \\
&\quad | \text{the cost of <transportation> <destination2>\?} . \\
<sentence2> &= \text{I am lost} \\
&\quad | \text{I need help} \\
&\quad | \text{please help me} \\
&\quad | \text{help} \\
&\quad | \text{help me} \\
&\quad | \text{help me please} . \\
<destination1> &= \text{a restaurant} \\
&\quad | \text{the <RestaurantType> restaurant} \\
&\quad | <BusinessType>\? <BusinessName> . \\
<RestaurantType> &= \text{best} | \text{nearest} | \text{cheapest} | \text{fastest} . \\
<BusinessType> &= \text{a} | \text{the nearest} . \\
<BusinessName> &= \text{filling station} \\
&\quad | \text{public rest room} \\
&\quad | \text{police station} . \\
<transportation> &= \text{the <TransportType>\? <TransportName>} . \\
<TransportType> &= \text{next} | \text{first} | \text{last} . \\
<TransportName> &= \text{bus} | \text{train} . \\
<destination2> &= \text{to metro central} \\
&\quad | \text{to union station} \\
&\quad | \text{to downtown} \\
&\quad | \text{to national airport} . \\
\end{align*}
\]

Take a minute to examine this grammar and try to imagine how you might design an application to process its requests. In case you are wondering, there are a possible 2664 sentences. Let’s list a few of them:

hello I need to know where the nearest police station is located
where is the nearest public rest room
please tell me the phone number of the nearest filling station
where am I
when will the next bus to union station arrive
excuse me but when will the next train to national airport arrive
help
I am lost
excuse me what time is it

This small grammar has the beginnings of a reasonable amount of flexibility. The sentences generated by this grammar could also present something of a challenge to an application parser. From a language processing perspective, the key to designing an efficient program is to recognize that of the 2664 possible sentences, only six different categories of information are requested:

1. A nonspecific request for assistance
2. A request for the local time of day
3. A request to locate a given landmark
4. A request for the phone number for a given business or facility
5. A request for information for public transportation
6. A request for fare information for public transportation

Providing such information, in and of itself, is not a difficult task, provided that the requests for it are straightforward. The difficulty lies in the fact that the information is being requested through a near-natural language interface. That interface, in order to be desirable and useful to speakers, must allow them to request information in a "natural" way. Hence, the grammar must anticipate many of the common ways that users might pose their questions and requests.

Fortunately, there is a way to almost totally eliminate the job of parsing. By attaching annotations to key phrases it is possible, through a relatively simple set of post-processing steps that involve only simple string manipulation, to quickly reduce the 2664 sentences to the six forms in the list and to place those requests in an almost template-like form that will be easy to process.

To illustrate how this is done, we will first rewrite the grammar to include the annotations.

```
kiosk = <greeting1>? <greeting2>? <sentence1>
   | <greeting1>? <sentence2> .
greeting1 = hello | excuse me | excuse me but .
greeting2 = can you tell me
           | I need to know
           | please tell me .
sentence1 = where:"op_locate" <destination1> is located
           | where:"op_locate" is <destination1>
           | where:"op_locate" am I
           | when:"op_sched" will <transportation> <destination2>? arrive
           | when:"op_sched" <transportation> <destination2>? will arrive
           | what:"op_time_of_day" time it is
           | the local:"op_time_of_day" time
           | the phone:"op_phone" number of <destination1>
           | the cost:"op_fare" of <transportation> <destination2>?
sentence2 = I:"op_help" am lost
           | I:"op_help" need help
           | please:"op_help" help me
           | help:"op_help"
           | help:"op_help" me
           | help:"op_help" me please .
<destination1> = a restaurant:"landmark"
```
the <RestaurantType> restaurant:"landmark"
<BusinessType>? <BusinessName> .

<RestaurantType> = best:"type"
| nearest:"type"
| cheapest:"type"
| fastest:"type" .

<BusinessType> = a:"type" | the nearest:"type" .

<BusinessName> = filling:"landmark" station:"landmark"
| public:"landmark" rest:"landmark" room:"landmark"
| police:"landmark" station:"landmark" .

<transportation> = the <TransportType>? <TransportName> .

<TransportType> = next:"type" | first:"type" | last:"type" .

<TransportName> = bus:"transport" | train:"transport" .

<destination2> = to metro:"dest" central:"dest"
| to union:"dest" station:"dest"
| to downtown:"dest"
| to national:"dest" airport:"dest" .

After adding the annotations, the sample sentences generated by our grammar look as follows:

Note: These sentence forms are conceptual. It will take programming logic to convert the real output to this form.

1) hello I need to know where."op_locate" the nearest."type"
   police."landmark" station."landmark" is located
2) hello excuse me where."op_locate" is the nearest."type"
   public."landmark" rest."landmark" room."landmark"
3) please tell me the phone."op_phone" number of the
   nearest."type" filling."landmark" station."landmark"
4) where:"op_locate" am I
5) when."op_sched" will the next."type" bus."transport"
   to union."dest" station."dest" arrive
6) excuse me but when."op_sched" will
   the next."type" train."transport"
   to national."dest" airport."dest" arrive
7) help."op_help"
8) I am lost."op_help"
9) excuse me what time is it."op_time_of_day"

Note: Words and annotations are returned in parallel structures through SMAPI. As such, they do not require additional tokenization or parsing, as in the previous examples.

With this set of sentences as input, the task of writing a parser becomes straightforward. No longer is it a job of parsing, it is one of separating the annotated parts of the sentence, which contain the information required by the application, from the nonannotated parts, and placing them into a standard form for processing.

SRCL Syntax

This section describes the syntax for the Speech Recognition Command Language (SRCL).

SRCL is a language that is easy to implement and learn, with a minimum set of constructs to implement a reasonable capability. We have avoided providing multiple ways to accomplish the same thing. This has resulted in a small, consistent language definition that is easy to learn.
The language elements addressed in the syntax are:

- Comment formats
- Terminals
- Nonterminals
- Grammar rules (productions)

These aspects are discussed in the following sections.

**Language Definition**

The following are general language considerations:

- Syntactically significant characters such as ", =, <, >, and . can be used to represent words and annotations if they are preceded by a \ escape character.

- Blanks can be used within a quoted string or a quoted annotation. The characters " and \ may be included within quoted strings and nonterminals if they are preceded by a \ escape character (that is, \" and \, respectively).

- The \ escape character may be used in a quoted terminal that is continued on the next line, with white space on the next line being significant.

- White space is not significant except within quoted terminal and variable definitions and to separate unquoted words.

- SRCL is a case-sensitive language and its string-handling syntax is modeled after the C language.

**Language Elements**

**Comment Formats**

SCRL accepts both the double-slash "/" and the semicolon ";" as comment markers, which occur at the beginning of a line or within the line. All characters up to the next new line are then taken as comment characters.

**Terminals**

For terminology we use "variable" and nonterminal as well as "word" and terminal for the basic elements of the language in the following discussions.

Terminals can be single spoken units (such as "San Francisco") that are optionally enclosed in quotation marks.

This syntax implies that a production such as:

\[ <var> = Name that Tune. \]

or
<var>="Name" "that" "Tune".

uses three separate words for name, that, and tune, and they are pronounced as "n ey m", "th ae td", and "t uw n".

A production such as:

<var>="Name that Tune".

uses a single phrase for the three words that are pronounced "n ey m th ae td uw n" and processed as a single word.

Note: Productions like "Name that Tune" should be used with caution. The primary intent of "quoted phrases" in grammars is to override the space-delimited tokenization of spoken words that include blanks, such as "Boca Raton" or "Van Dyke." You can use "quoted phrases," but understand that the user won't be able to pause between the words in the phrase, nor will the user be able to mumble between the words in the phrase. Also, "name" "that" "tune" will be returned as a recognized phrase message and word structures, potentially with three different annotations. "Name that Tune" will be returned as one word structure, with only one annotation.

Nonterminals

The definition of a nonterminal consists of a mandatory set of angle brackets enclosing an unquoted word identifier, with case being significant. Nonterminals should not cross line boundaries. Syntactically significant characters may be used in nonterminals without the \ escape character.

Examples of distinct nonterminals include:

<nonterminal>
<Nonterminal>
<this_is_a_nonterminal>

Grammar Rules (Productions)

The grammar rule format is given as follows:

1. A simple and consistent format for variables (nonterminals) of the form <id>, where the "<" and "">" are mandatory, and id is in the format of an unquoted word identifier.

2. Tokens, or words, may be represented as identifier strings without the need for quotation marks. It is sometimes necessary to delimit the words with quotation marks to include imbedded blanks.

3. The "=" symbol to separate the left-hand side ( lhs) and right-hand side ( rhs) clauses of grammar rules.

4. The "?" operator to define optional words and phrases. "?" marks the symbol to its immediate left (with no embedded blanks) as optional. If the symbol is a nonterminal, the entire phrase or set of phrases defined by the nonterminal becomes optional.

5. A repetition operator set to include:
   1. + for one or more occurrences of the preceding word or variable
   2. * for zero or more occurrences of the preceding word or variable
Any repetition operator can follow either a word identifier or a variable identifier used on the right-hand side of a rule. Repetition operators are not supported on the left-hand side of the rules.

6. Annotations that can be represented as character strings in the form of word identifiers, which would include decimal numbers or quoted strings like "this is an annotation" or "this is another annotation." The form of the annotation is given as:

   word identifier: annotation identifier

7. Variables can be defined only once. Multiple rules written with the same left-hand side cause an error.

8. The root production carries a special designator in that the left-hand side uses a double bracket construct as follows: <<root_name>>= a | <b> | c | ... . The root production does not need to be the first production in a grammar file. Also, you do not need to have a <<...>> root production. In this case, the first nonterminal becomes the root production.

9. Each grammar rule must be terminated with a period ".".

An example grammar rule might be as follows:

<transaction>=Buy <amount> "of" <companies> "at market".

Note that there must be a dictionary entry that provides a pronunciation for "at market" or, for example, "Amalgamated Rhubarb," if that were a word defined in <companies>.

Note: SRCL supports letters and digits and all national language characters in code page 1252.

Extensions to SRCL - Specifying Weights

In ViaVoice, the SRCL language is extended to support the weighting of alternatives. Specifying weights within a grammar can improve recognition accuracy when reliable information about the relative frequency of different words or phrases is available.

Weights are specified within a BNF by preceding any item in a list of alternatives by +w (where w is a positive floating point number). The value specified indicates the relative likelihood of a token or series of tokens. For example:

<a> = +.1 x y z | +.9 k.

In this example, the phrase "k" (consisting of single token k) is more likely to occur than the phrase "x y z."

Weights can be specified immediately following a =, | or (. There must not be any spaces between + and the weight. For example:

<a> = +.1 x y z | +.9 k.
<b> = c | +5 <a> | +3 (+.1 a b | +1.5 c) | x.

Weights are optional. The default weight is 1. Weights may be specified for none, some, or all of the terms in a list of alternatives.

Weights are best chosen based on real data representative of typical usage patterns. For example, weights can be used to indicate the frequency count of words or phrases as it is observed in actual use.
Note that weights need not sum to 1 in a list of alternatives. This is useful during grammar maintenance: You can change the weight of a token without having to change the weights of all the other tokens to maintain a sum of 1.

The normalization of weights to probabilities takes place locally for each list of alternatives (right-hand side or parenthesized expression) before they are propagated. Thus, the following three rules are equivalent:

\[
\begin{align*}
\langle a \rangle &= +1 \ w \ (+1 \ x \ | \ +4 \ y) \ | \ +4 \ z. \\
\langle b \rangle &= .2 \ w \ (+.2 \ x \ | \ +.8 \ y) \ | \ +.8 \ z. \\
\langle c \rangle &= +.04 \ w \ x \ | \ +.16 \ a \ y \ | \ +.8 \ z.
\end{align*}
\]

Weights are ignored unless -n is specified when the grammar is compiled. This instructs the compiler to use the weights, either explicitly or implicitly defined, and not to equalize arc probabilities from each state. For more information on using the grammar compiler with weights, see Using VTBNFC from the Command Line.

### Using the Translation Facility

The translation facility enables the application developer to associate a translation with any expression in the Extended BNF grammar. The grammar writer associates a translation with a BNF expression by writing the symbol "->" followed by the translation after the expression.

Consider for example the following rule:

\[
\langle <<\text{start>>> } \rangle = \text{my dog has fleas } \rightarrow \text{ scratch.}
\]

This rule defines a grammar that accepts exactly one phrase, "my dog has fleas", and produces the string "scratch" as a translation. A translation may be given any place where a sequence of expressions is specified, including each alternative in a list of alternatives. Consider for example the following rule:

\[
\langle <<\text{start>>> } \rangle = \text{my dog has fleas } \rightarrow \text{ scratch} \\
| \text{my dog is happy } \rightarrow \text{ wag.}
\]

This rule defines a grammar that accepts two phrases, each with a different translation ("scratch" and "wag" respectively).

In addition to literal strings, as in the preceding examples, the translation of a sequence of BNF expressions may include the translations of the constituent expressions. This is done by marking with braces "{}" the expressions in the sequence of expressions whose translations are to contribute to the translation of the whole sequence, and referring in the translation to the nth marked expression by using the string "{n}". So for example the following grammar

\[
<\text{pet}> = \text{dog} \ | \ \text{cat} \ | \ \text{fish}. \\
\langle <<\text{start>>> } \rangle = \text{my} \ <\text{pet}> \ \text{has fleas } \rightarrow \ \{1\} \ \text{scratches} \\
| \text{my } <\text{pet}> \ \text{is happy } \rightarrow \ \{1\} \ \text{wags.}
\]

defines a grammar that accepts six phrases; the input "my dog has fleas" produces the translation "dog scratches", while the input "my fish is happy" produces the translation "fish wags", and so on.

The same grammar can be expressed more succinctly as:

\[
<\text{pet}> = \text{dog} \ | \ \text{cat} \ | \ \text{fish}.
\]
Working with the Details

Every expression in the Extended BNF has a translation. Consider the following example:

<digit> = one -> 1 | two -> 2.
<<number>> = <digit> (hundred -> 00).

This grammar accepts the phrases “one hundred” and “two hundred” and produces the translations “1 00” and “2 00”, respectively, as per the following rules:

- The translation of a terminal is the terminal itself, although this may be overridden by treating any given occurrence of the terminal as a sequence of length one and specifying a translation using the "->" notation, as in "one -> 1" and "two -> 2" above. If necessary, the expression can be enclosed in parentheses as in "(hundred -> 00)" above.

- The translation of a list of alternatives is the particular alternative actually used in parsing the input. Thus the translation of the expression "one -> 1 | two -> 2" in the example above is "1" or "2" according to whether the first or second alternative is used in parsing the input.

- The translation of a non-terminal is the translation of the expression on the right-hand side of the BNF statement defining the non-terminal. Thus in the example above the translation of <digit> is the translation of "one -> 1 | two -> 2".

- Finally, the default translation of a sequence of expressions is the concatenation of the translations of the expressions in the sequence, separated by spaces. Thus the translation of <<number>> is either "1 00" or "2 00". This of course may be overridden by explicitly specifying a translation as described previously. For example, if the definition of <<number>> is changed as follows:

<digit> = one -> 1 | two -> 2.
<<number>> = {<digit>} {hundred -> 00} -> {1}{2}.

then the translation of <<number>> becomes "100" or "200", because an explicit translation "{1}{2}" is given which does not include a space.

Translation Example

The following example is taken from a simple travel expense reporting application. The first part of the example is a grammar for English numbers up to six digits. Each of the rules <n-digit> specifies all way of saying numbers up to n digits, and produces as a translation exactly n digits, using leading zeros if necessary. It is assumed that the application will strip the leading zeros if desired.

<1-digit> = oh -> 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 .

<2-digit> = oh? {<1-digit>} -> 0{1} |
10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
The following rule uses the number rules given above to specify amounts of U.S. currency:

<money> = {<number>}
  | {<number>} dollars and? {<2-digit>} cents? | "" -> 00 -> ((1).(2))
  | {<number>} point? {<2-digit>} -> (1).(2).

The following rules accept English dates and produce a translation in the form yy/mm/dd. For purposes of illustration, the grammar accepts only a limited number of years, and supplies 1996 as a default year.

<day> =
  1st -> 01 | 2nd -> 02 | 3rd -> 03 | 4th -> 04 | 5th -> 05 | 6th -> 06 |
  7th -> 07 | 8th -> 08 | 9th -> 09 | 10th -> 10 | 11th -> 11 | 12th -> 12 |
  13th -> 13 | 14th -> 14 | 15th -> 15 | 16th -> 16 | 17th -> 17 | 18th -> 18 |
  19th -> 19 | 20th -> 20 | 21st -> 21 | 22nd -> 22 | 23rd -> 23 | 24th -> 24 |
  25th -> 25 | 26th -> 26 | 27th -> 27 | 28th -> 28 | 29th -> 29 | 30th -> 30 |
  31st -> 31.

<month> =
  January -> 01 | February -> 02 | March -> 03 | April -> 04 |
  May -> 05 | June -> 06 | July -> 07 | August -> 08 |
  September -> 09 | October -> 10 | November -> 11 | December -> 12.


<date> = {<month>} {<day>} -> 96/{1}/{2}
  | the {<day>} of {<month>} -> 96/{2}/{1}
  | {<month>} {<day>} {<year>} -> {3}/{1}/{2}
  | the {<day>} of {<month>} {<year>} -> {3}/{2}/{1}.

For information about translation parameters for the grammar compiler, refer to the Grammar Compiler - VTBNFC.
Developing Grammars Using the Administration Tools

The ViaVoice Telephony Tools provide a set of web-based tools to build your application grammars and pronunciations. These tools also let you manage all of the telephony processes on the system. There are two primary functions available through the Administration Tools: Status and Build. The Status functions allow you to view the status of running processes as well as status of the current build. The Build tools enable you to compile your grammars and generate the associated pronunciations.

Note: To use the Administration Tools, your web browser should be enabled for JavaScript and it should accept cookies. Also, if you want to listen to text-to-speech generated baseforms, you need to install plugins for .WAV files.

To access these tools, click Administration from the start menu. You see the Start/Stop status web page. It displays the current status of all telephony processes.

Building your Application Grammars

Use the Build web pages to compile your grammars and generate pronunciations for your application. You will need to perform the following tasks:

1. Select your application.
2. Generate your grammar (subscriber.db format only) or compile your grammar (BNF format).
3. Generate baseforms for your grammar.
4. Edit baseforms as necessary. Compile the pronunciation pool for your baseforms.

Selecting Your Application

First select Build then Applications. You will see all of the applications in the directory structure <VTT_HOME>/app/appname, where appname is the name of each application. If you have added any applications to this directory structure, they appear in the list:
In this example, we are working with the **kim** application. Click the radio button next to the kim application and click **Set application**.

**Generating Grammars**

Grammars can be defined as text files using the subscriber.db format, or they can be defined using SRCL in BNF files. Grammars defined in the subscriber.db format need to have a corresponding BNF file generated and the BNF file compiled into an FSG; grammars already in BNF format need only to be compiled into FSGs.

**Generating Grammars from subscriber.db files**

The next step in the build process is to compile the grammars for your application. The grammar for the kim application is defined as follows:

```
team 1 kim
team 2 jackie
team 3 sunil
```
team 4 joe
team 5 sharon
team 6 henry
team 7 ming
team 8 alan

The file **kim.txt** contains this grammar. It is in the **subscriber.db** format and resides in the `<VTT_HOME>\app\kim` directory.

To generate the BNF file and compile the BNF into an FSG, enter **kim.txt** as the input file. There are no INTROs or OUTROs associated with this grammar, so leave that field blank. Select **Generate and compile grammars**.

You should see a message that indicates that one BNF file (**kim.team.bnf**) and one FSG file (**kim.ours.fsg**) have been generated:
Note: If you have not already created a subdirectory for your application in the \<VTT_HOME>\app\<language-code> directory, this step creates it for you.

Generating Grammars from BNF files

If the grammar is defined as a BNF, select Compile Grammars. You can choose to compile all of the grammars associated with an application, or selected ones.
Generating Baseforms

The next step is to generate the pronunciations for the grammar. To do this, click **Generate baseforms**. You see status of the last time that baseforms were generated for this application:
Click **Generate and publish pools.** This action performs three functions:

1. Baseforms are generated for all of the words in your application.

2. Baseforms are compiled into the pronunciation pools that the ViaVoice engine uses to recognize words. The compiled pool files are copied to the ViaVoice engine pools directory which is `<ViaVoice>/vocabs/langs/<language-code>/pools`.

You see a confirmation message that your baseforms and pronunciation pools have been generated and published.

Note that all ViaVoice recognition engines must be restarted for the new pools to be seen by the engine. This can be done by using the **Start/Stop** web page.

**Editing Baseforms**

If the system does not understand the pronunciation of a name or word, you may need to modify or customize the baseform pronunciation for your application. Prior to editing the baseforms, read the section on **Phonetic Pronunciation.** To edit baseforms, click **Edit baseforms:**
The Editing Baseforms web page allows you to add new baseforms and change existing ones. All of the words (spellings) in your application’s baseforms file are displayed. For each word (spelling) in the baseforms file, a Sounds Like spelling and its associated Baseform is displayed.

Typically, the baseforms file is large and cannot be viewed in its entirety on a single web page. To change the content and amount of information that is displayed at one time, do the following:

- By default, 20 words are displayed at a time on the page. To change this, update the Lines to show field.

Words are displayed alphabetically. You can specify the starting point at which to display the words by changing the Next segment to display begins with field.

To edit an existing baseform, do the following:

Select the word (spelling) by clicking in the Sounds Like field.
Click **Listen to Baseform**. You will hear the current pronunciation for that spelling. If the pronunciation is correct, but a different pronunciation is also acceptable, click **Add Baseform**. If the pronunciation is completely inaccurate, click on **Modify Baseform**.

In the **Sounds Like** field for that spelling (or in the blank **Sounds Like** field, if you clicked **Add Baseform**), type a phonetic sounds-like spelling for the word. For example, if the spelling is "Leila" and the preferred pronunciation uses a long "A" sound in the first syllable (as in "Layla") as opposed to the default pronunciation ("Leela," with the long E sound), type "Layla" into the **Sounds Like** field. The spelling may look incorrect, but this helps the system make the proper vowel and consonant associations.

Check the pronunciation by clicking **Listen to Baseform**. You may need to change the **Sounds Like** spelling several times to get just the right pronunciation.

When you are satisfied with the way the baseform sounds, click **Save**. (Note that **Save** saves the new baseform and leaves you on the same page. **Save & Next** saves the baseform and displays the next 20 words in the list. **Save & Back** saves the baseform and displays the previous 20 words in the list. Use these last two options when you are changing multiple baseforms).

**Note:** While you are editing baseforms, you can return to the stored sounds-like spellings and baseforms by using the **Undo** and **Undo All** buttons. To return to the stored values for a single word, select the word first and click **Undo**. To return to the stored values for all of the baseforms you changed, click **Undo All**. You are prompted to confirm that you want to undo all of your changes.

### Phonetic Pronunciation

If you encounter ViaVoice Run Time pronunciation errors, these are usually due to the stress placed on the vowel sounds. If you experience pronunciation errors on certain words, then the baseforms must be modified in order to improve speech recognition. Below are two phonetic charts to assist you in correcting baseform phonetic sounds.

#### Guide to "Sounds Like" Spellings

<table>
<thead>
<tr>
<th>Sound</th>
<th>Symbol</th>
<th>As in</th>
<th>Suggested</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ae&quot;</td>
<td>AE</td>
<td>add, cat, pan</td>
<td>ae</td>
<td>The short &quot;a&quot; sound.</td>
</tr>
<tr>
<td>&quot;ah&quot;</td>
<td>AA</td>
<td>Are, drop, content</td>
<td>ah</td>
<td>Rarely wrong but sometimes an open ‘o’ sound is used instead of the ‘ah’. So spell ‘mot’ as ‘maht’ to change it from ‘moat’ to ‘mott’</td>
</tr>
<tr>
<td>&quot;uh&quot;</td>
<td>AH</td>
<td>One, come</td>
<td>‘Uh’ or double the following letter</td>
<td>Very rarely wrong; sometimes an unstressed sound is used instead. So spell ‘Merom’ as ‘Meromm’</td>
</tr>
<tr>
<td>Sound</td>
<td>Symbol</td>
<td>As in</td>
<td>Suggested</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>---------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Short &quot;o&quot;</td>
<td>AO</td>
<td>All, off, caution</td>
<td>or</td>
<td>Very rarely wrong. Try spelling ‘Audi’ as ‘Ordi’.</td>
</tr>
<tr>
<td>au</td>
<td>AW</td>
<td>How, about</td>
<td>Ou, au, ow</td>
<td>If ‘ow’ is mispronounced as ‘oh’ try switching to ‘au’ or ‘ou’.</td>
</tr>
<tr>
<td>eh</td>
<td>AX</td>
<td>enable, moment</td>
<td>eh</td>
<td>The unstressed &quot;eh&quot; sound, as in &quot;moment&quot; or &quot;enable&quot;, but <strong>not</strong> as in &quot;enter.&quot;</td>
</tr>
<tr>
<td>er</td>
<td>AXR</td>
<td>another, over, or,</td>
<td>er</td>
<td>The unstressed &quot;er&quot; sound, as in &quot;another,&quot; over, or &quot;or,&quot; but <strong>not</strong> as in &quot;her &quot;</td>
</tr>
<tr>
<td>Long &quot;i&quot;</td>
<td>AY</td>
<td>Fire, why, eye, Y, ai,</td>
<td>Y, ai, add an e</td>
<td>Rare but if ‘Byron’ is mispronounced more like &quot;Birron&quot; then try spelling it as ‘Bairon’ or &quot;Byeron&quot; or even ‘Byre-on’</td>
</tr>
<tr>
<td>eh</td>
<td>EH</td>
<td>Enter, pleasant</td>
<td>eh</td>
<td>Rare but replacing the vowel letter(s) by ‘eh’ usually works. E.g., If ‘pleasant’ were mispronounced as &quot;pleasant&quot; then ‘plehsant’ corrects it.</td>
</tr>
<tr>
<td>er</td>
<td>ER</td>
<td>Bird, turn, her</td>
<td>er</td>
<td>Never observed but ‘er’ should work.</td>
</tr>
<tr>
<td>Long &quot;a&quot;</td>
<td>EY</td>
<td>Train, eight, A</td>
<td>Ai, ay, ey</td>
<td>If ‘Wacome’ is mispronounced &quot;Wahcome&quot; then one of ‘Waicome’, ‘Waycome’ or ‘Weycome’ should</td>
</tr>
<tr>
<td>Sound</td>
<td>Symbol</td>
<td>As in</td>
<td>Suggested</td>
<td>Comments</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>----------------------------</td>
<td>-----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Short "i" | IH     | If, it, pick               | ih        | The short I sound as in "it," "if," or "pick, but not as in "lasting."
|         |        |                            |           | ‘Piro’ should be pronounced with a short I and is mispronounced as "pyro" then ‘pihro’ should work.                                     |
| "ix"    | IX     | discuss, saving, budgeted, decided | ix        | The unstressed short "i" sound, as in "discuss," "budgeted," "decided," or "saving."                                                      |
| Long "e" | IY     | Any, he, obvious           | Ee, ey, ie| If ‘Matrio’ was mispronounced as "mah-try-oh" then ‘matreeo’, ‘matreeoh’ or ‘mahtreeoh’ should fix.                                    |
| Long "o" | OW     | Quote, open, go            | Oh, ow, oe| ‘Bow’ should be right for ‘Beau’ and ‘Dohl’ should work for ‘Dole’                                                                       |
| oi      | OY     | Avoid, enjoy, dive, why    | oy        | The long "i" sound (or oi sound) as in "avoid," "employ," "dive," or "why."                                                              |
| Short "u" | UH     | Good, full, could, put     | uh        | The short "u" sound, as in "good," "put," "full," or "could.” If ‘Clough’ should be pronounced as in "cluf" then ‘cluhf’ should work. |
| Long "u" | UW     | tooth, blue, view, use, you| Oo, yoo   | The stressed long "u" sound, as in "to," "use," "you," or "veiw." Try ‘Bloomehn’ for the                                                  |
Usually non-vowel sounds will not be a problem for the Run Time baseform engine, unless the name is of foreign extraction (as in the British pronunciation of Schedule (Schxhule). Below are the consonant sounds if you need to select a different sound than that produced by the automatic pronunciation rules.

### Guide to Consonant "Sounds Like" Spellings

<table>
<thead>
<tr>
<th>Sound</th>
<th>As in</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;b&quot;</td>
<td>be, able</td>
<td>The &quot;b&quot; sound as in &quot;be&quot; or &quot;able&quot;</td>
</tr>
<tr>
<td>&quot;bd&quot;</td>
<td>sob, tab, blurb</td>
<td>The hard &quot;b&quot; sound at the end of a word.</td>
</tr>
<tr>
<td>&quot;ch&quot;</td>
<td>cheap, child</td>
<td>The soft &quot;c&quot; sound before &quot;h&quot;. The &quot;ch&quot; sound as in &quot;cheap&quot;.</td>
</tr>
<tr>
<td>&quot;d&quot;</td>
<td>and, David</td>
<td>The &quot;D&quot; sound as in &quot;David&quot; or &quot;and&quot;.</td>
</tr>
<tr>
<td>&quot;dd&quot;</td>
<td></td>
<td>The &quot;D&quot; sound at the end of a word.</td>
</tr>
<tr>
<td>&quot;dh&quot;</td>
<td>the, either</td>
<td>The voiced &quot;th&quot; sound, as in &quot;the&quot; or &quot;either&quot;, but NOT as in &quot;thesis&quot; or &quot;thing&quot;.</td>
</tr>
<tr>
<td>&quot;dx&quot;</td>
<td>butter, greater, regarding, order</td>
<td>The very short T or D sound, as in &quot;butter&quot;, &quot;greater&quot;, &quot;regarding,&quot; or &quot;order.&quot;</td>
</tr>
<tr>
<td>&quot;d$&quot;</td>
<td></td>
<td>A dummy sound that marks the beginning and end of a sentence.</td>
</tr>
<tr>
<td>&quot;f&quot;</td>
<td>father, rough</td>
<td>The soft &quot;f&quot; sound.</td>
</tr>
<tr>
<td>Sound</td>
<td>As in</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>&quot;g&quot;</td>
<td>again, peg</td>
<td>The hard &quot;g&quot; sound.</td>
</tr>
<tr>
<td>&quot;gd&quot;</td>
<td>log, tug</td>
<td>The G sound at the end of a word.</td>
</tr>
<tr>
<td>&quot;hh&quot;</td>
<td>here, who</td>
<td>The &quot;h&quot; sound</td>
</tr>
<tr>
<td>&quot;jh&quot;</td>
<td>jeep, roger</td>
<td>The soft &quot;g&quot; sound.</td>
</tr>
<tr>
<td>&quot;k&quot;</td>
<td>because, dark, scale</td>
<td>The &quot;k&quot; sound, but <strong>not</strong> the J sound at the beginning of a word.</td>
</tr>
<tr>
<td>&quot;kd&quot;</td>
<td>cake</td>
<td>The &quot;k&quot; sound at the end of a word.</td>
</tr>
<tr>
<td>&quot;L&quot;</td>
<td>level, parallel</td>
<td>The &quot;L&quot; sound</td>
</tr>
<tr>
<td>&quot;m&quot;</td>
<td>am, must</td>
<td>The &quot;m&quot; sound</td>
</tr>
<tr>
<td>&quot;n&quot;</td>
<td>final, not, none</td>
<td>The starting &quot;N&quot; sound or the unaccented N in the middle of a word, as in &quot;final,&quot; &quot;not,&quot; or &quot;none.</td>
</tr>
<tr>
<td>&quot;ng&quot;</td>
<td>bang, think, singing</td>
<td>The &quot;n&quot; followed by a &quot;g&quot; or &quot;k&quot; sound</td>
</tr>
<tr>
<td>&quot;p&quot;</td>
<td>adoption, amps, rapid</td>
<td>The &quot;p&quot; sound, but <strong>not</strong> at the beginning of a word.</td>
</tr>
<tr>
<td>&quot;pd&quot;</td>
<td>pop, tarp, mop</td>
<td>The &quot;p&quot; sound at the end of a word.</td>
</tr>
<tr>
<td>&quot;r&quot;</td>
<td>abroad, brace, read</td>
<td>The &quot;r&quot; sound followed by a vowel sound in the same syllable.</td>
</tr>
<tr>
<td>&quot;s&quot;</td>
<td>sit, circus, decide</td>
<td>The &quot;s&quot; not followed by an &quot;h&quot; sound.</td>
</tr>
<tr>
<td>Sound</td>
<td>As in</td>
<td>Comments</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>&quot;sh&quot;</td>
<td>action, shade, splash</td>
<td>The &quot;sh&quot; sound</td>
</tr>
<tr>
<td>&quot;t&quot;</td>
<td>adapter, retry, let</td>
<td>The &quot;t&quot; (not &quot;th&quot;) sound not at the beginning of a word</td>
</tr>
<tr>
<td>&quot;td&quot;</td>
<td>date, quit,</td>
<td>The &quot;t&quot; sound at the end of a word</td>
</tr>
<tr>
<td>&quot;th&quot;</td>
<td>thesis, thing</td>
<td>The unvoiced &quot;th&quot; sound, but NOT as in &quot;the&quot; or &quot;either&quot;.</td>
</tr>
<tr>
<td>&quot;ts&quot;</td>
<td></td>
<td>The &quot;ts&quot; sound at the end of a word</td>
</tr>
<tr>
<td>&quot;v&quot;</td>
<td>eleven, improve, very</td>
<td>The &quot;v&quot; sound</td>
</tr>
<tr>
<td>&quot;w&quot;</td>
<td>frequent, way, question, anywhere</td>
<td>The &quot;w&quot; sound</td>
</tr>
<tr>
<td>&quot;y&quot;</td>
<td>emulate, yes, senior, you</td>
<td>The &quot;y&quot; sound that leads into a vowel</td>
</tr>
<tr>
<td>&quot;z&quot;</td>
<td>pans, goes, zero</td>
<td>The &quot;z&quot; sound</td>
</tr>
<tr>
<td>&quot;zh&quot;</td>
<td>Asia, pleasure, Zho</td>
<td>The soft &quot;z&quot; sound</td>
</tr>
</tbody>
</table>
Displaying Status

Use the **Status** web pages to display status of telephony processes as well as status of the current build. To display the status of all telephony processes, click **Start/Stop**. The current state of all telephony processes is displayed. You can also start and stop processes from this web page.
Displaying Build Information

You can also view current build errors by selecting **Build Errors**. In this example, there were no errors:
You can view the build history by clicking **Build Log**:
Grammar Development Tools

Use the make utilities to generate grammars, a dictionary of pronunciations, and the pronunciation pools needed by the speech recognition engine. These utilities are:

- **makeGrammars** - To produce .bnf and compiled .fsg files
- **makeBsfms** - To develop a pronunciation dictionary
- **makePol** - To generate the pronunciation pools needed by the speech recognition engine

For these utilities, `<VTT_HOME>` denotes the installation directory for ViaVoice Telephony Run Time and Tools.

Note that there are two ways to run these utilities:

- As Perl scripts - It is essential to specify the `.pl` extension to access and use them.
- As command (.cmd) files - The command file version of each utility invokes perl with the appropriate script. It also allows for redirection of input and output.

If you specify the utility name without specifying the `.pl` extension, the command file runs.

The `makeGrammars` Utility

The `makeGrammars` utility generates and compiles an application’s grammars. It reads the subscriber.db file (and, if it exists, the subscriber.wrap file) and converts each file into a .bnf file. It then compiles the .bnf files into .fsg files.

You can use the environment variable VTCMD_FLAGS to set any of the compiler options. By default, the compiler generates .fsg files that handle utterances where mumbling and other non-speech noises are injected. See The Grammar Compiler for details on the compiler options.

The `makeGrammars` syntax is:

`makeGrammars.pl <app_name> <location of subscriber.db file> <-L Ll_LL>`

where:

- `<app_name>` is the application name.
- `<location of subscriber.db file>` is the complete path specification to the directory in which appname.subscriber.db and optionally, appname.subscriber.wrap, reside. A directory called app_name must exist in `<VTT_HOME>\app\Ll_LL>`.
- `<-L Ll_LL>` specifies the language for which the grammar is being created. By default, the language is U.S. English. Specify this parameter whenever the language of the application is not U.S. English. Ll_LL can be one of the following:

  - **En_US** for U.S. English
  - **En_UK** for U.K. English
• Fr_FR for French
• Gr_GR for German

Refer to Preparing Words and Phrases for Phone Recognition for details on preparing the subscriber files.

The makeGrammars utility can also be run from a command (.cmd) file as follows:

```
makeGrammars <app_name> <location of subscriber.db file> <-L Ll_LL>
```

The makeGrammars command file invokes Perl using the makeGrammars.pl script described above. Running makeGrammars from this command file also allows for redirection of input and output.

---

**The makeBsfms Utility**

The makeBsfms utility generates a single dictionary of pronunciations (baseforms) for all of an application's grammars. The command syntax for makeBsfms is:

```
makeBsfms.pl <app_name> <-L Ll_LL>
```

where:

- `<app_name>` is the application name.
- `<-L Ll_LL>` specifies the language for which the baseforms are created. By default, the language is U.S. English. Specify this parameter whenever the language of the application is not U.S. English. Ll_LL can be one of the following:
  - En_US for U.S. English
  - En_UK for U.K. English
  - Fr_FR for French
  - Gr_GR for German

It reads the compiled grammars (.fsg files) from the

```
<VTT_HOME>\app\<Ll_LL>\<app_name>\grammars
```

subdirectory. If present, it searches

```
<VTT_HOME>\app\<Ll_LL>\<app_name>\baseforms\baseforms
```

This file is created the first time makeBsfms is run for an application. makeBsfms then identifies the words and phrases that need pronunciations and then creates these pronunciations.

You can now use the Baseforms Editor to customize your pronunciations, if needed. See Editing Baseforms.

The makeBsfms utility can also be run from a command (.cmd) file as follows:

```
makeBsfms <app_name> <-L Ll_LL>
```

The makeBsfms command file invokes Perl using the makeBsfms.pl script described above. Running makeBsfms from this command file also allows for redirection of input and output.
The makePol Utility

The makePol utility generates the pronunciation pools that are used by the speech recognition engine. Note that makePol can be run only if makeBsfms completes successfully. The command syntax for makePol is:

```
makePol.pl <app_name> <-L Ll_LL>
```

where:

- `<app_name>` is the name of your application.
- `<-L Ll_LL>` specifies the language for which the pools are created. By default, the language is U.S. English. Specify this parameter whenever the language of the application is not U.S. English. `Ll_LL` can be one of the following:
  - En_US for U.S. English
  - En_UK for U.K. English
  - Fr_FR for French
  - Gr_GR for German

The makePol utility processes the baseforms file found in the `<VTT_HOME>\app\<Ll_LL>\<app_name>\baseforms` directory, generates the pronunciation pool files, and places them into the directory `<viavoice>\app\<Ll_LL>\<app_name>\baseforms\<app_name>`.

You must now make the pronunciation pools known to the engine. This is done by placing the pool files into the pools directory of the ViaVoice Telephony Tools engine. The directory is `<viavoice>\vocabs\langs\Ll_LL\pools`. You can do this by:

1. Copying the files in `<VTT_HOME>\app\<Ll_LL>\<app_name>\baseforms\baseforms` into the engine pool directory. This method is recommended for small vocabularies having less than 200 words. If used for a larger vocabulary, schedule periodic copies of approximately 200 words each.

2. Using the Administration Tools or Process Manager, stop and restart the system. This step is necessary for the engine to recognize the changed pool files.

The makePol utility can also be run from a command (.cmd) file as follows:

```
makePol <app_name> <-L Ll_LL>
```

The command file invokes Perl using the makePol.pl script described above. Running makePol from a command file also allows for redirection of input and output.
The Grammar Compiler - VTBNFC

The grammar compiler, vtbnfc, converts a .bnf grammar file into a finite state grammar (.fsg) file. The .fsg file is used at run time by the speech recognizer to determine which words and phrases a caller can say to a telephony application. Create your grammars as plain-text files with an extension of .bnf. These files must then be compiled for use by the recognition engine.

This section describes how to use the:

- Grammar compiler to create .fsg file(s) from your grammar files using the command line interface.
- Grammar Compiler API to compile grammars from within your application

This section also describes error messages that are output by the grammar compiler.

Using VTBNFC from the Command Line

The syntax of vtbnfc is:

vtbnfc [-n] [-m | -m- | -m+] [-s | -s- | -s+] [-o outfile | -d outdir] [-tr] [-en] grammarfile

where:

- **-n** indicates that the uniform probability computation be turned off. By default, grammars are compiled with what is known as the uniform probability computation, which means that all words out of a given state are equally likely. This option enables the developer to turn the uniform probability computation off. The recognition performance difference between the two options is strongly grammar dependent. An extreme example is:

  `<Number> = "point" <digit> | <digit>
  <digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

  The first word that can be spoken is "point" or a digit. With the uniform probability computation, all initial words have probability 1/11. Without the uniform probability computation, "point" has a probability of 0.5, while each digit has a probability of only 0.05. Without the uniform probability computation, language model probabilities are assigned to be locally uniform for each production rule in the BNF file.

  The -n option is also used to instruct the compiler to use the weights that are explicitly or implicitly defined in the BNF. For more information on specifying weights in BNF files, see Extensions to SRCL - Specifying Weights.

  Note the specifying -n may cause some generated FSGs to be larger since less optimization is possible when probabilities must be taken into account in comparing FSGs.

- **-m** is on or off. Use this option to generate an FSG file that handles cases where the end user injects mumbling or other non-speech noise while speaking. Instructs the compiler to enable mumble words. Use -m to tell the speech engine to enable mumble words, but not to return the mumble text. Use -m+ to tell the speech engine to enable mumble words and to return the mumble text. Use -m- to disable mumble words. The default is -m-.
-s Use this option to generate an FSG file that handles cases where the caller pauses briefly while speaking a command. Instructs the compiler to enable silence words. Use -s to tell the speech engine to enable silences, but not to return silence words. Use -s+ to tell the speech engine to enable silences and to return silence words. Use -s- to disable silence words. The default is -s.

-o outfile is the name to be given to the .fsg file. By default, the grammar compiler uses the file name of the grammar file and adds an extension of .fsg.

-d outdir is the name of the directory in which the compiler places multiple .fsg files, one for each root. By default, if a BNF grammar file contains multiple roots marked by <<>> notation, the grammar compiler returns an error. Use this option to enable multiple roots to be specified in a single BNF file. For example, if the BNF file contains:

```
<1> = ...
<2> = ...
<3> = ...
<<a>> = <1> | <2>
<<b>> = <1> | <3>
<<c>> = <2> | <3>
```

Using the multiple roots option, the grammar compiler generates a.fsg, b.fsg, and c.fsg, corresponding to the root rules <<a>>, <<b>>, and <<c>>. There are no restrictions on the ordering of multiple roots within the .bnf file.

-tr generates an fsg file usable only by the translation API. This fsg file will not be accepted by the engine. For more information about translation rules, refer to Using the Translation Facility.

-en generates an fsg file without translations for the engine. If the grammar contains translations, you must specify either -tr or -en when you compile the grammar. This means that you compile the grammar twice: once with -tr, producing an fsg to be provided to the translation API (VtLoadGrammar), and once with -en, producing an fsg to be provided to the engine (SmDefineGrammar). For more information about translation rules, refer to Using the Translation Facility.

grammarfile is the name of the .bnf grammar file to be compiled. The grammar file name must be fully qualified if it does not reside in the current directory.

Compiling a Grammar

To compile a grammar:

- Open a DOS session and switch to the directory that contains the grammars you want to compile.
- Type `vtbnfc` followed by any optional parameters and the name of the grammar file to be compiled. Press the Enter key.
- If the syntax of the command has errors, command-line syntax help is displayed; otherwise, the grammar compiler runs. If the grammar has errors, error messages are displayed.
- To redirect the compilation to an output log, add " > filename " to the end of the command. For example, `vtbnfc active.bnf > active.log`
Example:
The following example uses vtbnfc to compile the grammar file, active.bnf, and enables mumble words. The output file name is active.fsg.

vtbnfc -m active.bnf

Using the Grammar Compiler API

The Grammar Compiler application programming interface (API) is a set of C-language functions that allows developers to compile grammars from within their applications. The Grammar Compiler API is intended for use in command and control applications. The Grammar Compiler API is provided as a separate DLL (vtbnfc31.dll) located in the ViaVoice\bin directory.

To compile applications that use the Grammar Compiler API, include the header file, vtbnfc.h, which contains all of the necessary definitions. Be sure to include VTBNFC.H after the standard Windows include files. Also, depending on your build environment, you may need to set the INCLUDE environment variable to point to the path where vtbnfc.h is located. Likewise, you need to include vtbnfc31.lib to the link step of your product build. If your application uses the Grammar Compiler API, you need to include the DLL with your application.

Note: You must install the ViaVoice SDK to use the Grammar Compiler API. The SDK includes all of the files you need to use these APIs.

Grammar Compiler Function Calls

Following are the function calls provided as part of the Grammar Compiler DLL:

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VtAddArg</td>
<td>Adds a compiler argument</td>
</tr>
<tr>
<td>VtCompileGrammar</td>
<td>Compiles a .bnf file and produces a .fsg file</td>
</tr>
<tr>
<td>VtGetMessage</td>
<td>Returns any error messages</td>
</tr>
<tr>
<td>VtSetArg</td>
<td>Fills in a compiler argument value</td>
</tr>
</tbody>
</table>

VtAddArg

```c
void VtAddArg (VtArg  arg,
               long   index);
```

Parameters:
arg input - The argument.

index output - The index into the argument structure. VtAddArg increments the index by one.

name input - The name of the attribute.

value input - The value of the attribute.

Returns:
None

Remarks:
VtAddArg is a macro that adds an argument, with the specified attributes, to the end of a VtArg structure. This function sets the components of the arg parameter. The pointer to arg or to a list of similarly created arguments can then be passed to VtCompileGrammar. VtAddArg increments an index to point to the last argument in the argument list.

VtCompileGrammar

```c
int VtCompileGrammar( int nargs,
                       VtArg *args );
```

Parameters:

nargs input - The number of arguments in the argument list.

args input - The pointer to an argument structure that indicates the Grammar Compiler parameters to be used.

Returns:

0 - Successful.

Other - Use VtGetMessage to obtain more detailed error information.

Remarks:

VtCompileGrammar compiles a .bnf file and produces an .fsg file using the specified parameters. The Grammar Compiler parameters are provided by setting up an argument list (VtArg). See xxx for more information on creating and using the VtArg structure.

VtGetMessage

```c
int VtGetMessage( char **message);
```

Parameters:

message input - The pointer to a character string which holds the error information.

Returns:

None

Remarks:
VtGetMessage returns a pointer to a message string that describes the errors encountered during the last VtCompileGrammar call that returned a non-zero error code. The messages returned by VtGetMessage are exactly the same messages (errors and warnings) that are generated by the Grammar Compiler and can be used to determine the cause of the errors and appropriate action to be taken by the application.

**VtSetArg**

```c
void VtSetArg (VtArg  arg,
        char  *name,
        long   value);
```

**Parameters:**

- **arg** input - The argument.
- **name** input - The name of the attribute.
- **value** input - The value of the attribute.

**Returns:**

None

**Remarks:**

VtSetArg is a macro that fills a VtArg structure with the specified attributes. This function sets the components of the `arg` parameter. The pointer to `arg` or to a list of similarly created arguments can then be passed to VtCompileGrammar. The argument structure specifies the input BNF file and, optionally:

- The output FSG file
- Whether the non-uniform probability computation is to be used for the grammar
- The output FSG directory, if multiple roots are supported in the grammar
  Whether embedded silences and mumbles are allowed within the words or phrases in the grammar

**Grammar Compiler API Data Types**

The following data type is used by the Grammar Compiler APIs.

**VtArg**

Argument structure for VtCompileGrammar.

```c
typedef struct {
    char  *name;
    long  value;
} VtArg;
```

Where:
name Name of a VtCompileGrammar argument.

value Value of the VtCompileGrammar argument if the size of the argument is less than or equal to the size of (long); otherwise, a pointer to the VtCompileGrammar argument value.

Possible arguments for VtSetArg (and VtAddArg) are:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VtNbnfFile</td>
<td>The input BNF file.</td>
</tr>
<tr>
<td>VtfsgFile</td>
<td>The output FSG file. This is equivalent to -o on the command line.</td>
</tr>
<tr>
<td>VtNfsgDirectory</td>
<td>The output directory for multiple roots. This is equivalent to the -d parameter on the command line.</td>
</tr>
<tr>
<td>VtNequalizeArcProbabilities</td>
<td>The non-uniform probability computation. When set to 1, arc probabilities are equalized. (This is the default if not specified.) When set to 0, arc probabilities are not equalized. This argument is equivalent to the -n parameter on the command line.</td>
</tr>
<tr>
<td>VtNfsgFlags</td>
<td>SmDefineGrammar flags. The possible flag values are:</td>
</tr>
<tr>
<td></td>
<td>SM_PHRASE_ALLOW_SILENCES, SM_PHRASE_SHOW_SILENCES, SM_PHRASE_NO_SILENCES,</td>
</tr>
<tr>
<td></td>
<td>SM_PHRASE_ALLOW_INSERTIONS, SM_PHRASE_SHOW_INSERTIONS, SM_PHRASE_NO_INSERTIONS.</td>
</tr>
</tbody>
</table>

The following example illustrates how to fill in a VtArg structure before calling VtCompileGrammar and how to use VtGetMessage if the return from VtCompileGrammar is not equal to zero:

```c
int n = 0;
VtArg args[10];
VtAddArg( args, n, VtNbnfFile, "mygrammar.bnf" );
VtAddArg( args, n, VtNfsgFlags, SM_PHRASE_SHOW_SILENCES | SM_PHRASE_ALLOW_INSERTIONS);

// Additional parameters specified by additional VtAddArg calls

int rc = VtCompileGrammar( n, args );
if ( rc != 0 )
{
    char *message;
    VtGetMessage( &message );
    printf( message );
}
```

Grammar Compiler Error Messages

The following Grammar Compiler error messages may be displayed:
<table>
<thead>
<tr>
<th>Error Message Syntax</th>
<th>Error Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;can't open '%s' (%s)\n, strerror (errno)&quot;</td>
<td>The compiler cannot open the filename you specified. The (%s) gives an error message.</td>
</tr>
<tr>
<td>&quot;invalid escape sequence&quot;</td>
<td>An invalid escape sequence (beginning with a backslash) was entered.</td>
</tr>
<tr>
<td>&quot;syntax error in &quot;%s&quot;, line %d, column %d&quot;</td>
<td>This error message gives the filename, line number, as well as the column number where the offending syntax error occurred. The correct syntax must be input.</td>
</tr>
<tr>
<td>&quot;%s is multiply defined&quot;</td>
<td>&quot;%s&quot; is the name of a non-terminal symbol.</td>
</tr>
<tr>
<td>&quot;%s is not defined&quot;</td>
<td>&quot;%s&quot; is the name of a non-terminal symbol.</td>
</tr>
<tr>
<td>&quot;extern %s is already defined&quot;</td>
<td>&quot;%s&quot; is the name of a non-terminal symbol.</td>
</tr>
<tr>
<td>&quot;cannot use both annotations and translations.&quot;</td>
<td>You must decide whether you want to use either annotations or translations in the grammars.</td>
</tr>
<tr>
<td>&quot;illegal recursive use of %s&quot;</td>
<td>A non-terminal symbol was used recursively.</td>
</tr>
<tr>
<td>&quot;more than 64K words and externs&quot;</td>
<td>You have exceeded the limit of 64K words for fsg file format words. Reduce the number of words or externs to rectify the error condition.</td>
</tr>
<tr>
<td>&quot;more than 64K annotations&quot;</td>
<td>You have exceeded the 64K buffer space for fsg file format annotations. Reduce the number of annotations to rectify the error condition.</td>
</tr>
<tr>
<td>&quot;more than 64K states&quot;</td>
<td>You have exceeded the limit of 64K states for fsg file format states. Reduce the complexity of the grammar to rectify the error condition.</td>
</tr>
<tr>
<td>&quot;no .bnf file specified&quot;</td>
<td>You have not given the name of the .bnf file to be compiled. Enter the name of the .bnf file in order to proceed.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;please specify -tr for translation or -en for engine fsg&quot;</td>
<td>You have not specified which fsg generation option you wish to use (-tr for translation or -en for engine fsg).</td>
</tr>
<tr>
<td>&quot;your grammar has multiple roots - please use -d&quot;</td>
<td>The grammar has multiple root non-terminals (non-terminals are delimited by (&lt;\text{&lt;&gt;})). In this case you must use -d dir to specify a directory in which an fsg for each root rule will be generated.</td>
</tr>
<tr>
<td>&quot;please specify only one of -d and -o&quot;</td>
<td>If you specify a directory name with -d you may not also specify a filename with -o because the filenames are derived from the root rule names.</td>
</tr>
<tr>
<td>&quot;too many args in rule at line %d, col %d&quot;</td>
<td>You have included too many arguments at the specified location (%d) of the translation utility.</td>
</tr>
<tr>
<td>&quot;too many marks in rule at line %d, col %d&quot;</td>
<td>You have included too many &quot;{ }&quot; at the specified location (%d) of the translation utility. The number of portions of a rule marked with &quot;{ }&quot; must not be larger than the largest {n} in the translation.</td>
</tr>
<tr>
<td>&quot;too few marks in rule at line %d, col %d&quot;</td>
<td>You have included too few &quot;{ }&quot; at the specified location (%d) of the translation utility. The number of portions of a rule marked with &quot;{ }&quot; must be as large as the largest {n} in the translation.</td>
</tr>
</tbody>
</table>
Generating Baseforms From Vocabularies

A vocabulary is the set of words, phrases and sentences that a caller is allowed to say and be recognized by the recognition engine. A pronunciation dictionary (or baseforms file) contains the phonetic pronunciations for a vocabulary of words and phrases. Baseforms are usually generated from grammars. However, ViaVoice Telephony Tools and Run Time provides a facility for generating baseforms directly from a text file that does not have to be compiled. This is the method to be used to generate baseforms for a dynamic command vocabulary. A dynamic command vocabulary is a vocabulary that is defined by your application while it is running rather than being defined by a precompiled grammar.

The dynamic command vocabulary must be specified as a simple list of words and phrases in a file with extension .VOC and placed into the directory \(<\text{VTT\_HOME}\><\text{app}\><\text{LL\_LL}\><\text{app\_name}\><\text{grammars}\) where \(\text{LL\_LL}\) is the language for which the application is used and \(\text{app\_name}\) is the name of the application.

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

To generate the baseforms and make them known to the recognition engine, use either the Administration Tools or use the makeBsfms and makePol utilities.

All pronunciations that are needed are generated and placed into \(<\text{VTT\_HOME}\><\text{app}\><\text{LL\_LL}\><\text{app\_name}\><\text{baseforms}\><\text{baseforms}\).
Testing Your Phone Grammars

After you define and compile your grammars and create the necessary pronunciations, you need to test them to verify that all the words and phrases can be recognized by ViaVoice. If ViaVoice cannot recognize selected words or phrases, you might need to modify your grammars or the pronunciations of the words or phrases to resolve the problems.

When testing your grammars, you should test both words and phrases that are included and words and phrases that are not included. If you have more than one grammar for your application, you should test each of them separately, and then test them together. The goal of testing "out-of-vocabulary" words and phrases is to ensure that ViaVoice is not inadvertently recognizing these out-of-vocabulary words and causing functions to be performed that the user did not specify. For more information on how the engine accepts or rejects utterances, refer to "Acceptance or Rejection of Utterances".

Test your vocabulary with a reasonable population of users. This should help you identify if there are words that are consistently misrecognized over a number of users; if there are, you might need to change the word or phrase or add multiple pronunciations. You should test your vocabulary against a variety of factors, including using telephones from various vendors, people with accents, males and females, people who speak the language natively and those who speak it as their secondary language.

Remember that testing your vocabulary is an iterative process. As you make changes to your grammars or the pronunciations, you should go back and retest everything to verify that all of the words and phrases can still be recognized.

Tools for Testing Grammars

The ViaVoice Telephony Tools provides three command-line utilities that can assist you as you design, test, and debug grammars:

- `fsgenum`
- `fsgprint`
- `fsgtest`

The ViaVoice SDK includes a Grammar Test Tool that can also be used to test your phone recognition grammars (although you will need a system with a microphone). For more information, refer to the SMAPI Developer's Guide which is included with the SDK.

fsgenum

The fsgenum tool is used to generate the sentences accepted by a grammar. These generated sentences can be used as test scripts for the grammar or as a reference for what the grammar will accept.

The command line parameters for fsgenum control how the sentences for the grammar will be generated. The fsgenum tool can enumerate all the sentences accepted from a compiled grammar (FSG) to standard output. The command-line options specified below determine which sentences are generated. This tool can be used to produce test scripts for "live" grammar testing. The syntax of fsgenum is:

where:

-r N - Generates N random sets of sentences from the grammar. The same set of random sentences are always generated each time the command is executed with this option. The sentences are chosen in accordance with the probability that is assigned to them by the grammar.

-R N - Generates N a new random set of sentences from the grammar. A different set of random sentences is generated each time the command is executed with this option.

-f - Generates a complete set of sentences acceptable by the grammar. For some grammars, this set might be very large.

NOTE: If none of the preceding options is specified, fsgenum will enumerate one sentence representing each possible state sequence in the FSG file.

-a - Displays the annotations associated with each enumerated sentence in the format word:annotation.

-p - Displays the probability associated with each enumerated sentence.

-l N - Specifies how many times loops (repetition operators) are expanded in the enumeration. For example, with -l 3 specified, the production "very+" generates "very," "very very," and "very very very" in the enumeration, while "very*" generates "", "very," and "very very." The default value for N is 2.

-850 - Specifies that the FSG file is not converted to codepage 1252. This is useful when running from the command line and there are language-specific characters in the FSG.

file.fsg - Specifies the FSG file containing the grammar to enumerate.

The following example enumerates 10 different, random sentences from the grammar mygram.fsg and shows the probability associated with each sentence:

fsgenum -R 10 -p mygram.fsg

fsgprint

The fsgprint tool prints a state-by-state description of a compiled grammar (FSG), with annotations and arc probabilities. This tool is used primarily for diagnostic purposes. The syntax of fsgprint is:

fsgprint [-v] [-850] file.fsg

The parameters are:

-v - Prints additional information about the contents of the FSG.

-850 - Specifies that the FSG file is not converted to codepage 1252. This is useful when running from the command line and there are language-specific characters in the FSG.

file.fsg - Specifies the FSG containing the grammar to print.

The following example prints the contents of the grammar mygram.fsg:

fsgprint mygram.fsg
**fsgtest**

The fsgtest tool accepts test sentences from standard input and, for each sentence, prints to standard output an indication of whether the sentence is accepted by the grammar. This tool is used to verify the expected behavior of a grammar. The syntax of fsgtest is:

```
fsgtest [-a] [-p] [-850] file.fsg
```

The parameters are:

- **-a** - Shows the annotations associated with each sentence in the format word:annotation.

- **-p** - Shows the probability assigned by the grammar to each sentence. A probability of "*******" indicates that the grammar does not accept the sentence. If -p is not specified, fsgtest prints each input sentence that is accepted by the grammar to standard output. Any sentence that is rejected by the grammar is printed to standard output preceded by an asterisk "*".

- **-850** - Specifies that the FSG file is not converted to codepage 1252. This is useful when running from the command line and there are language-specific characters in the FSG.

**file.fsg** - Specifies the FSG containing the grammar to test.

The following example tests whether the sentence "what time is it" is accepted by the grammar mygram.fsg and shows the probability associated with that sentence:

```
fsgtest -p mygram.fsg
what time is it
```

You can use Ctrl-c or Ctrl-z to end the session.

You might also want to create a file that contains all of the sentences you want to test. You can then redirect standard input to this file. In the following example, the file test.txt is used as input to fsgtest:

```
fsgtest <test.txt mygram.fsg
```
Tcl Extensions for Version 1.2

The ViaVoice Telephony Tools extends the Tcl/Tk scripting language by providing additional functions above the standard Tcl/Tk environment. Through these extensions, a developer can use Tcl to prototype a complete phone recognition application.

Note that the ViaVoice Telephony Tcl extensions were enhanced in Version 1.1. The new extensions provide higher-level and more robust interfaces to the Tcl environment. Although applications that were written to the V1.0 Tcl interfaces are still supported, it is recommended to use the new Tcl extensions. Please refer to Tcl Extensions for V1.0 for more information.

Several new Tcl commands are provided with the ViaVoice Telephony Tools and are packaged as follows:

- The extended Tcl Shell (tclch)
- The ernestine library

For information on the Tcl language, please consult Tcl and the Tk Toolkit by John Ousterhout.

The Extended Tcl Shell (tclch)

The standard Tcl/Tk distribution package includes a shell program (tclsh) that provides a convenient execution environment for Tcl scripts. The ViaVoice Telephony Tools contains a modified version of this shell. This modified version of the shell MUST be used to run Tcl scripts that use the ViaVoice Telephony extensions. The modified version of the Tcl shell included with the ViaVoice Telephony Tools is named tclch to distinguish it from tclsh (where ch stands for "channel," since a typical application uses a tclch instance for each telephony channel).

Using the tclch Program

The standard Tcl shell initializes its own environment through global variables. The following standard initialization values are modified in the tclch process:

- auto_path = $(VTT_HOME) $(VTT_HOME)/bin $(VTT_HOME)/scripts
- tcl_pkgPath = $(VTT_HOME)/bin $(VTT_HOME)/scripts/*
- tcl_libPath = $(VTT_HOME)/bin $(VTT_HOME)/scripts/*
- tcl_library = $(VTT_HOME)/bin $(VTT_HOME)

The format of the tclch command is:

```
tclch [script-name] [-c<number>]
```
where:

**script-name** - Starts the tclch process using the specified tcl script.

-\c<number>- - Starts the tclch process for the specified channel number. For example, tclch -c0 would start the tclch process for channel 0.

Channel 99 is the default channel. Channel 99 will be used whenever a channel is not specified. To exit the tclch environment, enter "**exit**" or use "**Ctrl C**".

The following functions are built into the tclch process:

- **GetResource**
- **LogEvent**
- **SetDebug**

### GetResource

The GetResource command obtains the value of the specified resource. The format of the GetResource command is:

```
GetResource [option] [className.]resourceName
```

where:

**option** - The type of data to be returned. This can be specified as -array, -boolean, or -float. By default, a string is returned. If -array is specified, the data returned is a space-delimited list of tokens. If -boolean is specified, GetResource returns a 0 or 1. If -float is specified, a string representing a floating point number is returned.

**className** - The name of a class of resources. By default, the value of className is chp.<cloneId.CallFlow. Note that the resource values for any class can be requested by specifying the className.

**resourceName** - The name of a resource defined in the VTTDefaults configuration file.

### Examples

The following example retrieves the value for the chp.<cloneId.CallFlow.timeout resource:

```
GetResource timeout
```

This example retrieves the value for the chp.<cloneId.TelLine.bargeIn resource:

```
GetResource -bool TelLine.bargeIn
```

This example retrieves the value of the *.*. dirsToClean resource. Note that in this example, GetResource returns the array {log stat pcm}:

```
GetResource -array dirsToClean
```
The LogEvent command sends a text message to a logging process. The destination of the message depends on the configuration of the system and its resources:

- If the useConsole resource is set to true, the message is printed to the terminal.
- If the tclch process was started outside of pm or the logger process is not available, messages are stored in the <VTT_HOME\log directory in file chp.NN.log (where NN is the cloneID supplied to the tclch process at startup). The location of the log files can be specified using the logDir resource.

The message is processed only if the debug level for the CallFlow class is equal to or lower than the level specified. The format of the LogEvent command is:

```
LogEvent [option] <string>
```

or

```
LogEvent [option] <list>
```

where: logLevel equals:

<table>
<thead>
<tr>
<th>logLevel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-error</td>
<td>error</td>
</tr>
<tr>
<td>-warn</td>
<td>warning</td>
</tr>
<tr>
<td>-advise</td>
<td>advice</td>
</tr>
<tr>
<td>-info</td>
<td>info</td>
</tr>
<tr>
<td>-debug</td>
<td>debug (-dbg)</td>
</tr>
<tr>
<td>-trace</td>
<td>trace (-trc)</td>
</tr>
</tbody>
</table>

If option is not specified, debug is assumed.

```
option - The level of information to be logged. This can be -error, -warn, -advise, -info, -debug (-dbg), or -trace (-trc). If option is not specified, debug is assumed.
```

```
string - The message to log. It is an application-defined text string. Messages should be enclosed in quotes. Messages appear in the system log file depending upon the current log-level setting.
```

```
list - A list of messages to be logged. Each message is logged on a separate line.
```

Both the DialWatch utility and the logtail command filter and display LogEvent messages. For more information on DialWatch and logtail, see Utilities.

### Examples

The following example logs a debug message:

```
LogEvent "Something happened"
```

This example logs a warning message:

```
LogEvent -warn "Don't know what happened"
```

This example logs an error message:

```
LogEvent -error "Something really bad happened"
```

This example logs a trace message that includes the value of the variable var:

```
LogEvent -trace "value of var is $var"
```

Finally, this example logs two informational messages, each with the value of the specified variable:
LogEvent -info "var1 is $var1" "var2 is $var2"

**SetDebug**

The SetDebug command changes the debug level for the specified class. By using the SetDebug command, an application can generate detailed logging for selected fragments of a Tcl script. The scope of the command is contained within the process that invokes SetDebug.

The format of the SetDebug command is:

```
SetDebug [option] className [className]...
```

where:

- **option** - The debug level for the specified class. This can be -error, -warn, -info, -debug (-dbg), or -trace (-trc). If option is not specified, debug is assumed.
- **className** - The name of a class for which the debug level is to be set. Multiple classes can be specified. Note that at least one class must be specified.

**Examples**

The following example sets the debug level for the TelLine and Tsmapi classes to trace:

```
SetDebug -trace TelLine Tsmapi
```

**The ernestine.dll Library**

The functions provided in the ernestine.dll library are implemented through a widget called vttCmd. A set of Tcl commands is provided that allows an application to create, modify, and delete vttCmd objects. Each command's name becomes a Tcl command itself, and an object associated with that command is constructed. Options passed to the command are stored in the object and retained through its life (on modification, the previous value is discarded).

**NOTE:** All of the ernestine library commands can be modified from the command line at runtime. The value stated at runtime will have precedence.

The invocation of a vttCmd object starts asynchronous execution of that command. Only one command is executed at a time. Each command may have multiple completion conditions. If any of the completion conditions is reached, the content of the object is searched for the appropriate action in the following manner:

- First, the prompt `<completionStatus>` is searched. If it is found, the command loops until the maximum number of retries is exceeded.
- The prompt is then replaced with the prompt `<completionStatus>`. If no prompt is found, the action `<completionStatus>` is examined. This action may contain another vttCmd object, a segment of Tcl code, or one of the reserved return codes. If an action contains another vttCmd object, it is dispatched without returning to the originating Tcl script.
You must initialize the tclch environment prior to loading the ernestine library. To verify which parameter values are set, use the PrintCmd. To initialize the tclch environment (with channel 0) from the command line, type:

$: tclch -c0

Where -c0 is the channel ID number. Then enter:

$: load ernestine

The ernestine library is now loaded.

There are several commands provided in the ernestine library:

- AddRecoCmd
- AddTelCmd
- AddVoiceCmd
- CopyCmd
- DelCmd
- Dispatch
- ListCmds
- LogStat
- ModCmd
- PrintCmd

**NOTE:** When a single action is given on the command line, it will execute automatically. For information on possible return codes from these Tcl widgets, see Error Returns.

### AddRecoCmd

The AddRecoCmd creates a command object that interacts with the speech recognition engine. A Tcl list is used to pass arguments to each action. In addition, AddRecoCmd can also be used to connect to the NLU engine. The AddRecoCmd allows only one action with multiple options per command line.

The format of the AddRecoCmd is:

AddRecoCmd <name> [action] [options]

where:

- *name* - The name of a command. This is defined by the application.

**Example:**

$: AddRecoCmd cnCmd -connect

The following Actions_COMMANDS may be used with AddRecoCmd:
**connect [list]** - Activates a speech engine for the current audio channel. If <engine type> is specified, it identifies the type of engine to be used in the recognition session. The type is application defined. For example, if you specify the following in VTTDefaults:

```plaintext
tsm.0.*.engineType: SBU
tsm.1.*.engineType: USA
tsm.2.*.engineType: USA
```

The engine for channel 0 will be selected by applications that call the function TsmStartAppl() with the engineType set to "SBU". Note that the default engine type is "Dialer" and must be used when running appletTester.

The specific context must first be enabled using a recoCmd with -enable option before connecting to the engine.

**disable <list>** - Disables the specified context for the current speech engine.

**disconnect** - Disables the currently connected recognizer (if any) and clears all of the current operations.

**enable <list>** - Enables the specified context for the current recognizer. You must preload the grammars belonging to the context prior to enabling it. It is not an error to enable a grammar that has already been enabled.

**load <list>** - Preloads (a) grammar(s). Note that using this command can have significant effects on system performance and is important for loading large grammars during application initialization. Large grammars can take a long time to load; if memory and application type permit, all grammars should be preloaded. This can be performed during initialization as the application waits for the first phone call. Parameters that can be specified for the load command are -timeout [value] (non-zero value causes the load command to block until the grammar is loaded).

**recognize** - Activates a recognizer for the current channel ID. The specific grammars must be enabled first.

**start** - Note that specific grammars must first be enabled using the EnableVocab command before connecting to the engine. Parameters that can be specified includes: -nosildet (indicates that the engine should not return silence between spoken words), -timeout <duration> (specifies the number of milliseconds to wait until a spoken response is detected. The default is 7 seconds). This action may be used with either of the following options: annotype,maxtries, nosildet, onrejectreco, onstepbeep, sildet.

**stop** - Disables the AddRecoCmd and clears all of the current operations.

**unload [list]** - Unloads a previously loaded grammar(s).

Following are the options that may be used with the AddRecoCmd actions:

**annotype <all|last>** - Specifies the type of annotation to be returned. This option has two possible parameters (1) all (returns all recognized annotations), or (2) last (returns only the last recognized annotation). (This option is used with the "-start" action.)

**maxretries <count>** - Specifies the number of times the command should reschedule itself before giving up. This applies only if the command completes and there is a corresponding completion prompt. Here, you must also specify one of the following parameters: -promptonrejectreco <list>, -promptomsilence
<list>, or -promptonstepbeep <list>. (This option is used with the "-start" action.)

-<nl> [string] - String which allows you to connect to the NLU engine. Toggles on and off from the command line. The default state is "off."

-<nonl> - Allows you to disconnect from the NLU engine. Toggles off from the command line. The default state is "off."

-nosildet - Turns silence detection off. (This option is used with the "-start" action.)

-nowait - Causes the command to not wait until the grammar is loaded. This is important for loading large grammars during application initialization. (Asynchronous mode.)

-ondone [list] - Specifies the action to take if the command is successful.

-onfailure [list] - Specifies the action to take if the command fails.

-onrejectreco [list] - Enables a specified list or action when the recognized text is rejected by the speech engine. (This option is used with the "-start" action.)

-onstepbeep [list] - Specifies an action to execute when an utterance is detected over the beep. (This option is used with the "-start" action.)

-sildet - Turns silence detection on. (This option is used with the "-start" action.)

-timeout <ms> - The number of milliseconds before timeout occurs. (This option can be used with any Action command.)

-wait - Causes the command to block until the Action command is loaded. This is important for loading large grammars during application initialization. This option can be used with the "Load" action as well as with the "recognize" action. "Wait" causes the command to not return until finished. (Synchronous mode).

Examples

To connect to the engine and play a prompt when the connection is successful:

AddRecoCmd <name> -connect -ondone PlayPrompt

AddTelCmd

The AddTelCmd creates a command object that controls the telephony channel. A filename (<name>) must be shown when using this command. The AddTelCmd module utilizes options and actions. The format of AddTelCmd is:

AddTelCmd name [action] [options]

where:

name - The name of a command. This is defined by the application.
**options** - The type of action to perform on the telephony channel. Note that only one action can be specified for a command.

Following are the **actions** or **commands** that can be used with AddTelCmd:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>answercall</td>
<td>Waits for the telephone to ring and returns when the call is answered.</td>
</tr>
<tr>
<td>-blockcalls</td>
<td>Sets the telephone line to the busy state (which is interface dependent). This prevents anyone from calling into the system. BlockCalls should be used during system initialization or following a shutdown or fatal error.</td>
</tr>
<tr>
<td>-dropcall</td>
<td>Disconnects the current caller and returns the system to the on-hook state. It should be called at the end of an application's call flow procedure.</td>
</tr>
<tr>
<td>-getdigits</td>
<td>Retrieves the number of DTMF digits specified by the count parameter.</td>
</tr>
<tr>
<td>-makecall &lt;digits&gt;</td>
<td>Takes the phone off hook and initiates a call to the number specified by the digits parameter.</td>
</tr>
<tr>
<td>-senddigits &lt;digits&gt;</td>
<td>Generates the DTMF sequence corresponding to the specified digit string.</td>
</tr>
<tr>
<td>-transfercall &lt;digits&gt;</td>
<td>Transfers the user to the telephone number specified by the digits parameter. The call is completed once the caller is transferred and the system is placed back into the on-hook state.</td>
</tr>
<tr>
<td>-unblockcalls</td>
<td>Releases the -blockcall feature. Returns the system to the ready state, reversing the effect of a blockcalls command. On an analog system, this is equivalent to going on-hook (no longer busy).</td>
</tr>
</tbody>
</table>

Note that a digit string may contain 0 to 9, *, and ".#". A "," can be used as a pause marker (the duration of the pause is specified by the dialPauseDuration resource), "!" (Natural Microsystems board) or "&" (Dialogic board) indicates hookflash.

Following are the **options** that can be used with AddTelCmd:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmfcount &lt;count&gt;</td>
<td>Specifies how many digits to collect. (Only works with &quot;getdigits.&quot;)</td>
</tr>
<tr>
<td>-dtmftimeout &lt;ms&gt;</td>
<td>The number of milliseconds to expire when entering digits. When this number is reached, the system will</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-ondone &lt;cmd</td>
<td>proc&gt;</td>
</tr>
<tr>
<td>-ondtmf &lt;cmd</td>
<td>proc&gt;</td>
</tr>
<tr>
<td>-onfailure &lt;cmd</td>
<td>proc&gt;</td>
</tr>
<tr>
<td>-onhangup &lt;cmd</td>
<td>proc&gt;</td>
</tr>
<tr>
<td>-ontimeout &lt;cmd</td>
<td>proc&gt;</td>
</tr>
<tr>
<td>-mode &lt;sync</td>
<td>async&gt;</td>
</tr>
</tbody>
</table>

**Examples**

To add a command that waits for a call and executes the PlayGreeting function when the call is answered:

```
AddTelCmd Ans -answer -ondone PlayGreeting
```

**NOTE:** In the previous example, "PlayGreeting" must be previously defined and Ans must be defined as AnswerCall.

The following command collects four digits using a timeout of 5 seconds. On completion, the string "Gotdigits" will be returned to the Tcl script:

```
AddTelCmd Collect4Digits -getdigits 4 -dtmfcount-timeout 5000 -ondtmf GotDigits -onfailure {shutdown now}
```

**AddVoiceCmd**

The AddVoiceCmd creates a complex widget that describes the sequence play-record or recognize-play (possibly collecting DTMF digits). The format of the AddVoiceCmd is:

```
AddVoiceCmd <name> [options]
```

where:

- **name** - The name of a command. This is defined by the application.
- **options** - The type of action to perform for the voice channel. Multiple options can be specified for a command.
-**annotype**<all||last> - Specifies the type of annotation to be returned. This action has two possible parameters (1) all (returns all recognized annotations), or (2) last (returns only the last recognized annotation).

-**audiofilename [filename]** - The name of the audio file to be used.

-**bargein** - Turns barge-in on.

-**dtmfcount <count>** - Specifies how many digits to collect.

-**dtmftimeout <ms>** - Specifies how long to wait for digits if only play function is used (for example, AddVoiceCmd Play -prompt @hello -dtmfcount 4 -dtmftimeout 5000 creates a command which after playing will wait up to 5 seconds trying to collect 4 digits).

-**endprompt <list>** - Specifies the prompt to be played as soon as recording is stopped.

-**endsilencetimeout <ms>** - Defines duration of silence which will stop recording (activated only if speech was detected at least once).

-**enginetype [engineType(default:Dialer)]** - Specifies the type of engine to be used in the recognition session. The type is application defined. For example, if you specify the following in VTTDefaults:

```plaintext
tsm.0.*.engineType: SBU
tsm.1.*.engineType: USA
tsm.2.*.engineType: USA
```

The engine for channel 0 will be selected by applications that call the function TsmStartAppl() with the engineType set to "SBU". Note that the default engine type is "**Dialer**" and must be used when running the appletTester.

-**grammars <list>** - Specifies the grammars to be used in recognition.

-**initialsilencetimeout <ms>** - Defines how long to wait for speech before giving up recording.

-**interrupt** - Specifies that DTMF will break a play and record sequence.

-**maxretries <count>** - Specifies the number of times the command should reschedule itself before giving up. This applies only if the command completes and there is a corresponding completion prompt. Here, you must also specify one of the following parameters: -promptonrejectreco <list>, -promptonsilence <list>, or -promptonestepbeep <list>.

-**minspeech <ms>** - Minimum speech duration.

-**mode <sync|async>** - Asynchronous (callback mode) or synchronous.

-**nlparserr** - Starts the NLU engine.

-**nobargein** - Turns barge in off.

-**nointerrupt** - Specifies that DTMF will not break a play and record sequence.

-**norecord** - Turns recording off.

-**nosildet** - Turns silence detection off.

-**nostepbeep** - Disregard any utterances that may be detected over the beep.
-ondone <cmd|proc> - Specifies the command or process to execute if the command is successful.

-ondtmf <cmd|proc> - Specifies the command or process to execute when a DTMF response is detected.

-onfailure <cmd|proc> - Specifies the command or process to execute if the command fails.

-onhangup <cmd|proc> - Specifies the command or process to execute if the caller hangs up.

-onreco <cmd|proc> - Specifies the command or process to execute when the speech engine recognizes the utterance.

-onretriesexceeded <cmd|proc> - Specifies the command or process to execute when the number of retries has been exceeded.

-onrejctreco <cmd|proc> - Specifies the command or process to execute when the recognized text is rejected by the speech engine.

-onsilence <cmd|proc> - Specifies the command or process to execute when silence detection is turned on.

-onstepbeep <cmd|proc> - Specifies the command or process to execute when an utterance is detected over the beep.

-ontimeout <cmd|proc> - Specifies the command or process to execute if a timeout occurs.

-prompt <list> - Specifies the initial prompt to be played.

-promptonrejectreco <list> - Specifies the prompt to be played when the recognized text is rejected by the speech engine.

-promptonsilence <list> - Specifies the prompt to be played when silence is detected.

-promptonstepbeep <list> - Specifies the prompt to be played when an utterance is detected over the beep.

-record [filename] - Turns recording on and specifies the file in which to store the recorded voice data. This flag is set automatically if any grammars are defined.

-sildet - Turns silence detection on.

-stepbeep - When an utterance is detected over the beep.

-timeout <ms> - The maximum duration for a recording.

**Examples**

The following example creates a command that plays a prompt, interrupts the prompt when DTMF is detected, returns __DONE if play is completed, and returns __DTMF if a digit was received:

AddVoiceCmd PlayGreeting -prompt @greeting -interrupt

This example creates a command that plays a prompt and then processes a speech utterance:

AddVoiceCmd PlayAndRecognize -prompt @greeting -maxretries 3 -grammar "names
help" \
  -interrupt -maxretries 3 -promptonstep @speakafterbeep \ 
  -promptonsilence @didthear -promptonrej @didntunderstand -onretries GiveUp \ 
  -onreco ProcessReco

**CopyCmd**

The **CopyCmd** command creates a new command. The format of CopyCmd is:

CopyCmd destination source

where:

*destination* - The name of the new command.
*source* - The original command from which the new command is generated.

**DelCmd**

The **DelCmd** command removes a command name from the working set. The format of DelCmd is:

DelCmd name

where:

*name* - The name of the command to delete.

**Dispatch**

In asynchronous (callback) mode, the Dispatch command returns control to the event loop after a partial result has been received. The format of Dispatch is:

Dispatch name

where:

*name* - The name of a command.

Note: Specifying just a widget command name is equivalent to using Dispatch [name].

**ListCmds**

The **ListCmds** command returns a sorted list of all of the commands in the working set. The format of ListCmds is:
LogStat
The LogStat command generates an event to be sent to the statlog process. The format of LogStat is:

LogStat eventName

where:

eventName - The event to be sent to statlog.

ModCmd
The ModCmd command modifies attributes of a command. The format of ModCmd is:

ModCmd name [options]

where:

name - The name of the command to modify.

options - This depends on the type of command (AddRecoCmd, AddTelCmd, AddVoiceCmd). Any of the options supported by each type can be modified with ModCmd.

PrintCmd
The PrintCmd command provides a listing of the specified command. The format of PrintCmd is:

PrintCmd name

where:

name - The name of the command to print.

Error Returns
The possible default return values from any vttCmd command are listed below:

<table>
<thead>
<tr>
<th>Return</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>__DONE</td>
<td>Indicates that the command completed successfully.</td>
</tr>
<tr>
<td>__DTMF</td>
<td>Indicates that DTMF digits were detected. The actions for handling</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>__FAILURE</td>
<td>Indicates that an unrecoverable error was encountered. The Tcl script should exit gracefully.</td>
</tr>
<tr>
<td>__HANGUP</td>
<td>Indicates that a hangup was detected on the phone line. All pending activities are abandoned.</td>
</tr>
<tr>
<td>__PARTIAL</td>
<td>Indicates that the command is still pending but that a partial result is available. This value applies only to asynchronous execution.</td>
</tr>
<tr>
<td>__RECBEP</td>
<td>Indicates step-on-beep (that is, speech is already present when the recognizer connects to the audio stream).</td>
</tr>
<tr>
<td>__RECO</td>
<td>Initiates a successful recognition (that is, the callInfo array contains recognized text).</td>
</tr>
<tr>
<td>__RECREJ</td>
<td>Indicates that the recognizer rejected the utterance. There is no valid result available.</td>
</tr>
<tr>
<td>__RETRIES</td>
<td>Indicates that the maximum number of retries was exceeded.</td>
</tr>
<tr>
<td>__SILENCE</td>
<td>Indicates that recording terminated due to initial silence.</td>
</tr>
<tr>
<td>__TIMEOUT</td>
<td>Indicates that a time-restricted command did not before the timeout expired (for example, while waiting for DTMF digits or waiting for speech).</td>
</tr>
</tbody>
</table>

**Tcl Extensions for V1.0**

**NOTE:** The ViaVoice Telephony Tools V1.1 and V1.2 provide enhancements to the ViaVoice Telephony Tools V1.0 Tcl extensions. The V1.0 Tcl extensions are still supported; however, we recommend using the V1.2 extensions wherever possible.

The ViaVoice Telephony Tools V1.0 uses Tcl as the scripting language through which developers can easily prototype a complete phone recognition application. The Telephony Tools V1.0, V1.1 and V1.2, provide several new Tcl commands that allow manipulation of the underlying system components. These new Tcl commands are packaged in the `Ernestine.dll` library.

The `Ernestine.dll` library provides a variety of functions that can be categorized as follows:

- **Logging**
- **Speech Recognition**
- **Telephony**
User Event Handling

Error Returns

For information on the Tcl language, please consult *Tcl and the Tk Toolkit* by John Ousterhout. Note that the ViaVoice Telephony Tools Programmer's Reference follows the notational conventions of this book.

Logging Commands

The Telephony Tools provide three new Tcl commands that support logging functions:

**GetResource**

**LogEvent**

**LogStat**

**GetResource**

The GetResource command returns a string containing the specified resource. The format of the GetResource command is:

```
GetResource resourcename
```

- `resourcename` - The name of a resource defined in the VTTDefaults configuration file.

The Resource Manager looks for the resource using chp.<channel.CallFlow.<resourcename, or defaults to a resource of more global scope. It is considered a Tcl error if no resource exists.

**LogEvent**

The LogEvent command provides access to the logger process. The format of the LogEvent command is:

```
```

- `level` - This parameter is optional and may be any of the levels supported by the system, which are: -dbg, -warn, -info, -trace, -advise, -err, and -fatal.

- `message` - This parameter is an application-defined text string. A message should be enclosed in quotes. Messages appear in the system log file depending upon the current log-level setting. If multiple messages are provided, they appear one per line in the log file.

The DialWatch utility and the logtail command filter and display LogEvent messages. For more information on DialWatch and logtail, see Utilities.

**LogStat**

The LogStat command provides access to the statLog process. The format of the LogStat command is:
LogStat code ?code ...

code - An application-defined text string.

In addition to logging the specified code, information regarding the time of the event, the call channel, and the call number are also logged to the statlog file. Note that the DialWatch program is capable of displaying the status codes dynamically and in a summary table.

LogStat messages can be used to track the system state for use in collecting statistics. These codes are most effective when limited to a small manageable set.

Speech Recognition Commands

Several new Tcl commands (ConnectEngine and DisconnectEngine) have been created for this Release. Although they are new for this Release, the ConnectEngine and DisconnectEngine commands have been created to support the Version 1.0 Tcl extensions. All of the following Version 1.0 commands perform various speech recognition functions for an application. Although most speech recognizer functions can be handled by the UserEvent command, the following commands are provided for applications that need finer real-time control of speech recognition event handling:

ConnectEngine
DisconnectEngine
DisableVocab
EnableVocab
LoadGrammar
Recognize
StartRecognize
StopRecognize
UnloadGrammar

To see sample source code for Version 1.0 Tcl usage, go to the Section entitled Writing the API/Tcl Programming Tasks/" Using the V1.0 Tcl Interfaces Synchronously", or to "Using the V1.0 Tcl Interfaces Asynchronously" in this online document (IBM ViaVoice Telephony Tools Programmer's Reference).

ConnectEngine

The ConnectEngine command activates a speech engine for the current audio channel. The format of the ConnectEngine command is:

ConnectEngine <engine type>
**DisconnectEngine**

The DisconnectEngine command disables the currently connected recognizer and clears all of the current operations. The format of the Disconnect Engine command is:

```
DisconnectEngine
```

**DisableVocab**

The DisableVocab command disables a single grammar file for the current recognizer. The format of the DisableVocab command is:

```
DisableVocab grammarfile
```

**EnableVocab**

The EnableVocab command enables the specified grammar file for the current recognizer. The format of the EnableVocab command is:

```
EnableVocab grammarfile
```

You must preload the grammar prior to enabling it. It is not an error to enable a grammar that has already been enabled.

**LoadGrammar**

The LoadGrammar command is used to preload a grammar context. Note that using this command can have significant effects on system performance.

```
LoadGrammar ?-wait? contextnumber
```

- `-wait` - Causes the LoadGrammar command to block until the grammar is loaded. This is important for loading large grammars during application initialization.

- `-contextnumber` - Identifies the grammar to load.

Large grammars can take several milliseconds to load; if memory permits, all grammars should be preloaded. This can be performed during initialization as the application waits for the first phone call.

**Recognize**

The Recognize command is the primary event notifier for polled access to the recognizer. The format of the Recognize command is:

```
Recognize ? -wait?
```

- `-wait` - If this flag is specified, Recognize will wait until an event occurs before returning.
**StartRecognize**

The StartRecognize command activates a recognizer for the current audio channel. The flags are identical to those used in UserEvent. However, specific grammars must first be enabled using the EnableVocab command. The format of the StartRecognize command is:

```
StartRecognize ?-nosildet?  ?-timeout ntimeout? ?-record filename?
```

- `-nosildet` - Do not return the silence between spoken words.
- `-ntimeout` - The number of milliseconds to wait until a spoken response is detected. The default duration is 7 seconds.
- `-record` - Log the speaker's (caller's) PCM wave data, regardless of the logPcm setting.

**StopRecognize**

The StopRecognize command disables the recognizer and clears all of the current operations. The format of the StopRecognize command is:

```
StopRecognize
```

**IMPORTANT:** Each StartRecognize command must be followed by a StopRecognize command after the Recognize command returns the RECDONE event.

**UnloadGrammar**

The UnloadGrammar() command unloads a previously loaded grammar context. The format of the UnloadGrammar command is:

```
UnloadGrammar contextnumber
```

- `contextnumber` - Identifies the grammar context to unload.

**Telephony**

Several new Tcl commands are provided that perform the telephony functions for an application. These commands are:

- `AnswerCall`
- `BlockCalls`
- `Dial`
- `MakeCall`
- `Prompt`
TerminateCall
TransferCall
UnblockCalls

**AnswerCall**

The `AnswerCall` command waits for the telephone to ring and returns when the call is answered. The format of the `AnswerCall` command is:

```
AnswerCall
```

**BlockCalls**

The `BlockCalls` command sets the telephone line to the busy state (which is interface dependent). This prevents anyone from calling into the system. `BlockCalls` should be used during system initialization or following a shutdown or fatal error.

The format of the `BlockCalls` command is:

```
BlockCalls
```

**Dial**

The `Dial` command generates the DTMF sequence corresponding to the specified digit string. The format of the `Dial` command is:

```
Dial digits
```

*`digits`* - The phone number to dial.

**MakeCall**

The `MakeCall` command takes the phone off hook and initiates a call to the specified number. This can be used for developing applications that initiate calls automatically, or for load testing purposes. The format of the `MakeCall` command is:

```
MakeCall digits
```

*`digits`* - The phone number to call.

**Prompt**

The `Prompt` command plays a prompt. The format of the `Prompt` command is:
Prompt -nointerrupt? promptid ?promptid ...?

-nointerrupt - Prevents DTMF keys from interrupting the playback of a prompt.

-promptid - The id of the prompt to play (from the promptsConfig file).

The Prompt command uses the promptsConfig file to match the specified promptid parameters with the corresponding text or ulaw files and plays the prompt for the user.

**TerminateCall**

The TerminateCall command disconnects the current caller and returns the system to the on-hook state. It should be called at the end of an application's call-flow procedure. The format of the TerminateCall command is:

```
TerminateCall
```

**TransferCall**

The TransferCall command transfers the user to the telephone number specified by the digits parameter. The call is completed once the caller is transferred and the system is placed back into the on-hook state. The format of the TransferCall command is:

```
TransferCall digits
```

digits - The number to which the +call will be transferred.

**UnblockCalls**

The UnblockCalls command returns the system to the ready state, reversing the effect of a BlockCalls command. On an analog system, this is equivalent to going on-hook (no longer busy). The format of the UnblockCalls command is:

```
UnblockCalls
```

**User Event Handling**

The UserEvent command handles interaction with the user. It can be used in either of two modes:

1. If a grammar is specified, the system waits for a spoken response.
2. If no grammar is specified, the system waits for a DTMF response.

To make call flow scripts more manageable, all of the interaction from the user has been concentrated into the UserEvent command. It returns prioritized event codes. In addition, a collection of commands (Speech Recognition Commands) are provided. These commands essentially break the UserEvent function into several steps and can be used by applications that need finer real-time control of speech recognition event handling.
The format of the UserEvent command is:

```
```

- **grammar contextnumber** - If -grammar is specified, the system waits for a spoken response. If -grammar is not specified, UserEvent waits for a single DTMF key. The contextnumber parameter identifies the grammar to use for recognition.

- **annotype** - If all is specified, return all of the annotations associated with the recognized speech. If last is specified, return only the last annotation. The default is to return all annotations. Refer to the *ViaVoice SDK Speech Programmer's Guide* for more information on grammar annotations.

- **bargeIn** - Enables voice interruption of call prompts. This flag is ignored on systems that do not support barge-in.

- **dtmfcount** - Specifies how many digits to collect.

- **dtmftimeout** - The number of milliseconds to wait until a dtmf response is detected. The default duration is 7 seconds.

- **timeout** - The number of milliseconds to wait until a voice response is detected. The default duration is 7 seconds.

- **prompt** - Plays the specified prompt.

- **nointerrupt** - Prevents DTMF keys from interrupting playback.

- **record** - Log the speaker's (caller's) PCM wave data, regardless of the logPcm setting. A filename can be specified to enable data collection. The directory into which the file is stored is determined by the locPcmDir resource.

The UserEvent command returns a Tcl list that can be parsed using standard Tcl list functions (for example, lindex). The first element of the list is the response codes corresponding to the terminating condition. For example, one possible recognition result might be the Tcl list:

```
RECO {WATSON RESEARCH} 123
```

In this example, RECO is the terminating event.

### Event Codes

Following are the event codes that are returned:

<table>
<thead>
<tr>
<th>EVENT CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTMF {digitstring}</td>
<td>One or more DTMF digits was received. If the recognizer was enabled, only a single DTMF digit is returned and the recognition operation is terminated. If the -dtmf option was used, multiple keys may be returned. If -grammar is not specified, UserEvent waits for a single DTMF key.</td>
</tr>
<tr>
<td>Event Code</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DONE</td>
<td>Indicates that the command completed successfully.</td>
</tr>
<tr>
<td>FAILURE</td>
<td>Indicates that an unrecoverable error was encountered.</td>
</tr>
<tr>
<td>HANGUP</td>
<td>The caller is disconnected.</td>
</tr>
<tr>
<td>PARTIAL (recognition text)</td>
<td>This event code is used only in asynchronous mode and is returned only by the Recognize command. It returns the infirm recognition results for a partially recognized utterance.</td>
</tr>
<tr>
<td>RECDONE</td>
<td>This is returned only by Recognize, and signifies that the recognizer has reached the end of its processing for the current utterance.</td>
</tr>
<tr>
<td>RECEBP (recognition text)</td>
<td>The user spoke before the beep.</td>
</tr>
<tr>
<td>RECO (recognition text) (annotation) (high confidence flag)</td>
<td>A recognition was successfully completed. The high confidence flag is a one or zero, depending upon the resource settings. The high confidence flag is set when the score from the recognizer exceeds the value set in VTTDefaults for DMThresholdNoConfirm.</td>
</tr>
<tr>
<td>RECREJ (recognition text)</td>
<td>The recognizer rejected the utterance. This occurs if the engine rejects the utterance or the confidence level falls below the value set in DMThresholdReject in VTTDefaults.</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>A timeout occurs when the specified number of milliseconds has elapsed and no user event has occurred. If no -timeout value is specified, the default duration is 7 seconds.</td>
</tr>
</tbody>
</table>

**Error Returns**

Most of the Tcl commands return error messages that the Tcl application can handle as appropriate. The following table lists each command and the possible error returns from that command:

<table>
<thead>
<tr>
<th>Tcl Command</th>
<th>Error Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnswerCall</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td></td>
<td>Error detecting ring</td>
</tr>
<tr>
<td></td>
<td>Error connecting to reco engine</td>
</tr>
<tr>
<td>BlockCalls</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td></td>
<td>Error busying out channel</td>
</tr>
<tr>
<td>Dial</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td></td>
<td>Error dialing</td>
</tr>
<tr>
<td>DisableVocab</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td>Function</td>
<td>Errors</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DropCall</td>
<td>Wrong number of arguments Error dropping call</td>
</tr>
<tr>
<td>EnableVocab</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td>GetResource</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td>LoadGrammar</td>
<td>Wrong number of arguments Invalid argument Internal error in Tcl_SplitList</td>
</tr>
<tr>
<td>Make Call</td>
<td>Wrong number of arguments Error making call</td>
</tr>
<tr>
<td>Prompt</td>
<td>Play prompt error Internal error in Tcl_SplitList</td>
</tr>
<tr>
<td>RecordFile</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td>Recognize</td>
<td>Wrong number of arguments Expected integer but got &quot;(string)&quot; Integer value too large to represent Invalid annotate Invalid option Internal Error in Tcl_SplitList Play error getDigits error</td>
</tr>
<tr>
<td>StartRecognize</td>
<td>Expected integer but got &quot;(string)&quot; Integer value too large to represent Fatal Recognizer Failure</td>
</tr>
<tr>
<td>TransferCall</td>
<td>Wrong number of arguments Error transferring call</td>
</tr>
<tr>
<td>UnblockCalls</td>
<td>Wrong number of arguments Error enabling channel</td>
</tr>
<tr>
<td>UnloadGrammar</td>
<td>Wrong number of arguments</td>
</tr>
<tr>
<td>UserEvent</td>
<td>Expected integer but got &quot;(string)&quot; Integer value too large to represent Invalid annotate Invalid option DTMF cannot be specified with a grammar or record Internal Error in Tcl_SplitList Play error getDigits error</td>
</tr>
</tbody>
</table>
C++ Telephony Classes

The ViaVoice Telephony Tools provides two class libraries for implementing telephony applications using C++. These libraries are:

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libBase</td>
<td>Implements a reference-counted memory object.</td>
</tr>
<tr>
<td>libTsm</td>
<td>Implements interfaces to the speech recognition engine.</td>
</tr>
</tbody>
</table>

The Tools also provide the following classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Controls an audio stream for data collection and barge-in</td>
</tr>
<tr>
<td>ResourceMgr</td>
<td>Provides access to services that can be used by other ViaVoice Telephony classes.</td>
</tr>
<tr>
<td>VttTts</td>
<td>Provides text-to-speech conversion.</td>
</tr>
</tbody>
</table>
The Audio Class

The Audio class is used to directly control an audio stream for data collection and barge-in. This class provides several functions, some of which are reserved for system use.

NOTE: There are more public methods defined in Audio.h than are documented in this Reference. Undocumented methods are not supported and are subject to change.

The following functions are available for use by applications:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aud_write( )</td>
<td>Sends a buffer of audio data to the recognition engine.</td>
</tr>
<tr>
<td>AudioInit()</td>
<td>Initializes an Audio object.</td>
</tr>
<tr>
<td>getParams()</td>
<td>Return the current value of speech recognition resources.</td>
</tr>
<tr>
<td>setParams()</td>
<td>Sets the value of specified speech recognition resources.</td>
</tr>
</tbody>
</table>

The following audio states can be queried:

<table>
<thead>
<tr>
<th>Audio State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>No data is coming into the audio data stream from the audio source.</td>
</tr>
<tr>
<td>RECORDING</td>
<td>Audio data is flowing form the audio source to the recognition engine. The Audio object is recording the data.</td>
</tr>
<tr>
<td>ERROR_RPCM</td>
<td>Recording error.</td>
</tr>
<tr>
<td>AUDIO_CONNECTED</td>
<td>A connection has been established between the recognition engine and the audio source.</td>
</tr>
</tbody>
</table>

The following status can be obtained at the end of a recording session:

<table>
<thead>
<tr>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIO_INITIAL_SILENCE_TIMEOUT</td>
</tr>
<tr>
<td>AUDIO_SPEECH_OVER_THE_BEEP</td>
</tr>
</tbody>
</table>
Audio::aud_write()

```c
static int aud_write(int threadId, char *ptr, int cnt, int playFlag,
VttMemoryBuffer *mb)
```

**Parameters:**

- `threadId` - Reserved.
- `*ptr` - Pointer to the buffer containing the audio data to be sent to the recognition engine.
- `cnt` - Number of bytes of audio stream data sent to the recognition engine.
- `playFlag` - Reserved. Set this parameter to 0; if your application is using echo cancellation, set it to NULL.
- `*mb` - Reserved. Set this parameter to 0; if your application is using echo cancellation, set it to NULL.

**Returns:**

- `cnt` if successful.
- `-1` if recording is to be terminated.

**Remarks:**

aud_write sends audio data to the recognition engine and returns in `cnt`, the number of bytes sent. Call this function when a buffer of audio is available. The format of the audio is assumed to be Mulaw encoded byte data. Your application need not include logic for silence detection. Use the Audio() class status flags for silence detection.

**Note:** The aud_write function cannot be used directly with Dialogic adapters. To use aud_write with Dialogic devices, declare your own function that calls aud_write.

Audio::AudioInit()

```c
static Audio *Audio::AudioInit(int id=-1)
```

**Parameters:**

- `id` - Specifies the Tsm object to which this Audio object should be associated. For example, if the Tsm
object was created with id 0, the id parameter on the AudioInit() call should also be 0. The default id is -1, which indicates that this is a standalone configuration; that is, there is a single Tsm object and single Audio object per process.

**Returns:**

NULL - Initialization failed.

**Remarks:**

Returns a pointer to and initializes the Audio object. Note that the Audio object can only be instantiated after the Tsm object is instantiated.

### Audio::getParams()

```cpp
void getParams(int *initialSilenceDuration=0, int *minSilenceDuration=0,
    int *minSpeechDuration=0, int *voiceFormat=0,
    int *enableSpeechOverBeep=0);
```

**Parameters:**

*initialSilenceDuration* - Pointer to the returned initialSilenceDuration resource value.

*minSilenceDuration* - Pointer to the returned minSilenceDuration resource value

*minSpeechDuration* - Pointer to the returned minSpeechDuration resource value.

*voiceFormat* - Pointer to the returned voiceFormat resource value.

*enableSpeechOverBeep* - Pointer to the returned enableSpeechOverBeep resource value.

**Returns:**

None

**Remarks:**

This function retrieves the value of the specified speech recognition resources. The initialSilenceDuration, minSilenceDuration, and minSpeechDuration values are returned as a number of speech buffers. Each buffer represents 69 milliseconds of speech data. Refer to [VTTDefaults:Speech Recognition](#) for a description of each resource and possible settings.

### Audio::setParams()

```cpp
void setParams(int initialSilenceDuration=-1, int minSilenceDuration=-1,
    int minSpeechDuration=-1, int voiceFormat=-1, int enableSpeechOverBeep=-1);
```
Parameters:

initialSilenceDuration - The value to which the initialSilenceDuration resource is to be set.

minSilenceDuration - The value to which the minSilenceDuration resource is to be set.

minSpeechDuration - The value to which the minSpeechDuration resource is to be set.

voiceFormat - The value to which the voiceFormat resource is to be set.

enableSpeechOverBeep - The value to which the enableSpeechOverBeep resource is to be set.

Returns:

None

Remarks:

This function sets the value of the specified speech recognition resources. The initialSilenceDuration, minSilenceDuration, and minSpeechDuration values should be specified as a number of speech buffers, where each buffer represents 69 milliseconds of speech data. Refer to VTTDefaults:Speech Recognition for a description of each resource and valid value settings.

On call, all parameters are set to -1. Specify only those parameters that are required. Once set, parameters retain their values until another call to setParams() is made.
The libBase Class Library

The libBase class library contains many classes, most of which are used internally by the system. The VttMemoryBuffer class, explained in this section, is for use in developing telephony applications.

To use the VttMemoryBuffer class, you will need to include the appDllDefaults.h, resourceMgr.h, and VttMemoryBuffer.h files in your source file. For other files required to use the C++ APIs, refer to Files Required to Use the C++ APIs.

NOTE: There are more public methods defined in VttMemoryBuffer.h than are documented in this Reference. Undocumented methods are not supported and are subject to change.

The VttMemoryBuffer Class

The VttMemoryBuffer class implements a reference-counted memory area. It is the caller’s responsibility to control read-write access to the memory buffer.

The VttMemoryBuffer class provides the following functions:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VttMemoryBuffer()</td>
<td>Instantiates a reference-counted memory buffer.</td>
</tr>
<tr>
<td>getSize()</td>
<td>Returns the size of the file in bytes.</td>
</tr>
<tr>
<td>refCount()</td>
<td>Returns the number of references to the memory buffer.</td>
</tr>
</tbody>
</table>

VttMemoryBuffer:: VttMemoryBuffer()

VttMemoryBuffer(const char *buf="", int size=0, int dynamic=1);
OR
VttMemoryBuffer( VttFile&, char *buf=0, int size=0 );

Parameters:

*buf - A pointer to the memory buffer.

size - The size of the memory buffer. If size is 0, the buffer is treated as a string and the size of the buffer is determined as strlen+1.

dynamic - A flag which controls when the new() function is called. If dynamic is set to non-zero, the VttMemoryBuffer class allocates and maintains memory for the memory buffer.

OR

186
VttFile& - A reference to a file whose contents will be placed into a memory buffer.

*buf - A pointer to the memory buffer. If buf and size are not zero, buf is used and the caller is responsible for deleting the memory buffer.

size – The size of the buffer. If size is less than the size of the file, only size bytes are loaded. If size is 0, the size of the buffer is determined as the length of the file and the buffer is allocated (and freed) internally.

Returns:

None.

Remarks:

The VttMemoryBuffer constructor instantiates a reference-counted memory buffer object. To access the memory buffer of a VttMemoryBuffer object, cast the VttMemoryBuffer object to type char *. For example:

```c
char *buffer
VttMemoryBuffer *speechBuffer = tts.ttsText( textBuffer, TRUE );
buffer = ( char * ) ( *speechBuffer );
PlaySound( ( LPCSTR ) buffer, NULL, SND_MEMORY );
```

The following example allocates 5 bytes of memory (strlen+1) for a reference counted memory buffer:

```c
VttMemoryBuffer mb( "text" );
```

This example allocates a memory buffer to the size of a file in bytes and initializes the buffer with the content of the file:

```c
VttFile file( "d:\tmp","myfile" );
VttMemoryBuffer mb( file );
```

Finally, this example loads the specified number (size) of bytes from a file into a memory buffer:

```c
VttFile file( "d:\tmp","myfile" );
int size = file.getSize();
char *buf = new char[size];
VttMemoryBuffer mb( file, buf, size );
```

VttMemoryBuffer::getSize()

```c
int getSize( void );
```

Parameters:

None.

Returns:

The size of the memory buffer.

Remarks:

The getSize() function returns the size of the memory buffer.
VttMemoryBuffer::RefCount()
int refCount( void );

Parameters:
None.

Returns:
The reference count of the memory buffer.

Remarks:
The refCount() function returns the reference count to the memory buffer.
The libTsm Class Library

The ViaVoice Telephony Tools provides interfaces to the speech recognition engine and the text-to-speech engine through the libTsm library. This library provides a condensed, asynchronous set of interfaces via the Tsm class.

To use the Tsm class, you will need to include the appDllDefaults.h, resourceMgr.h, and tsm.h files in your source file. For other files required to use the C++ APIs, refer to Files Required to Use the C++ APIs.

NOTE: There are more public methods defined in tsm.h than are documented in this Reference. Undocumented methods are not supported and are subject to change.

The objects in the libTsm library are built on top of the SMAPI interfaces of the ViaVoice engine and to the ECI interfaces to the ViaVoice Outloud text-to-speech engine. For more information about these interfaces, refer to the ViaVoice Developer Tools Speech Programmer’s Guide.

Note that libTsm has no knowledge of the telephony adapter.

The Tsm class is implemented as a singleton object. There is only one Tsm object per process, and the process can be connected to only one engine at a time.

To work with a Tsm object, a ResourceMgr object must be previously instantiated and initialized. This guarantees proper processing of the resource database and correct logging of information.

The Tsm class provides the following functions:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TsmDelete()</td>
<td>Deletes a word from the specified context.</td>
</tr>
<tr>
<td>TsmDisableVocab()</td>
<td>Disables a vocabulary.</td>
</tr>
<tr>
<td>TsmEnableVocab()</td>
<td>Enables a vocabulary.</td>
</tr>
<tr>
<td>TsmEndAppl()</td>
<td>Releases the speech recognition engine.</td>
</tr>
<tr>
<td>TsmGetAnno()</td>
<td>Returns the annotations associated with a string.</td>
</tr>
<tr>
<td>TsmGetLastError()</td>
<td>Returns the last error that occurred.</td>
</tr>
<tr>
<td>TsmGetNbest()</td>
<td>Returns the n best alternatives for a given speech utterance.</td>
</tr>
<tr>
<td>TsmGetNbestAnno()</td>
<td>Returns the annotations associated with the n best alternatives for a given speech utterance.</td>
</tr>
<tr>
<td>TsmGetText()</td>
<td>Returns the recognized text.</td>
</tr>
<tr>
<td>TsmInit()</td>
<td>Creates and initializes a Tsm object.</td>
</tr>
<tr>
<td>TsmLoadContext()</td>
<td>Loads the specified context(s).</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TsmRetrieveSynth()</td>
<td>Retrieves the synthesized text from the text-to-speech engine.</td>
</tr>
<tr>
<td>TsmSaveContext()</td>
<td>Saves the specified context(s).</td>
</tr>
<tr>
<td>TsmSendText()</td>
<td>Sends text to the text-to-speech engine for synthesis.</td>
</tr>
<tr>
<td>TsmStartAppl()</td>
<td>Allocates an engine (speech recognition or text-to-speech) and initiates a session.</td>
</tr>
<tr>
<td>TsmStartReco()</td>
<td>Causes the speech engine to wait for speech input.</td>
</tr>
<tr>
<td>TsmStopReco()</td>
<td>Waits for the audio feed to be turned off.</td>
</tr>
<tr>
<td>TsmTerm()</td>
<td>Closes the engine session and releases associated resources.</td>
</tr>
<tr>
<td>TsmTrain()</td>
<td>Adds a new word to the specified context.</td>
</tr>
<tr>
<td>TsmUnloadContext()</td>
<td>Unloads the specified context.</td>
</tr>
<tr>
<td>TsmWait()</td>
<td>Waits until a previously requested function completes and returns status to the called function.</td>
</tr>
<tr>
<td>TsmWordData()</td>
<td>Returns the data for a word.</td>
</tr>
</tbody>
</table>

All operations with the speech recognition engine are asynchronous, but Tsm methods, which serve these operations, return immediately with either VttSuccess or VttError. Note that a return of VttSuccess does not mean that the entire operation was successful. To ensure that the application processes all possible returns from the engine correctly, it should wait for a response from the engine by using the TsmWait() method. TsmWait() returns a set of flags. As soon as the error bit is set, the application should call TsmGetLastError() and process this error message.

**Tsm::TsmDelete()**

```cpp
int TsmDelete(int WordNum, char *ContextName, VttMemoryBuffer *mbuf);
```

**Parameters:**

- `WordNum` - The number of the word to delete.
- `*ContextName` - A pointer to the context name from which the word should be deleted.
- `*mbuf` - The file name or memory buffer which contains the context.

**Returns:**

- VttSuccess
- VttError

**Remarks:**

The TsmDelete() function deletes the word number from the context specified by ContextName.
ContextName is used to interpret the mbuf parameter. If ContextName is PERSONAL_WDS, mbuf is a buffer pointer that contains the context. If ContextName is PERSONAL_FILE, mbuf is interpreted as a file name.

**Tsm::TsmDisableVocab()**

```c
int TsmDisableVocab(char *ContextId, int Text=0);
```

**Parameters:**

*ContextId* - The application-defined index to a context that was previously loaded.

*Text* - Disables the language model. The default is 0, which indicates that no language model is used.

**Returns:**

VttSuccess
VttError

**Remarks:**

The `TsmDisableVocab()` function disables the context specified by the ContextId parameter. This function returns immediately, although it can then be waited on using TsmWait(). This function can be called multiple times and does nothing if the context is already disabled.

**Tsm::TsmEnableVocab()**

```c
int TsmEnableVocab(char *ContextId, int Text=0);
```

**Parameters:**

*ContextId* - The application-defined index to a context that was previously loaded.

*Text* - Enables a language model. The default is 0, which indicates that no language model is used. Note that specifying this option is ONLY supported for U.S. English, and will return an error if used with other languages.

**Returns:**

VttSuccess
VttError

**Remarks:**

The `TsmEnableVocab()` function enables the context specified by the ContextId parameter. This function returns immediately, although it can be waited on using TsmWait(). This function assumes that the context is already loaded.

If the context being enabled is a grammar, the ContextID parameter can not be NULL, and Text should be set to 0.

If the context being enabled is a language model, the ContextID parameter is ignored, and Text should be set to 1.
**Tsm::TsmEndAppl()**

```c
int TsmEndAppl(int noAudio=0);
```

**Parameters:**

`noAudio` - Identifies whether the Audio object was created with the corresponding TsmStartAppl() call. A value of 0 indicates that the Audio object was created during the TsmStartAppl() call; therefore, TsmEndAppl() deletes the Audio object. If the value of noAudio is not 0, TsmEndAppl() does not delete the Audio object.

**Returns:**

VttSuccess
VttError

**Remarks:**

The `TsmEndAppl()` function releases the engine that was started with a corresponding TsmStartAppl() method. Note that extra calls to TsmEndAppl() with no associated TsmStartAppl() call are harmless.

---

**Tsm::TsmGetAnno()**

```c
int TsmGetAnno(int AnnReq, char *anno, int buflen);
```

**Parameters:**

`AnnReq` - A flag which indicates whether to get the last annotation (TSM_LAST_ANN) or all annotations (TSM_ALL_ANN).

`*anno` - A pointer to the memory buffer where the annotation(s) will be stored.

`buflen` - The length of the memory buffer.

**Returns:**

VttSuccess
VttError

**Remarks:**

If TsmWait() returns a status of TSM_EOU or TSM_PHRASE, `TsmGetAnno()` returns the annotations since the last TsmStartReco() call.

It makes the most sense to call TsmGetText() and, if some words are available, call this function to get the annotations for the string. This is a synchronizing command which waits until the response of the message is received. All other asynchronous events received during this time will be consumed.
**Tsm::TsmGetLastError()**

```cpp
int TsmGetLastError(char **errorMessage, int *command);
```

**Parameters:**

*errorMessage* - The address of a pointer to the string which will contain the last error that occurred.

*Command* - A pointer to the number which represents the command in which the error occurred.

**Returns:**

VttSuccess  
VttError

**Remarks:**

The **TsmGetLastError()** function retrieves the last error message from the Tsm object. The Tsm object saves the last error and all subsequent accesses to the Tsm object return error messages until **TsmGetLastError()** is called.

**TsmGetLastError()** returns VttSuccess and VttError, but these are used differently than they are with other Tsm methods. They do not reflect the status of the current Tsm method (**TsmGetLastError()**), but, more precisely, they reflect the status of the previous Tsm method. If **TsmGetLastError()** returns VttSuccess, then no errors occurred on the previous Tsm method. If **TsmGetLastError()** returns VttError, this indicates that an error occurred and **TsmGetLastError()** can provide additional information about the error.

Errors can be issued by the speech recognition engine and by the Tsm library itself. In the first case, **TsmGetLastError()** returns VttError and additional information about the error. In the second case (for example, an invalid parameter was passed to a Tsm method), the Tsm method returns VttError, but **TsmGetLastError()** returns VttSuccess. In both cases, you should check the corresponding log file for more specific details about the error that occurred.

For each function, the following errors are reported:

<table>
<thead>
<tr>
<th>Function</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>TsmDelete()</td>
<td>FAILURE IN TSM_DELETE</td>
</tr>
<tr>
<td>TsmDisableVocab()</td>
<td>FAILURE IN TSM_DISABLE_VOCAB</td>
</tr>
<tr>
<td>TsmEnableVocab()</td>
<td>FAILURE IN TSM_ENABLE_VOCAB</td>
</tr>
<tr>
<td>TsmEndAppl()</td>
<td>FAILURE IN TSM_END_APPL</td>
</tr>
<tr>
<td>TsmGetAnno()</td>
<td>FAILURE IN TSM_GET_ANNO</td>
</tr>
<tr>
<td>TsmGetNbest</td>
<td>FAILURE IN TSM_GET_NBEST</td>
</tr>
<tr>
<td>TsmGetNbestAnno</td>
<td>FAILURE IN TSM_GET_NBEST_ANNO</td>
</tr>
<tr>
<td>TsmGetText()</td>
<td>FAILURE IN TSM_GET_TEXT</td>
</tr>
</tbody>
</table>
TsmLoadContext()  FAILURE IN TSM_LOAD_CONTEXT
TsmSaveContext()  FAILURE IN TSM_SAVE_CONTEXT
TsmStartAppl()  FAILURE IN TSM_START_APPL
TsmStartReco()  FAILURE IN TSM_START_RECO
TsmStopReco()  FAILURE IN TSM_STOP_RECO
TsmTrain()  FAILURE IN TSM_TRAIN
TsmUnloadContext()  FAILURE IN TSM_UNLOAD_CONTEXT
TsmWordData  FAILURE IN TSM_WORD_DATA

See also the TsmWait() method.

Tsm::TsmGetNbest()

int TsmGetNbest(int ind, char *text, int buflen, float *score=NULL);

Parameters:

ind - The index into the buffer where the alternatives are stored.

*text - A pointer to a buffer where the alternatives are stored.

buflen - The length of the buffer where the annotation(s) will be stored.

*score - A pointer to a buffer where the scores are stored.

Returns:

VttSuccess
VttError

Remarks:

The TsmGetNbest() function returns the first n alternatives for a given speech utterance. The maximum number of alternatives that are returned by the speech engine is 10. Associated with each hypothesis is a relative score (or likelihood), as determined by the speech engine. Note that the speech engine does not always return 10 alternatives; it may return less than 10.

You must loop to get the number of desired alternatives. Start with an index of 0. A result (0 to n) is valid if it is not an empty string.

If the TsmWait() function returns a status of TSM_EOU or TSM_PHRASE, TsmGetNbest() retrieves the text into the text buffer. This is a synchronizing command which waits until the response is received. All other asynchronous events received during this time will be consumed.
Tsm::TsmGetNbestAnno()

int TsmGetNbestAnno(int ind, int AnnReq, char *Anno, int buflen);

Parameters:

ind - The index into the list of annotations.

AnnReq - A flag which indicates whether to get the last annotation (TSM_LAST_ANN) or all annotations (TSM_ALL_ANN).

*Anno - A pointer to the memory buffer where the annotation(s) will be stored.

buflen - The length of the buffer where the annotation(s) will be stored.

Returns:

VttSuccess
VttError

Remarks:

If TsmWait() returns a status of TSM_EOU or TSM_PHRASE, TsmGetNbestAnno() returns the annotations associated with each alternative to the recognized text since the last TsmStartReco() call.

It makes the most sense to call TsmGetNbest() and, if some words are available, call this function to get the annotations for the string. This is a synchronizing command which waits until the response of the message is received. All other asynchronous events received during this time will be consumed.

Tsm::TsmGetText()

int TsmGetText(char *text, int buflen, float *score=NULL);

Parameters:

*text - A pointer to a buffer that will contain the recognized text.

buflen - The length of the buffer into which the text is returned.

*score - A pointer to a buffer where the scores are stored.

Returns:

VttSuccess
VttError

Remarks:

If the TsmWait() function returns a status of TSM_EOU or TSM_PHRASE, TsmGetText() retrieves the text in the string ‘text’. This is a synchronizing command which waits until the response is received. All other asynchronous events received during this time will be consumed.
Tsm::TsmInit()

static Tsm* TsmInit(int id=-1, char *name=NULL);

Parameters:

id - Identifies the Tsm object that corresponds to each channel. The value of id can range from 0 to 99. The default value is -1, which indicates the default object in ResourceMgr.

*name - Identifies the name of the remote connection. This parameter is meaningful ONLY in the client/server environment. The default value is NULL, which uses a name of "RPCMailbox" for the remote connection.

Returns:

A pointer to the Tsm object
NULL in case of error

Remarks:

The TsmInit() method may be called any time a Tsm object is required. If a Tsm object does not exist, TsmInit() creates one, initializes it, and returns a pointer to it. Subsequent calls to TsmInit() simply return a pointer to the Tsm object.

TsmInit() is a static method. It is the only way to create a Tsm object, because the Tsm constructor is protected.

The Tsm object exists until the application exits and cannot be deleted.

TsmInit() returns NULL in the following cases:

- When trying to connect to a Tsm object without having initialiazed a ResourceMgr object first.
- If the Tsm object is already initialized.
- If the application tries to create more than one Tsm object (in a local configuration).
- When the engine session associated with the Tsm object was closed by TsmTerm(). Further access to the Tsm object is not possible. In this case, TsmGetLastError() cannot be used to determine the error returned from TsmInit() because the Tsm object doesn't exist any more. You can review the log file to determine the cause of the error.

The following example creates and intializes a Tsm object for channel 0 for a local session:

TsmInit( 0 );

This example creates and initializes a Tsm object for channel 1 for a remote session:

TsmInit( 1, "ernestine" );

Note that you must first initialize the ResourceMgr object before calling TsmInit(). For example:

ResourcMgr resm;
resm.init("main");
resm.initialize("chp", 0);
Tsm *myTsm = Tsm::TsmInit( 0 );

Note that there is no confirmation returned after TsmInit(). Simply check for the existence of the Tsm object.
Tsm::TsmLoadContext()

int TsmLoadContext(int Wait, char *ContextName, VttMemoryBuffer *context, char* ContextId);

Parameters:

Wait - Specifies whether the function runs synchronously (Wait=1) or asynchronously (Wait=0). If Wait is set to 0, it can be monitored using TsmWait().

*ContextName - A pointer to the type of context to load.

*context - A pointer to the name of the file list (if FILELIST is specified as the contextName).

ContextId - An application-specified index to the context. Subsequent requests to TsmEnableVocab(), TsmDisableVocab(), and TsmUnloadContext() require the ContextId. For convenience, you can use the file name.

Returns:

VttSuccess
VttError

Remarks:

The TsmLoadContext() function loads the specified context into memory. Contexts are identified by the contextName parameter. The following table defines each context type and the action taken by TsmLoadContext():

<table>
<thead>
<tr>
<th>CONTEXT NAME</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASEFORM_FILE</td>
<td>Loads a baseform file. The words in a vocabulary must have associated pronunciations (baseforms) so that the engine can recognize them. To provide these pronunciations, use TsmLoadContext() with BASEFORM_FILE.</td>
</tr>
<tr>
<td>FILELIST</td>
<td>Enables multiple contexts to be specified and loaded (excluding those that require memory buffers). ContextName then indicates the name of the file that contains a list of the files to be loaded. TsmLoadContext() is called for all of the files in the FILELIST, even if some of the files return an error. If any files return an error at load time, TsmLoadContext() returns VttError.</td>
</tr>
<tr>
<td>GRAMMAR_FILE</td>
<td>Loads a compiled grammar file (.fsg). This is the most frequently used context type. If any of the pronunciations for the words in the grammar file are unknown, the context is not loaded, and the log file will contain a warning message and a list of the words for which the pronunciations are missing. The pronunciations must be</td>
</tr>
<tr>
<td><strong>MNR</strong></td>
<td>Specifies a pointer to the memory buffer where the MNR_FILE is loaded.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>MNR_FILE</strong></td>
<td>Loads a cepstral mean norm vector file (.mnr). This file is used to tune the system and filter background noise. The .mnr file is an array of values that represents the cepstral mean norm vector. The cepstral mean substraction technique is used to provide an efficient way of removing the influence of the transmission channel (such as the telephone line or microphone).</td>
</tr>
<tr>
<td><strong>PERSONAL_FILE</strong></td>
<td>Loads a personal vocabulary file. This file contains user-specific pronunciations and is created when a user enrolls or trains words to his/her voice.</td>
</tr>
<tr>
<td><strong>PERSONAL_WDS</strong></td>
<td>Specifies a pointer to the memory buffer where the PERSONAL_FILE is loaded.</td>
</tr>
<tr>
<td><strong>VOCAB_FILE</strong></td>
<td>Loads a plain text file (.voc) that contains a list of the words and phrases to be recognized. Every word to be recognized should be placed on a new line. Multiple words on the same line are recognized as a phrase. Note that the length of a single line should not exceed 20 bytes.</td>
</tr>
</tbody>
</table>

An example of a FILELIST is shown below:

```
VOCAB_FILE c:\dialer\userid.voc
GRAMMAR_FILE c:\dialer\dialer2000.fsg
GRAMMAR_FILE c:\dialer\dialer2001.fsg
GRAMMAR_FILE c:\dialer\dialer2002.fsg
```

All of the files specified in the FILELIST file are loaded as a single context. A subsequent TsmUnloadContext() call will, correspondingly, unload all of the files specified in the list.

The most commonly used context types are GRAMMAR_FILE and FILELIST. For more information on creating the contexts themselves, refer to Developing Phone Grammars.

Note that a common error when trying to load contexts is that the file name is specified incorrectly. In the current version of Windows NT and the ViaVoice engine, grammar files are required to be specified as follows:

```
GRAMMAR_FILE c:vtt\app\yesno\yesno.YESNO.fsg
```

Errors from loading contexts are returned using the TsmWait() function, unless ‘wait’ (the timeout parameter) is specified. If wait is specified and an error occurred, -1 is returned. Note that if the TsmLoadContext() call returns an error, the engine should be released or a TsmWait() can be issued to clear the error.

In a client-server scenario, the files that are loaded must reside on the server, and the server's VTTDefaults file specifies where they are found (via the appPath resource). On the client, the grammarPrefix resource should be set in the client's VTTDefaults file to the path on the server where the files are found.
Tsm::TsmRetrieveSynth()

```cpp
int TsmRetrieveSynth(VttMemoryBuffer **ombuf);
```

**Parameters:**

**ombuf** - The address of a pointer to the memory buffer in which to receive the synthesized text.

**Returns:**

VttSuccess
VttError

**Remarks:**

The TsmRetrieveText() function requests the buffer of synthesized text. This method blocks until the buffer is ready or the timeout is exceeded.

Tsm::TsmSaveContext()

```cpp
int TsmSaveContext(char *ContextName, VttMemoryBuffer* imbuf, VttMemoryBuffer **ombuf);
```

**Parameters:**

*ContextName* - The context to save.

*imbuf* - A pointer to a memory buffer which contains the file name in which the context will be saved. See Remarks.

**ombuf** - The address of a pointer to the memory buffer where the context will be saved. See Remarks.

**Returns:**

VttSuccess
VttError

**Remarks:**

The TsmSaveContext() function saves the context specified by ContextName. The ContextName parameter can only be MNR, MNR_FILE, PERSONAL_FILE, or PERSONAL_WDS. The other context types are not valid with TsmSaveContext().

For PERSONAL_FILE and MNR_FILE, only the imbuf parameter is used (ombuf is not meaningful for these context types). The imbuf parameter is interpreted as the file name where the data will be stored.

For PERSONAL_WDS and MNR, only the ombuf parameter is used (imbuf is not meaningful for these context types). The ombuf parameter specifies the address of a memory buffer in which to store the context.

Deallocation of the ombuf buffer is the responsibility of the caller.

The following example saves an MNR_FILE context to the file mnr.tmp:

```cpp
VttMemoryBuffer mnrFile("mnr.tmp");
DialReco->TsmSaveContext( "MNR_FILE", &mnrFile, NULL );
```
To store a PERSONAL_WDS context to a memory buffer, note that the imbuf parameter is NULL:

VttMemoryBuffer* pwdsBuf;
DialReco->TsmSaveContext( "PERSONAL_WDS", NULL, &pwdsBuf );

**Tsm::TsmSendText()**

```c
int TsmSendText(char *text, int splice=0, int useEloqOnly=1);
```

**Parameters:**

*text - A pointer to the text to synthesize.

splice - This parameter is RESERVED and must be set to 0.

useEloqOnly - This parameter is RESERVED and must be set to 1.

**Returns:**

VttSuccess
VttError

**Remarks:**

The `TsmSendText()` function sends text to the text-to-speech engine for synthesis.

**Tsm::TsmStartAppl()**

```c
int TsmStartAppl(char *appName, int noAudio=0, char *lang=NULL);
```

**Parameters:**

*appName - A pointer to the name of the application to start or a pointer to the type of engine to find. Note that specifying the type of engine to find is ONLY valid in remote configurations. If NULL is specified, it indicates that this is a local configuration and appName will be "**".

noAudio - Identifies whether the Audio object was previously created. A value of 0 means that the Audio object was not created; therefore TsmStartAppl() creates it. The corresponding TsmEndAppl() deletes the Audio object if the same value for noAudio is specified. Note that noAudio should be set to 1 when connecting to the text-to-speech engine and to 0 when connecting to the speech recognition engine. See Remarks below.

*lang - A pointer to a string which specifies the language of the engine. Note that the lang parameter is ONLY valid in remote configurations. If lang is specified, the tsmRouter selects an engine of the specified language and type. If lang is NULL, then the TsmRouter ignores the language of an engine and selects an engine of the specified type only.

**Returns:**

VttSuccess
VttError

**Remarks:**

The `TsmStartAppl()` function sends text to the text-to-speech engine for synthesis.
The TsmStartAppl() function allocates an engine and indicates that a new session is beginning. The engine can be a speech recognition engine or a text-to-speech engine. This function determines if the application properly stopped the last session using TsmEndAppl. If it did not, a warning message is logged and the previous session is cleaned up.

If the specified speech recognition engine is remote, this function finds an engine of the type specified by *appname. For example, if you specify the following in VTTDefaults:

tspm.0.*.engineType: SBU
tspm.1.*.engineType: USA
tspm.2.*.engineType: USA

The engine for channel 0 will be selected by applications that call the function TsmStartAppl() with the engineType set to "SBU".

Some additional notes about using the noAudio flag: Every process has only one (common) array of pointers to Audio objects. Every Audio object corresponds to a channel. The AudioInit() method may be called many times, and it simply returns a pointer if the Audio object exists. If the Audio object doesn't exist, it will first create it. The noAudio flag, then, tells Tsm whether to create the Audio object inside Tsm or not. If the Audio object is already created by the application, "noAudio=0" connects the existing object to the corresponding pointer inside Tsm. If "noAudio=1" is specified, Tsm won't know about the existing Audio object and some of its methods will return errors.

Upon successful completion of a TsmStartAppl() call, a connection to the engine is established.

Note: Do not use TsmWait() to monitor successful completion of TsmStartAppl().

**Tsm::TsmStartReco()**

```c
int TsmStartReco( int MaxTime, int silenceDetection=1, int save_audio=0, int echoCancel=0, int mode=0);
```

**Parameters:**

*MaxTime* - The amount of time to wait for an utterance.

*silenceDetection* - Specifies whether or not the silence model is enabled for this utterance. The default is enabled (=1).

*saveAudio* - Used in conjunction with training. It tells the engine whether or not to save the audio so that it can train a new word using that audio. Note that this parameter doesn't provide the actual audio for feedback in training; that is done separately, as is data collection. The default is "not to save" (=0).

*echoCancel* - Specify 1 to use host barge-in. Ensure that the appropriate barge-in resource is set.

*mode* - Reserved.

**Returns:**

VttSuccess
VttError

**Remarks:**

The **TsmStartReco()** function causes the engine to wait for speech input. The amount of audio data that
the engine can capture is controlled by the MaxTime parameter.

In addition, resources can be set that control silence timeout, the amount of speech to begin looking for, the end of utterance, and the amount of silence to declare the end of utterance may be set using resources. For more information, refer to ViaVoice Telephony Resources.

The TsmStartReco() function may be called if the audio stream is already on (it simply does nothing). To stop the audio stream, call TsmStopReco().

The TsmStartReco() command is synchronous and should not be monitored with TsmWait().

Tsm::TsmStopReco()

int TsmStopReco();

Parameters:
None.

Returns:
VttSuccess
VttError

Remarks:
The TsmStopReco() function waits for the audio feed to be turned off. Calling TsmStopReco() is a REQUIRED synchronizing step. This ensures that previous asynchronous TsmStartReco() calls are completed before TsmStopReco() returns.

When TsmStopReco() is called, the Audio object is first turned off. Next, the message TSM_STOP_RECO is sent to the tsmProc object. It then waits for the Audio object to complete its session and return to IDLE state.

Calling TsmStopReco() when the audio stream is already off does nothing.

Tsm::TsmTerm()

int TsmTerm();

Parameters:
None

Returns:
VttSuccess
VttError

Remarks:
The TsmTerm() function closes the engine session (speech recognition or text-to-speech) associated with the Tsm object and releases all of its associated resources. It does not, however, delete the Tsm
object from memory. Further attempts to access the Tsm object return errors (including trying to reinitialize it through TsmInit(), which returns NULL).

As a result, TsmGetLastError() cannot be used to determine the error returned from TsmInit() because the Tsm object doesn't exist any more.

Tsm::TsmTrain()

```c
int TsmTrain(char *wordData, int *wordNum, char *ContextName, int *wordStartTime, int *wordEndTime, VttMemoryBuffer *mbuf);
```

Parameters:

* `wordData` - A pointer to the word to add.
* `wordNum` - A pointer to the number of the word.
* `ContextName` - A pointer to the context in which to add the new word.
* `wordStartTime` - Returned by TsmTrain to help the application trim the feedback prompt.
* `wordEndTime` - Returned by TsmTrain to help the application trim the feedback prompt.
* `mbuf` - A pointer to the memory buffer in which the context is stored.

Returns:

VttSuccess
VttError

Messages from TsmWait():

- TSM_TRAIN_OK
- TSM_TRAIN_TOO_LONG
- TSM_TRAIN_TOO_SHORT
- TSM_SILENCE_TIMEOUT
- TSM_SPEECH_OVER_BEEP

Remarks:

The TsmTrain() function adds a new word (wordData) with the specified number (wordNum). The new word is added into the context specified by the memory buffer. This enables personalization of vocabularies for specific users. The context can be saved to a file using the TsmSaveContext() method.

If ContextName, mbuf, or wordNum is NULL, TsmTrain() returns an error immediately.

Note that the result of the TsmTrain() function should be monitored by TsmWait(), which can return additional flags (see Returns above).

Tsm::TsmUnloadContext()

```c
int TsmUnloadContext(char *contextId, int force=0);
```
Parameters:

*contextId* - The application-defined index to a context that was previously loaded.

*force* - Specifies whether or not to forcefully remove the context from memory. If force is specified (force=1), then the context is removed from memory immediately. The default action (force=0) does not forcefully remove previously loaded contexts from memory. However, even if force=0, a context may be removed from memory if it has not been active for some time.

Returns:

VttSuccess
VttError

Remarks:

The `TsmUnloadContext()` function removes the specified context from memory. This function returns immediately, although it can be waited on using `TsmWait()`. Unloading a context that has already been unloaded results in no error.

If FILELIST was specified on the `TsmLoadContext()` function, the corresponding `TsmUnloadContext()` will unload all of the files in the list. No subset can be unloaded unless it was loaded individually.

Tsm::TsmWait()

```cpp
int TsmWait(int timeout, int *status, char *arg=NULL, int argLen=0, char **errorMsg=NULL, int *errorCommand=NULL);
```

Parameters:

*timeout* - The maximum amount of time to wait for a result to be returned. This is specified in milliseconds.

*status* - A pointer to the status flags returned by `TsmWait()`. Ensure that the status is set to 0 prior to calling `TsmWait()`.

*arg* - A pointer to a string into which the word results are placed. The default NULL indicates that the word results are not to be given to your application.

*argLen* - The length of arg.

**errorMsg** - The address of a pointer to a string into which error messages are placed. The default NULL indicates that error messages are not to be given.

*errorCommand* - A pointer to the number of the last command which caused the error. The default NULL indicates that error messages are not to be given.

Returns:

VttSuccess
VttError

Remarks:

The `TsmWait()` function waits for a previously requested function. If wait is set to 0, `TsmWait()` polls and returns immediately. If wait is set to -1, `TsmWait()` waits for the next message from the Tsm class. This is
useful if your application is solely waiting for recognition results. If wait is set to any other value, TsmWait() returns either when the function is complete or when the time has expired, whichever occurs first.

The status parameter is returned by TsmWait() and can be a combination of the following flags (using the OR operation):

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSM_NO_STATUS</td>
<td>No error on current message.</td>
</tr>
<tr>
<td>TSM_SILENCE_TIMEOUT</td>
<td>Speech at the beginning of the utterance is silence to the length of the initial silence timeout parameter</td>
</tr>
<tr>
<td>TSM_SPEECH_OVER_BEEP</td>
<td>Initial frames of speech indicate that the person has started speaking before the beep was played.</td>
</tr>
<tr>
<td>TSM_REJECTED</td>
<td>The utterance was rejected by the engine or by the threshold logic of the Tsm object.</td>
</tr>
<tr>
<td>TSM_EOU</td>
<td>Completed processing the current utterance. Note that [TsmStopReco()] must be called after a TSM_EOU occurs before any new recognition events can be handled.</td>
</tr>
<tr>
<td>TSM_PHRASE</td>
<td>A recognized phrase is available.</td>
</tr>
<tr>
<td>TSM_NOCONFIRM</td>
<td>The score of the current utterance was sufficient to not require confirmation on the last command.</td>
</tr>
<tr>
<td>TSM_RECORDDONE</td>
<td>The audio stream is stopped and perhaps recognition results are being waited on.</td>
</tr>
<tr>
<td>TSM_TRAIN_OK</td>
<td>Training of the current phrase is successful.</td>
</tr>
<tr>
<td>TSM_TRAIN_TOO_LONG</td>
<td>The utterance presented for training was too long for good recognition.</td>
</tr>
<tr>
<td>TSM_TRAIN_TOO_SHORT</td>
<td>The utterance was too short for good recognition.</td>
</tr>
</tbody>
</table>
TSM_FAILURE

The last command resulted in a failure.

The TsmWait() function retrieves the last error noted by the Tsm object as well as the last command in which the error occurred, if the pointers are not null. In this respect, TsmWait() duplicates the TsmGetLastError() function. If TsmWait returns VttError, information about the error is returned in the flags. A subsequent call to TsmGetLastError() will return VttSuccess (no errors), because TsmWait() has actually called TsmGetLastError() itself.

Tsm::TsmWordData()

int TsmWordData(int WordNum, char *wordData, char *ContextName,
VttMemoryBuffer* mbuf, int put);

Parameters:

WordNum - The number of the word for which to retrieve or store data.

*wordData - The data to be associated with the word.

*ContextName - The name of the context.

*mbuf - A pointer to the memory buffer which contains the context.

put - A flag which specifies the function to be performed. See Remarks below.

Returns:

VttSuccess
VttError

Remarks:

The TsmWordData() function either retrieves word data for the word number, or it associates word data to the word number. The function of TsmWordData() is specified by the put parameter. If put is 1, the string specified by wordData is associated with the specified word number. If put is 0, the word data associated with the specified word number is retrieved.

The ContextName parameter is used to interpret the mbuf parameter. If ContextName is PERSONAL_WDS, mbuf is a pointer to the buffer that contains the context. If ContextName is PERSONAL_FILE, mbuf is interpreted as a filename.
The ResourceMgr Class

The ResourceMgr class provides a C++ application access to services that can be used by other ViaVoice Telephony classes. These services are:

- Logging of debugging information and alarms
- Dynamic change of debug level
- Access to configuration resources defined in VTTDefaults

To use the ResourceMgr class, you will need to include the `appDIIDefaults.h` and `resourceMgr.h` files in your source file. For other files required to use the C++ APIs, refer to Files Required to Use the C++ APIs.

**NOTE:** There are more public methods defined in resourceMgr.h than are documented in this Reference. Undocumented methods are not supported and are subject to change.

The ResourceMgr class provides the following member functions:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceMgr()</td>
<td>Instantiates a ResourceMgr object.</td>
</tr>
<tr>
<td>~ResourceMgr()</td>
<td>Removes a ResourceMgr object</td>
</tr>
<tr>
<td>doLog()</td>
<td>Generates a log file entry of a specified type</td>
</tr>
<tr>
<td>finish()</td>
<td>Removes common objects.</td>
</tr>
<tr>
<td>hexDump()</td>
<td>Generates formatted output of binary data that includes an ASCII transcript column.</td>
</tr>
<tr>
<td>init()</td>
<td>Initializes a ResourceMgr object for a class.</td>
</tr>
<tr>
<td>initialize()</td>
<td>Initializes a ResourceMgr object for a process.</td>
</tr>
<tr>
<td>resRequestBool()</td>
<td>Gives access to VTTDefaults configuration resources and provides the resource value as boolean.</td>
</tr>
<tr>
<td>resRequestFloat()</td>
<td>Gives access to VTTDefaults configuration resources and provides the resource value in floating point format.</td>
</tr>
<tr>
<td>resRequestInt()</td>
<td>Gives access to VTTDefaults configuration resources and provides the resource value in integer format.</td>
</tr>
<tr>
<td>resRequestStr()</td>
<td>Gives access to VTTDefaults configuration resources and provides the resource value in character string format.</td>
</tr>
<tr>
<td>resRequestStrArray()</td>
<td>Returns a VTTDefaults resource's value as an array of tokens.</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>resRequestUStr()</td>
<td>Returns a VTTDefaults resource's values as an array of tokens</td>
</tr>
</tbody>
</table>

**ResourceMgr::ResourceMgr()**

```cpp
ResourceMgr();
```

**Parameters:**

None

**Returns:**

None

**Remarks:**

The `ResourceMgr()` constructor instantiates a `ResourceMgr` object. A `ResourceMgr` object can be used for a class or a process. If created for a class, the `init()` method must be used to initialize the object. If created for a process, the `initialize()` method must be used.

**ResourceMgr::~ResourceMgr()**

```cpp
~ResourceMgr();
```

**Parameters:**

None

**Returns:**

None

**Remarks:**

The `ResourceMgr()` destructor destroys a `ResourceMgr` object

**ResourceMgr::doLog()**

```cpp
void doLog(ostringstream& str, enum LoggingLevel l);
```

**Parameters:**

str - a pointer to the message data stream to be written to the log.

l - The debug level. Possible levels and their meaning are:
• trace - Sets trace on for all classes. Because of the volume of messages written, trace degrades system performance.

• dbg - Generates a large amount of information but does not impede system performance.

• info - Generates information for events that monitor the call process. Approximately 20 to 30 messages are generated for each call.

• advise - Provides advisory information. For example suggestions for configuration changes.

• warn - Warning. Errors from which the system automatically recovers. This is the default.

• err - Logs information about an error from which the system cannot recover. However, the process may be restarted.

• fatal - Logs information about an error from which the system cannot recover. The process is exited immediately.

• exit0 - The process is exited immediately with an exit code of 0.

• exit1 - The process is exited immediately with an exit code of 1.

Returns:
None

Remarks:
The DoLog() function formats the message that is logged. The message includes the time, debug level, process name, and clone id. The DoLog() function is called by the LOG macro to achieve logging. The LOG macro, defined within the ResourceMgr header file, determines if there is information to be logged. Use the LOG macro to perform logging. The LOG macro format is:
LOG (logginglevel, ostream)

ResourceMgr::finish()
void finish();

Parameters:
None

Returns:
None

Remarks:
The finish() function destroys entities that are common to all ResourceMgr objects. Perform this call before exiting. Further access to the ResourceMgr object as well as reinitialization is no longer possible.
ResourceMgr::hexDump()
void hexDump(unsigned char *data, int count);

Parameters:
*data - Pointer to data stream that is to be formatted.
count - The length of data stream.

Returns:
None

Remarks:
The hexDump() function generates formatted output of binary data that includes the ASCII transcript column.

ResourceMgr::init()
void init(const char *classname, const char *who=0, int id=-1);

Parameters:
'classname' - A pointer to the name of the class that is using ResourceMgr.
*who - A pointer to the nickname of the class that is using ResourceMgr.
'id' - The id used in the TsmInit() call.

Returns:
None

Remarks:
The init() function initializes the ResourceMgr object for a class. Each class object must be initialized using this function. Furthermore, you must use the ResourceMgr object named resm to perform the initialization:
ResourcMgr resm;
resm.init("main");
The resm object is declared by the Tsm class and its scope is the specified class.

ResourceMgr::initialize()
void initialize(const char *procname, int procid=0, int local=FALSE, logginglevel l=warn);

Parameters:
**procname** - The name of the process to be initialized.

**procid** - The process' clone identification. The default is 0, the initial instance.

**local** - Indicates whether to log in the system logger file or in process' log file. FALSE indicates that messages should be written to the system logger file, current.log. TRUE indicates that messages should be written to the file cloneid.log.

**logginglevel** - Value to which the debug level is to be set for process *procname* and cloneid *procid*. The default is warning messages which describe errors from which the system automatically recovers. Use this parameter to dynamically change a process' debug level that was set through VTTDefaults. Refer to the explanation of the LoggingLevel parameter of the DoLog() function for the valid values.

**Returns:**

None

**Remarks:**

The initialize() function initializes the ResourceMgr object for a process. Each process object must be initialized using this function. Furthermore, you must use the ResourceMgr object named resm to perform the initialization:

```cpp
ResourceManager resm;
resm.initialize("chp", 0);
```

The resm object is declared by the tsm class and its scope is the specified process (procname).

Through the initialize() function, the ResourceMgr class implements logging of messages from all ViaVoice Telephony processes that are running. The messages are either logged to a process-designated file or to the system file, current.log, if the system's logger process is active.

When a ViaVoice Telephony system object uses logging for the first time after initialization, the system logger process is called. If communication is successfully established, the process is registered with the system logger process. The debug level is dynamically set to the specified level.

**ResourceManager::resRequest xxx**

The resRequest() functions give access to VTTDefaults resources. The system maintains all resources as strings. Determine the resources's string type and use the appropriate resRequest function to return a boolean indicator or convert the resource string to a Float or Integer value.

```cpp
ResourceManager::resRequestBool()
ResourceManager::resRequestFloat()
ResourceManager::resRequestInt()
ResourceManager::resRequestStr()
```

```cpp
int resRequestBool(const char *name)
double resRequestFloat(const char *name)
int resRequestInt(const char *name)
const char *resRequestStr(const char *name);
```

**Parameters:**

* name - Pointer to the string containing the scoped name of the resource in the format:
Refer to the VTTDefaults Resources Overview for an explanation of this format.

Returns:

0 - Resource was not found

Remarks:

Upon return, the resource value in the requested type, is pointed to by "name. Except for Str, you can then change the returned resource value. Bool is defined as on|off|1|0|true|false|yes|no and is case insensitive. Bool returns 1 or 0 if the resource is found. If a resource is not found, Bool also returns 0, an error message is printed, and the process is ended using the C Runtime Library assert() function.

ResourceMgr::resRequestStrArray()
ResourceMgr::resRequestUStr()

const char **resRequestStrArray(char *name, char *separator = " ");

or

custom char **resRequestStrArray(char *name, const char *separator = " ");
custom unsigned char *resRequestUStr(const char *tag);

Parameters:

*name or *tag - Pointer to the string containing the scoped name of the resource in the format:
<process name>.<clone id>.<classname>.<resource name>
Refer to the VTTDefaults Resources Overview for an explanation of this format.

*separator - array element separator. A space or Tab is the default.

Returns:

0 - Resource was not found.

Remarks:

The resRequestStrArray() and resRequestUStrArray() functions tokenize the value of the resource and return an array of pointers to the tokens. The returned array is terminated by a NULL. Array elements are separated by the default separator whitespace or Tabspace. If a resource is not found, an error message is printed and the process is ended using the assert() function.
The *vttTts* Class

The *vttTts* class converts a string of text to synthesized speech in PCM or WAV format.

To use the VttTts class, you will need to include the `appDllDefaults.h`, `resourceMgr.h`, and `VttTts.h` files in your source file. For other files required to use the C++ APIs, refer to Files Required to Use the C++ APIs.

**NOTE:** There are more public methods defined in vttTts.h than are documented in this Reference. Undocumented methods are not supported and are subject to change.

The class has the following member functions:

<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>vttTts()</em></td>
<td>Instantiates a vttTts object.</td>
</tr>
<tr>
<td>~<em>vttTts()</em></td>
<td>Destroys a vttTts object.</td>
</tr>
<tr>
<td><em>ttsInit()</em></td>
<td>Initializes a vttTts object.</td>
</tr>
<tr>
<td><em>ttsSetPitchBaseline()</em></td>
<td>Sets the pitch of the voice with which the text is converted.</td>
</tr>
<tr>
<td><em>ttsSetPitchFluctuation()</em></td>
<td>Sets the pitch fluctuation of the voice with which the text is converted.</td>
</tr>
<tr>
<td><em>ttsSetSpeed()</em></td>
<td>Sets the speed of the voice with which the text is converted.</td>
</tr>
<tr>
<td><em>ttsSetVolume()</em></td>
<td>Sets the volume of the voice with which the text is converted.</td>
</tr>
<tr>
<td><em>ttsStop()</em></td>
<td>Performs housekeeping after the conversion is complete.</td>
</tr>
<tr>
<td><em>ttsText()</em></td>
<td>Returns a memory buffer containing synthesized speech for a specified text string.</td>
</tr>
</tbody>
</table>
vttTts::vttTts()

vttTts();
Parameters:
None
Returns:
None
Remarks:
Instantiates a vttTts object and sets the resource class to "Tts".

vttTts::~vttTts()

~vttTts();
Parameters:
None
Returns:
None
Remarks:
Destroys a vttTts object.

vttTts::ttsInit()

int ttsInit();
Parameters:
None
Returns:
VttSuccess
VttError
Remarks:
Initializes a vttTts object, loads the corresponding DLLs, and sets such parameters as language, quality, volume, speed, and roughness. This function must be called after a vttTts object is instantiated.
vttTts::ttsSetPitchBaseline()

int ttsSetPitchBaseline(int pitch);

Parameters:

pitch - The pitch of the voice at which the text is converted. Pitch can range from 0 to 100 (where 0 is approximately 40 Hz and 100 is approximately 422 Hz). The default value is 60.

Returns:

VttSuccess
VttError

Remarks:

Use of this method is optional. The default setting for pitch is used if it is not specified.

vttTts::ttsSetPitchFluctuation()

int ttsSetPitchFluctuation(int fluctuation);

Parameters:

fluctuation - The pitch fluctuation rate of the voice at which to convert the text. Pitch fluctuation can range between 0 to 100, where 0 is a very monotonous voice and 100 shows extreme fluctuation in the voice. The default value is 40.

Returns:

VttSuccess
VttError

Remarks:

Use of this method is optional. The default setting for pitch fluctuation is used if it is not specified.

vttTts::ttsSetSpeed()

int ttsSetSpeed(int speed);

Parameters:

speed - Sets the speed of the voice with which the text is converted. The speed can range from 0 to 100, where 0 is very slow and 100 is very fast. The default is 60.

Returns:

VttSuccess
VttError

Remarks:

Use of this method is optional. The default speed is used if it is not specified.
vttTts::ttsSetVolume()

ttsSetVolume(int volume);

Parameters:

volume - Sets the volume of the voice at which the text is converted. The valid range is 0 to 100 log-scaled. The default is 99.

Returns:

VttSuccess
VttError

Remarks:

Use of this method is optional. The default volume is used if it is not specified.

vttTts::ttsStop()

int ttsStop();

Parameters:

None

Returns:

VttSuccess
VttError

Remarks:

Performs housekeeping. This function must be called before exiting.

vttTts::ttsText()

VttMemoryBuffer *ttsText( VttMemoryBuffer *text, int alaw, int wave);

Parameters:

*text - Pointer to the text string to be converted.

alaw - Indicates the output format. Set to TRUE fo ULAW or to FALSE for MULAW. The default is ULAW.

wave - Indicates the output format. Set to TRUE for WAV output or to FALSE for PCM output. The default is PCM.

Returns:

VttMemoryBuffer
Null in case of error

Remarks:
The ttsText method() allocates a memory buffer into which the converted text is returned. Refer to the VttMemoryBuffer class. It is the responsibility of the caller to delete the memory buffer returned by ttsText() when it is no longer needed.
Writing the Application Interface

Now that you have designed, tested, and debugged your grammars, you need to write the application interface for your phone recognition application. This can be done in a TCL script using the TCL extensions, or it can be developed as a C++ program using the libTsm and libBase libraries. For information on the TCL extensions, refer to TCL Extensions. For information on the C++ interfaces, refer to C++ Telephony Interfaces.

All phone recognition applications need to perform certain functions. Writing the application interface to perform these functions is relatively straightforward, whether you choose to implement them using the TCL extensions or the C++ interfaces.

This section describes the basic functions that a phone recognition application should support. There are some differences in the tasks that a TCL application needs to implement versus those that a C++ application needs to implement, so information is provided specific to each environment. This section also suggests some optional tasks that a developer may want to consider which are independent of the implementation environment.

The Preliminary Tasks

Before beginning to write your telephony application, plan the characteristics of your system environment and your application. Define the ViaVoice Telephony resource configuration through the procman.cfg, promptsConfig, and VTTDefaults files. Plan your application's vocabulary and prompts. See Phone Recognition Grammars Overview. You should then develop your phone grammars. See Developing Phone Grammars.

Basic Phone Recognition Tasks

A phone recognition application is, essentially, a speech recognition application that uses a different input device. Many of the same programming tasks that apply to speech recognition applications also apply to phone recognition applications. For more information the basic tasks that all speech recognition applications should implement, refer to Chapter 11. Writing the Application Programming Interface in the ViaVoice Developer Tools Speech Programmer's Guide.

The basic process for interacting with the speech engine is:

1. Establish a connection to the speech engine.
2. Tell the engine what words the user can say. To do this, you define and enable your application vocabularies.
3. Tell the engine to process speech. An application does this by requesting that the engine begin recognizing speech.
4. As speech is recognized by the engine and decoded into text, process this information and decide what to do with it.
5. Tell the engine when to stop recognizing speech.
6. Disconnect from the engine.

Programming Tasks for Tcl Applications

At a minimum, all phone recognition applications in Tcl should implement steps 1 through 6 listed above. This section describes the phone recognition tasks that are relevant to the Tcl environment.

Connecting to the Speech Recognition Engine

A Tcl application must explicitly establish a connection to the speech recognition engine. To do this, use the AddRecoCmd with the -connect option specified.

Enabling and Disabling Vocabularies

You can have multiple grammars (associated with vocabularies) active for your phone recognition application at any time. However, to improve performance (both recognition and accuracy), your application should narrow the possibilities by enabling and disabling grammars as they are needed.

Use the AddRecoCmd with the -load option specified to load a grammar and the AddRecoCmd with the -enable option specified to enable a grammar. To disable a grammar, use the AddRecoCmd with the -disable option specified. Or, you can use the AddVoiceCmd with the -grammar option specified. (If you are using the 1.0 Tcl interfaces, use the LoadGrammar command to load a grammar, the EnableVocab command to enable a grammar, and the DisableVocab command to disable a grammar).

To identify and use a dynamic command vocabulary, use the same Tcl commands. You must ensure that baseforms containing pronunciations exist for all of the words for the vocabulary. Refer to Generating Baseforms From Vocabularies.

The speech recognition engine stops decoding after it recognizes a word or phrase. This is an opportune time for your application to change the active grammars (associated with vocabularies) for recognizing the next command. (While the engine is halted, it continues to capture and process audio. Therefore, no words are lost).

Directing the Engine to Process Speech

Using the TCL interfaces, an application directs the engine to start processing speech with the AddRecoCmd -recognize option. The AddRecoCmd -stop option is used to tell the speech engine to stop processing speech input. If you use the AddVoiceCmd, you do not need to explicitly direct the engine to start and stop processing speech. (The V1.0 equivalent commands are StartRecognize and StopRecognize, respectively).
Processing Recognized Speech

A TCL application interprets the return from the AddRecoCmd -recognize command to determine what to do with the recognized text. Or, if using the AddVoiceCmd, it processes the return from that command. (Using the V1.0 interfaces, the UserEvent command is interpreted to process speech results).

Disconnecting from the Speech Engine

A Tcl application must explicitly disconnect from the speech recognition engine. To do this, use the AddRecoCmd with the -disconnect option specified. If you are using the AddVoiceCmd to process speech, you do not have to explicitly disconnect from the engine.

Examples

Tcl interfaces can be used synchronously or asynchronously. The following four examples are provided to demonstrate the programming tasks described above:

- Using the Tcl interfaces synchronously
- Using the Tcl interfaces asynchronously
- Using the V1.0 Tcl interfaces synchronously
- Using the V1.0 Tcl interfaces asynchronously

Using the Tcl Interfaces Synchronously

The following example illustrates the synchronous use of Tcl interfaces. The AddVoiceCmd is used in this example.

```
load ernestine
set grammar [GetResource grammarPrefix]
append grammar "yesno/grammars/yesno.YESNO.fsg"
# ====================== Define functions =============================
# Wait for and answer a call
AddTelCmd ProcessCall -answercall
# Play prompts
AddVoiceCmd Play
# Connect with engine, load, enable grammar, start reco, reco
# stop reco, unload and disable grammar and disconnect with engine
AddVoiceCmd Recognize -grammar $grammar -timeout 30000
# Terminate call
AddTelCmd DropCall -drop
# ========================= Call Flow ==============================
while {1} {
    puts "Wait for a call .."
}
Using the Tcl Interfaces Asynchronously

The following example illustrates the asynchronous use of the Tcl interface. The AddRecoCmd is used in this example.

```tcl
set grammar [GetResource grammarPrefix]
append grammar "yesno/grammars/yesno.YESNO.fsg"
# ============= Define functions =========================
# for play prompts
AddVoiceCmd Play -prompt [list "say yes or no"]
# for terminate call
AddTelCmd DropCall -drop
# For wait for and answer a call
AddTelCmd Answer -answercall
# For connect with speech engine
AddRecoCmd ConnectEngine -connect
# For disconnect with reco engine
AddRecoCmd DisconnectEngine -disconnect
# For load grammar
AddRecoCmd LoadVocab -load $::grammar
# For enable the grammar
AddRecoCmd EnableVocab -enable $::grammar
# For start reco process
AddRecoCmd StartReco -start -sildet -timeout 20000
# For process reco
AddRecoCmd Recognize -recognize
# For stop reco process
AddRecoCmd StopReco -stop

# =============== Call Flow ================================
while {1} {
    puts "Wait for a call ..."
    Answer
    break
}
```

```
ProcessCall
puts "say yes or no"
Play -prompt [list "say yes or no"]
set rc [Recognize]

switch $rc {
    __RECO {
        puts "RECO : $::callinfo(text)"
        Play -prompt [list "I recognized $::callinfo(text)"]
    }
    __HANGUP {}
    __DONE {Prompt [list "result is" "done"]}
    __TIMEOUT {Prompt [list "result is" "timeout"]}
    __STEPBEEP {Prompt [list "result is" "step on beep"]}
    __SILENCE {Prompt [list "result is" "silence"]}
    default {}
}
DropCall
```
puts "ConnectEngine: [ConnectEngine]"
puts "LoadVocab: [LoadVocab]"
puts "EnableVocab: [EnableVocab]"
set done 0
Play -prompt [list "say yes or no"]
puts "StartReco: [StartReco]"
while {$done == 0} {
    set rc [Recognize]
    switch $rc {
        __PARTIAL { puts "WORD: $::callinfo(text)" }
        __RECO {
            puts "RECO : $::callinfo(text)"
            Play -prompt [list "I recognized $::callinfo(text)"
            set done 1
        }
        __HANGUP {set done 1}
        __DONE {Play -prompt [list "result is" "done"]}
        __TIMEOUT {Play -prompt [list "result is" "timeout"]}
        __STEPBEEP {Play -prompt [list "result is" "step on beep"]}
        __SILENCE {Play -prompt [list "result is" "silence"]}
        default {}
    }
}
DisconnectEngine
DropCall

Using the V1.0 Tcl Interfaces Synchronously

The following example illustrates the synchronous use of V1.0 Tcl interfaces. The UserEvent command is
used in this example.

load ernestine
set grammar [GetResource grammarPrefix]
append grammar "yesno/grammars/yesno.YESNO.fsg"
# == Call Flow ==
while {1} {
    puts "Wait for a call ..."
    AnswerCall
    puts "say yes or no"
    Prompt [list "say yes or no"]
    set rc [UserEvent -grammar $grammar -timeout 20000]
    switch $rc {
        __RECO {
            puts "RECO : $::callinfo(text)"
            Prompt [list "I recognized $::callinfo(text)"
            set done 1
        }
        __HANGUP {}
        __DONE {Prompt [list "result is" "done"]}
        __TIMEOUT {Prompt [list "result is" "timeout"]}
        __STEPBEEP {Prompt [list "result is" "step on beep"]}
Using the V1.0 Tcl Interfaces Asynchronously

The following example illustrates the asynchronous use of the V1.0 Tcl interfaces. The StartRecognize command is used in this example.

```
__SILENCE {Prompt [list "result is" "silence"]
    default ()
}
TerminateCall
}

using the V1.0 Tcl Interfaces Asynchronously

The following example illustrates the asynchronous use of the V1.0 Tcl interfaces. The StartRecognize command is used in this example.

```
load ernestine
set grammar [GetResource grammarPrefix]
append grammar "yesno/grammars/yesno.YESNO.fsg"

================================ Call Flow =================================

while {1} {
    puts "Wait for a call ..."
    AnswerCall
    puts "ConnectEngine: [ConnectEngine Dialer]"
    puts "LoadGrammar: [LoadGrammar $::grammar]"
    puts "EnableVocab: [EnableVocab $::grammar]"
    set done 0
    Prompt [list "say yes or no"]
    puts "StartRecognize: [StartRecognize]"

    while {$done == 0} {
        set rc [Recognize]

        if { [string match PARTIAL* $rc] } {
            puts "$rc"
        } elseif { [string match RECO* $rc] } {
            puts "$rc"
            set done 1
        } else {
            switch $rc {
                HANGUP {set done 1}
                DONE {Prompt [list "result is" "done"]}
                TIMEOUT {Prompt [list "result is" "timeout"]}
                STEPBEEP {Prompt [list "result is" "step on beep"]}
                SILENCE {Prompt [list "result is" "silence"]}
            default {}
Programming Tasks for C++ Applications

At a minimum, all phone recognition applications implemented in C++ should implement steps 1 through 6 listed above. This section describes the phone recognition tasks that are relevant to the C++ environment.

Files Required to Use the C++ APIs

The following header files must be included in your source program to use the ViaVoice Telephony C++ APIs:

- **appDllDefs.h** - This header file must ALWAYS be included and should be the first one listed.
- **Audio.h** - This header file is required to use the Audio class.
- **resourceMgr.h** - This header file must ALWAYS be included. Refer to the resourceMgr class.
- **tsm.h** - This header file is required to use the Tsm class.
- **VttMemoryBuffer.h** - This header file is required to use the VttMemoryBuffer class.
- **vttTts.h** - This header file is required to use the vttTts class.

Additionally, if you want to use the grammar compiler APIs, you must include the vtbnfc.h file. The following libraries are also required:

- **libBase.lib** - This library must ALWAYS be linked to your application.
- **libTsm.lib** - If you are using the Tsm class, this library must be linked to your application.
- **libTts.lib** - If you are using the vttTts class, this library must be linked to your application.
Connecting to the Speech Recognition Engine

A C++ phone recognition application must explicitly establish a connection to the speech recognition engine. To do this, use the TsmStartAppl() function.

Enabling and Disabling Vocabularies

You can have multiple grammars (associated with vocabularies) active for your phone recognition application at any time. However, to improve performance (both recognition and accuracy), your application should narrow the possibilities by enabling and disabling grammars as they are needed. Use the TsmEnableVocab() function to enable a grammar and the TsmDisableVocab() function to disable a grammar.

To identify and use a dynamic command vocabulary, use the TsmLoadContext() function, specifying the context BASEFORM_FILE, to load the baseform (pronunciation) file into which missing pronunciations were placed. Recall that baseform files reside in <VTT_HOME>/app/<Ll_LL>/app_name/baseforms/ where <Ll_LL> specifies the language directory. For example:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German.

And app_name is an application name. Then use the TsmLoadContext() function specifying the context VOCAB_FILE to load the dynamic command vocabulary. You must ensure that baseforms containing pronunciations exist for all of the words for the vocabulary. Refer to Generating Baseforms From Vocabularies.

To compile a grammar from within your application, refer to Using the Grammar Compiler API.

The speech recognition engine stops decoding after it recognizes a word or phrase. This is an opportune time for your application to change the active vocabularies for recognizing the next command. (While the engine is halted, it continues to capture and process audio. Therefore, no words are lost). Use TsmLoad() and TsmUnload() to load and unload vocabularies and grammars.

Directing the Engine to Process Speech

Using the C++ interfaces, an application directs the engine to start processing speech with the TsmStartReco() function. The TsmStopReco() function is used to tell the engine to stop processing speech input.

Processing Recognized Speech

A C++ application is notified by the TsmWait() function that the speech recognition engine has detected and decoded speech. The application uses the TsmGetText(), TsmGetAnno(), and TsmWordData()
functions to process the data from the speech engine.

Disconnecting from the Speech Engine

A C++ application must explicitly disconnect from the speech engine. To do this, use the `TsmStopAppl()` function.
Sample Programs

There are three sample programs provided with the ViaVoice Telephony Tools. Two Tcl scripts (appletTester.tcl and ndsample.tcl) and one C++ program (recotest) are included. There are also several sample grammars provided for use with the samples. These sample grammars can be enabled within your application as is or modified and extended to meet the requirements for your application's vocabulary.

appletTester.tcl

This script (appletTester.tcl) illustrates the use of the ViaVoice Telephony Tcl extensions for speech recognition. It demonstrates the following concepts:

- Providing multiple application grammars and enabling single user-selected grammars at the appropriate times.
- Parsing recognized text and taking appropriate action.
- Constructing and playing audio prompts back to the caller.
- Handling DTMF as well as voice input.

The appletTester program is found in the <VTT_HOME>\scripts\samples directory. Refer to the appletTester.tcl Script for the actual contents of the script.

The appletTester script uses the sample grammars that are provided with the Tools (in the <VTT_HOME>\app\<language> directory). You can use these grammars as the basis for your application grammars if so desired. Language-specific versions of the grammars are provided.

You can use the applet tester (applettester.tcl) to verify that the tclch program and all input is working properly. To initialize the applet tester, enter:

$: source scripts/samples/.../applettester.tcl

It is not necessary to enter the channel number again since it was entered previously. However, if you did not enter the channel ID previously, you must enter it on one of the following lines, otherwise the tcl script will not be able to find the server.

There are two ways to specify command line information needed by the tclch environment:

1. The Channel number can be specified separately (as in the previous example).
2. The Channel number and application name may be specified on the same command line (as in the following example).

$: tclch scripts/samples/applettester.tcl -c0

If you have used previous versions of the ViaVoice Telephony Run Time, you will notice that the AppletTester is now being installed with the Tools software. Therefore, the appletTester is no longer automatically started. To automatically start the appletTester, you must create an entry in the procman.cfg file. For your convenience, we have provided a sample appletTester entry that can be inserted into the
procman.cfg file. This sample entry is located in the `<VTT_HOME>\applettester_procman.cfg` (or root) directory. Merely, insert this file (applettester_procman.cfg) at the end of the regular procman.cfg file to start the appletTester.

To determine what words you can say, refer to Sample Grammar Vocabularies. You can also enable these grammars within your telephony application to prototype and test the call flow.

**ndsampke.tcl**

The *ndsampke.tcl* script illustrates the use of the ViaVoice Telephony Tcl extensions for speech recognition and telephony.

- Enabling a grammar.
- Parsing recognized text and taking appropriate action.
- Constructing and playing audio prompts back to the caller.
- Transferring the call to another phone number.
- Handling speech recognition events such as misrecognition.
  Handling telephony events such as DTMF and user hangup.

The *ndsampke* is found in the `<VTT_HOME>\scripts\samples` directory. It uses a simple text database of names and phone numbers that is stored in the *ndsampkesubscriber.db* file. From this file, the grammars, baseforms, and engine pronunciation pools can be generated (using MakeApp or the Administration Tools).

For your convenience, we have provided a sample ndsampke entry for use with the procman.cfg file. This file (ndsampke_procman.cfg) is located in the `<VTT_HOME>\ndsampke_procman.cfg` (or root) directory. Merely, insert this file at the end of the regular procman.cfg file to start the ndsampke.

Note: The ndsampke script cannot be used in a client-server configuration.
appletTester Script Source

This is the content of the appletTester.tcl script described in Sample Programs.

```tcl
package require ernestine

set grammars [ list {banking/grammars/banking.BAN.fsg} \ 
{banking/grammars/banking.BAN.fsg} \ 
{commands/grammars/commands.COM.fsg} \ 
{dates/grammars/dates.dates.fsg} \ 
{spelling/grammars/sp_v_mail.VMA.fsg} \ 
{times/grammars/times.times.fsg} \ 
{vanc/grammars/vanc.1.fsg} \ 
{v_mail/grammars/v_mail.VMA.fsg} \ 
{yesno/grammars/yesno.YESNO.fsg} \ 
{dig/grammars/dig.UNC.fsg} ]

set grammarprefix [GetResource grammarPrefix]

proc PlayResult {} {
    set rc [Play -prompt "$::callinfo(text) @recogd" ]
    if { [string compare __DONE $rc] == 0 } {
        return $rc
    }
    return "PromptForGrammar -prompt @tkintro"
}

proc SelectGrammar {} {
    switch -glob -- $::callinfo(digits) {
        # -
        {*}{ return { PromptForGrammar -prompt @tkintro } }
        [0-9]{
            set grm $::grammarprefix
            append grm [lindex $::grammars $::callinfo(digits)]
            return "Recognize -grammar $grm"
        }
    }
    return __RESULT
}

# define Tcl commands for the script:
# ProcessCall - waits for a call and invokes PromptForGrammar
# when it arrives
# DropCall - just disconnect caller (or respond to hangup)
# Play - play prompt given at invocation
# PromptForGrammar - ask for grammar to be selected by dtmf
# Recognize - ask for speech entry, optionally play user
# error message (silence, step on beep,
# ejection)
AddTelCmd ProcessCall -answercall -onDone {PromptForGrammar prompt @tkintro} -onhangup DropCall
AddTelCmd DropCall -dropcall
```

229
AddTelCmd UnblockCalls -unblockcalls

AddVoiceCmd Play -dtmftimeout 1 -onhangup DropCall \
-ondone {PromptForGrammar -prompt @tkintro}

AddVoiceCmd PromptForGrammar -prompt @tkintro -dtmftimeout 5000 \
-ontimeout { PromptForGrammar -prompt @tkintro } -ondtmf { 
SelectGrammar } -onhangup DropCall

AddVoiceCmd Recognize -prompt @sayitem \
-maxretries 100 \
-onstepbeep {PromptForGrammar -prompt "@afterbeep @tkintro"} \
-onsilence {PromptForGrammar -prompt "@didnthear @tkintro"} \
-ontimeout {PromptForGrammar -prompt "@didnthear @tkintro"} \
-onrejectrec {PromptForGrammar -prompt "@didntud @tkintro"} \
-ondtmf {PromptForGrammar -prompt @tkintro} \
-onfailure { DropCall } -ondone { DropCall } \
-onreco { PlayResult } -onretries { DropCall } -onhangup \
dropCall
if [catch {
while (1) {
  UnblockCalls
  set rc [ProcessCall]
  while { ![string match {[^__]*} $rc] == 0 } {
    set rc [eval $rc]
  }
} error ] {
  puts $error
  puts $::errorInfo
}

ndsmaple Script Source

This is the content of the ndsample.tcl script as described in Sample Programs:

```tcl
# The IBM Name Dialer Sample TCL Script
#
package require ernestine

# voice commands
AddVoiceCmd PlayMessage -norecord
AddVoiceCmd ExecuteGrammar -bargein -record
AddVoiceCmd _Delay

# telephone commands
AddTelCmd LockTelLine -blockcalls
AddTelCmd UnlockTelLine -unblockcalls
AddTelCmd DropCall -dropcall
```
AddTelCmd WaitForCall -answercall
AddTelCmd DispatchCall

# Load the IBM NT telephone extension libraries. This provides
# access to all telephony and resource Tcl functions.
# load libtts.dll
# wait <milliSeconds>

proc Delay { milliSeconds } {
    return [__Delay -timeout $milliSeconds]
}

LockTelLine

# Set the VERSION global variable for display during the startup.
# This variable is assigned by the CVS versioning software.
# set VERSION {$Id: ndsample.tcl,v 1.3 1999/03/16 11:01:40 vsdl
# Exp $}
#
# The xfer Tcl procedure is one of the terminal points in the
# dialer. The "Just a minute..." prompt is played before
# initiating a transfer.
# The user has time to press the pound or star key, or to hang up.
# Press of the star key takes the user back to the top of the
# callflow by calling cancelrestart. Any other spurious input
# takes the user to onematch which asks for a confirmation before
# returning to xfer.
#
proc xfer {number trynumber retry} {
    set rc [Delay 500]
    LogEvent -dbg $rc
    if { $rc == "__HANGUP" } { return HANGUP }
    if { $rc == "__TIMEOUT" || ($rc == "__DTMF" && $::callinfo(digits) == ">#") } {
        DispatchCall -transfercall $number
        return XFEROK
    } else {
        LogStat CANCEL
        return 0
    }
}

# The onematch Tcl procedure is called when there is only one
# telephone number that matches the recognized name, but the
# confidence level of the recognition was not high enough to
# allow automatic transfer. The function asks the user to
# confirm the result by saying "Yes," or pressing the pound
# key. Saying "No" or pressing the star key will take the user
# back to the top of the call flow using the cancelrestart
# procedure.
#
proc onematch {number trynumber retry} {
global sampleDirectory
append grm $sampleDirectory "yesno/grammars/yesno.yesno.fsg"
while { $retry < [GetResource maxRetry]} {
  incr retry
  set rc [ExecuteGrammar -timeout 4000 -prompt @yndialagn
  # set rc [ExecuteGrammar -prompt @yndialagn -grammar $grm]
  LogEvEvent -dbg $rc
  switch $rc {
    __HANGUP { return HANGUP }
    __DTMF {
      switch $::callinfo(digits) {
        
          "#" { return [xfer $number $trynumber $retry]
          "*" { LogStat CANCEL
          return 0
          }
        } 
        __RECO {
          switch $::callinfo(anno) {
            "1" { return [xfer $number $trynumber $retry] 
            "0" { LogStat CANCEL
            return 0
            }
        }
        __RECBEP {
          PlayMessage -prompt @afterbeep
        }
        __SILENCE -
        __TIMEOUT { return [xfer $number $trynumber $retry] }
      }
      return DISCON
    }
  }# The Tcl reexamine procedure is called by the nameonly
  # recognition routine to test if the annotation now contains a 
  # complete name and location.
  # After playing the recognition result to the user and logging the 
  # appropriate call state, the procedure transfers to the lookup 
  # procedure to initiate a database query.
  
  proc reexamine {text annotation highconf trynumber} {
      global sampleDirectory connectMessage recoMessage
      LogStat RECNAM
      set list ( )
      set length [llength $text]
      for { set j 0 } { $j < $length } { incr j 1 } {
        set element [lindex $text $j]
        set char [string index $element 0]
if { $char != "_" } {
    lappend list $element
}

set tstr [split $list -]
set playstr [join $tstr " "]
if { $highconf } {
    set playstr [format $connectMessage $playstr]
} else {
    set playstr [format $recoMessage $playstr]
}

set tmpFile [format "%sndsample/tmp.%d.ulw" $sampleDirectory ::callinfo(channel)]
regsub -all {([\])} $tmpFile "/" tmpFile
set rc [PlayMessage -timeout 4000 -prompt [tts $playstr tmpFile]]
switch $rc {
    __HANGUP { return HANGUP }
    __DTMF {
        switch $::callinfo(digits) {
        "#" { set highconf 1 }
        "*" {
            LogStat CANCEL
            return 0
        }
    }
    }
    # The Tcl nameonly procedure is the top of most dialers call
    # flow. It prompts the user with the greeting or "Please say
    # only the full name" prompt, as specified by the calling
    # procedure. The nameonly procedure will transfer to reexamine
    # upon a successful recognition event, or recursively call itself
    # upon failure. If the user explicitly presses a DTMF code, the
    # nameonly procedure will call cancelrestart. This is important
    # for name dialers with locations to allow the user to return to
    # the initial prompt should a location name mis-recognition
    # event occur.

    proc nameonly {retry prompt} {
        global sampleDirectory
        append grm $sampleDirectory "ndsample/grammars/
        ndsample.phone.fsg"
while ( $retry < [GetResource maxRetry] ) {
    incr retry
    LogStat NAMREQ
    set rc [ExecuteGrammar -nointerrupt -prompt $prompt
            -grammar $grm]
    LogEvent -dbg $rc
    switch $rc {
        __RECO {
            set result [reexamine $::callinfo(text)
                        $::callinfo(anno) $::callinfo(confirm) $retry]
            switch $result { "0" {
                set prompt "@nameonly"
            }
            default {
                return $result
            }
        }
        __RECBEP {
            PlayMessage -prompt "@afterbeep"
            set prompt "@nameonly"
        }
        __RECREJ {
            set prompt "@nameonly"
        }
        __SILENCE -
        __TIMEOUT {
            PlayMessage -prompt "@dintmehear"
            set prompt "@nameonly"
        }
        __DTMF {
            LogStat CANCEL
            set prompt "@greeting1"
        }
        __HANGUP { return HANGUP }
        default { error $rc }
    }
    return DISCON
}

# The Tcl callflow procedure is a wrapper for the whole name
dialer
# script. It is called after an AnswerCall event signals that a
# call has been received. It calls locname or nameonly depending
# upon the resource settings.
#
proc callflow {} {
    global VERSION
    set greeting "@greeting1"
    LogEvent -info "$VERSION"
    LogStat IDLE
WaitForCall
LogStat ANSWRD
    return [nameonly 0 $greeting]
}

# The GiveUp procedure is a site-specific script that tells the
# user the name dialer is aborting due to exceeding the maxRetry
# limit. It is used to specify the procedures required to, for
# example, transfer the user to a human operator. The default is
# to do nothing and hang up on the user.
#
proc GiveUp {} {}#

proc SetLanguageMsg {languageCode } {
    global connectMessage recoMessage
    switch $languageCode {
        gr_gr {
            set connectMessage "Sie werden mit %s verbunden."
            set recoMessage "Das System hat den Namen %s erkannt."
        }
        en_uk -
        en_us {
            set connectMessage "You will be connected with %s"
            set recoMessage "The system recognizes the name %s"
        }
        default {
            set connectMessage "%s"
            set recoMessage "%s"
        }
    }
}

UnlockTelLine

    set connectMessage { }
    set recoMessage { }

    set sampleDirectory [GetResource grammarPrefix]
    set language [GetResource Language]
    set language [string tolower $language]
    SetLanguageMsg $language

    catch {
        while (1) {
            set rc [callflow]
            if {($rc == "DISCON")} {
                LogStat GIVEUP
                GiveUp
            }
            LogStat $rc
            DropCall
        }
    } error

235
LogEvent "Call handler exits with $error"
DropCall after 500
LockTelLine
C++ Sample: recotest

This program (recotest) illustrates the use of the ViaVoice Telephony C++ interfaces to develop a grammar-based phone recognition application. It illustrates the following concepts:

- Connecting to and disconnecting from the speech engine.
- Enabling single user-selected precompiled grammars.
- Parsing recognized text and taking appropriate action.

The recotest program is implemented using Microsoft Visual C++ 4.0. The source code is found in the directory <VTT_HOME>/samples/recotest. You can compile the source code and run it. The recotest executable is found in the <VTT_HOME>/bin directory.

The following data files are required to run the sample:

- grammarlist - This file contains the digit grammar.
- test1.xx.ulw - This is a ulaw utterance file, where xx indicates the language of the utterance (for example, test1.us.ulw is used for U.S. English).

The recotest sample first enables the grammars specified in the grammarlist file. It then processes the utterance file. The recognition result should be the digit sequence 9452819.

Building recotest

To build the recotest sample, you must have Microsoft Visual C++ 4.0 or higher installed on your system. At the command line, type the following:

cd <VTT_HOME>/samples/recotest
nmake recotest.mak

where VTT_HOME is the directory into which the Tools were installed.

The file recotest.exe is compiled and linked.

There is also a project file included in this directory so that you can use Developer Studio to extend and debug the recotest sample.

To view the source code for the recotest sample, see recotest Source.

Running recotest

Once recotest is compiled and linked:

- Stop any active telephony processes by using the Process Manager (pm) as a command. Refer to the Process Manager.
- Run recotest.
- Restart the telephony processes.

To run recotest, type:
recotest -c0

where the 0 represents channel 0. This should produce the following output:

AUDIO FILENAME: test1.us.ulw
FINISHED PLAY: test1.us.ulw
Result is : NINE FOUR FIVE TWO EIGHT ONE NINE 9 4 5 2 8 1 9

Here the text symbols NINE FOUR FIVE TWO EIGHT ONE NINE comprise the recognized string, and the numerals 9 4 5 2 8 1 9 represent the returned annotation.
recotest Source

The recotest sample uses the ViaVoice Telephony class ResourceMgr and calls its initialization member function. The Tsm class declares the ResourceMgr object named resm. The ResourceMgr class implements logging of messages from all ViaVoice Telephony processes that are running. The messages are either logged to a process designated file or to the system file, current.log, if the system's logger process is active. Refer to the ResourceMgr **initialize()** function for an explanation. The recotest source file calls in a driver file that helps the user to use the Tsm Class.

Following is the content of recotest.cpp:

```cpp
// Copyright (C) 1998 IBM Corporation
// The following IBM source code is provided to you solely for the purpose of
// assisting you in your development. You may use this code in accordance
// with the IBM License Agreement for the ViaVoice Telephony Tools.
// This Copyright statement may not be removed.

static const char *rcsid = "@(#) $Id: recotest.cpp,v 1.9 1999/07/21 13:00:03 abei Exp $";
#include <appDllDefs.h>
#include <tsm.h>
#include <resourceMgr.h>
extern int recotest(int id, long max_trials);
#define MAX_CHANNELS 99

static char *usage =
"Recotest is a sample driver for the ViaVoice engine.\n"
"USAGE:\n"
" recotest [-?] -c process_num [-t num_trial] [-n num_threads]\n"
"PARAMETERS:\n"
" -? - Displays this help text.\n"
" -c process_number - Specifies the process number\n"
" -t num_trials - Specifies number of trials to run\n"
" -n num_threads - Specifies number of threads to launch\n"
" To use num_threads you must be using the remote architecture\n"
" Also process_num + [0..num_threads] must be unique on a host\n";

class RecoTest : public VttThread{
public:
RecoTest(int tid, char *name, long m):VttThread(tid,name) {
  id = tid;
  max_trials = m;
  doneFlag = new VttSemaphore;
  doneFlag->create(MAX_CHANNELS);
}
unsigned long threadFunction() {
  recotest(id,max_trials);
  doneFlag->signal();
  return 1;
}
```
VttSemaphore *doneFlag;
private:
int id;
long max_trials;
};

int
main(int argc, char *argv[])
{
int cloneId = -1;
ResourceMgr resm;
int max_channels=1;
long max_trials=1;
int pmControl = 1;
{
int opt;
while((opt = getopt(argc,argv,"n:c:t:p")) != -1){
switch(opt){
case 'n':
max_channels = (int)strtol(optarg,0,10);
break;
case 't':
max_trials = (int)strtol(optarg,0,10);
break;
case 'c':
cloneId = (int)strtol(optarg,0,10);
break;
case 'p':
pmControl = 0;
break;
default:
printf ("%s\n", usage);
return -1;
}
}
if(cloneId < 0) {
printf ("%s\n", usage);
return -1;
}
resm.init("main");
resm.initialize("chp",cloneId,pmControl);
RecoTest *rec[MAX_CHANNELS];
for (int i=0; i<max_channels; i++) {
rec[i] = new RecoTest(cloneId + i, "RecoTest", max_trials);
rec[i]->startThread();
}
for (i=0; i<max_channels; i++) {
rec[i]->doneFlag->wait();
}
LOG(dbg,"Exiting the program");
return 0;
Following is a sample of the recotest driver file, driver.cpp:

```c++
#include <appDllDefs.h>
#include <tsm.h>
#include <resourceMgr.h>

int do_recognize(char *ContextId, char *result_str, char *anno,
        int filenum, int threadId);
int start_rec(int filenum, int threadId);
static ResourceMgr resm;
const char *languageCode;

int recotest (int threadId, long max_trials)
{
    resm.init("CallFlow", "ND");

    int done = 0;
    int selected = 0;
    char result_str[5000];
    char anno[TSM_STR_LEN];
    char filename[256];
    Tsm *DialReco= Tsm::TsmInit(threadId);
    languageCode = resm.resRequestStr("Language");
    FILE *fp;

    printf("Thread Id %d Starting ", threadId);
    if (DialReco == NULL) {
        printf("Error initializing engine\n");
        return 1;
    }
    strcpy(result_str, "");
    strcpy(anno, "");

    if ((fp=fopen("grammarlist", "r")) == (FILE *) NULL) {
        printf("Error opening file grammarlist\n");
        return 1;
    }

    // Find an engine without specific name "" can be specified
    if (DialReco->TsmStartAppl ("") == -1) {
        printf("Router is down or failed to connect to session\n");
        return 1;
    }

    printf("Finished start appl\n");
    fflush(stdout);

    while (fscanf(fp, "%s", filename) == 1) {
    }

    return 0;
}
```

VttMemoryBuffer b(filename);
if (DialReco->TsmLoadContext(1, "GRAMMAR_FILE", &b, "Context1") == VttError) {
    printf("Error in calling TsmLoadContext\n");
    return 1;
}
}
fclose(fp);
for (int i=0; i<max_trials; i++) {
    if (do_recognize("Context1", result_str, anno, 1, threadId)) {
        printf("Error in do_recognize\n");
        return 1;
    }
}
if (DialReco->TsmUnloadContext("Context1") == VttError) {
    printf("Error calling TsmUnloadContext\n");
    return 1;
}
if (DialReco->TsmEndAppl () == VttError) {
    printf("Error calling TsmEndAppl\n");
    return 1;
}
if (DialReco->TsmTerm() == VttError) {
    printf("Error in calling TsmTerm\n");
    return 1;
}
printf("Thread Id %d Finished\n", threadId);
return 0;

int do_recognize(char *contextId, char *result_str, char *anno, int filenum, int threadId)
{
    char reco_result[3000];
    char reco_nbest[3000], arg[3000];
    int done, status = 0;
    int word_num=0, prev_word_num=-1;
    Audio *aud = Audio::AudioInit(threadId);
    Tsm *DialReco= Tsm::TsmInit(threadId);
    float scr = 0;
    float score = 0;
    reco_result[0]=0;
    static int uttno=1;
    if (DialReco == NULL) {
        printf("Error initializing engine in do_recognize\n");
        return 1;
    }
    if (DialReco->TsmEnableVocab(contextId, 0) == VttError) {
        printf("Error calling TsmEnableVocab\n");
        return 1;
    }
    // Further code execution...
}
return 1;
}
if (DialReco->TsmStartReco(10000) == VttError) {
    printf ("Error calling TsmStartReco\n");
    return 1;
}
if (start_rec(filenum, threadId)) {
    printf ("Error in calling start_rec ... check audio file etc.\n");
    return 1;
}
done = 0;
while (!done) {
    if (DialReco->TsmWait(100, &status, arg, 3000) == VttError) {
        LOG(dbg,"Serious error: possibly server went away " << status);
        done = 1;
    }
    if (status & TSM_EOU) {
        LOG(dbg,"STATUS IS TSM_EOU: " << status);
        done = 1;
        if (DialReco->TsmGetText(reco_result, 3000, &score) == VttError) {
            printf ("Error in calling TsmGetText\n");
            return 1;
        }
        // Here pick up the second best result.
        if (DialReco->TsmGetNbest(2,reco_nbest, 3000, &scr) == VttError) {
            printf ("Error in calling TsmGetNbest\n");
            return 1;
        }
        if (DialReco->TsmGetAnno(TSM_ALL_ANN, anno, TSM_STR_LEN) == VttError) {
            printf ("Error in calling TsmGetAnnot\n");
            return 1;
        }
        char *rejStr = (status&TSM_REJECTED) ? "REJECT" : "ACCEPT";
        printf ("%4d best [%s] %s %f %s
", uttno, rejStr,
                reco_result, score, anno);
        printf ("%4d best(2) [%s] %s %f
", uttno, rejStr,
                reco_nbest, scr);
        uttno++;
    }
    if (status & TSM_FAILURE) {
        LOG(dbg,"STATUS IS TSM_FAILURE: " << status);
        printf ("status = %x -- TSM_FAILURE\n", status);
        done = 1;
    }
    if (strlen(arg) > 0) {
        printf ("%s\n", arg);
    }
}
if (DialReco->TsmStopReco() == VttError) {
    printf ("Error in calling TsmStopReco\n");
    return 1;
}
if (DialReco->TsmDisableVocab(contextId, 0) == VttError) {
    printf ("Error in calling TsmDisableVocab\n");
    return 1;
}
if (DialReco->TsmWait(-1, &status) == VttError) {
    printf ("Error in TsmWait call in waiting for DisableVocab\n");
    return 1;
}
LOG(dbg,"RECO RESULT: res " << reco_result << " score " << score);
strcpy(result_str, reco_result);
return 0;
}

int start_rec(int filenum, int threadId) {
    FILE *fp;
    char buf[1200];
    char langCodeBuf[3];
    char filename[30];
    int xxx;

    strcpy(langCodeBuf,"us");
    if(languageCode) {
        if(strlen(languageCode) == 5) {
            strcpy(langCodeBuf,languageCode + 3);
        }
    }
    sprintf (filename,"test%d.%s.ulw", filenum,langCodeBuf);

    printf ("AUDIO FILENAME: %s\n", filename);
    if ((fp=fopen(filename, "rb")) == (FILE *) NULL) {
        printf ("Error opening audio file %s\n", filename);
        return 1;
    }
    xxx = 1;
    while (!feof(fp) && xxx) {
        fread(buf, 1, 1200, fp);
        if (Audio::aud_write(threadId, buf, (unsigned int) 1200,0,0) == -1)
            xxx = 0;
    }
    fclose (fp);
    printf ("FINISHED PLAY: %s\n", filename);
    filenum++;

    // One last write to indicate end of audio
    // if audio didn't tell us to stop
    // IMPORTANT: First argument is now the tsm id
    if (xxx == 1)
        Audio::aud_write(threadId, buf, (unsigned int) -1,0,0);

    return 0;
}
Sample Grammars

The ViaVoice Telephony Tools provide samples of application grammars that you may find useful. You can use and extend these grammars to quickly and easily develop phone grammars for your application. Sample grammars are provided for the following languages:

- **U.S. English**
- **U.K. English**
- **French**
- **German**

The data provided for each grammar resides in the directory:

\<VTT_HOME\>\app\<LL_LL>\<comp_name>\n
where:

\<VTT_HOME> is the directory into which the ViaVoice Telephony Tools were installed.

\<LL_LL> specifies the language for which the sample grammar is intended. **LL_LL** can be one of the following:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

\<comp_name> is the subdirectory name for the component (for example, “banking”). This directory contains the following subdirectories:

- **baseforms** - This directory contains the pronunciation dictionary file for the component. This file contains pronunciations for all of the words and phrases that are to be recognized. The dictionary is a plain text file named **baseforms**. This directory also contains the optional special pronunciation dictionary, called **baseforms.special**. When baseforms.special is present, the engine searches it first when it looks for pronunciations. Use a text editor to create this file. It consists of one pronunciation per line in the following format:

  `<word> | <pronunciation>`

  where:

  `<pronunciation>` is the pronunciation (using the IBM phonetic alphabet) of `<word>`.

Examples:

AIX | EY AY EH K S
baseforms\<comp_name> - The pronunciation pool files required by the speech recognition engine.

grammars - A component’s grammar source files (.bnf) and compiled grammar files (.fsg).

You can modify the .bnf files for any of the sample grammars. If you modify the .bnf files, you need to recompile the grammars and regenerate pronunciations. To do this, use the following utilities:

1. vtbnfc - To produce the compiled .fsg files from the .bnf files.
2. makeBsfms - To creates a source pronunciation dictionary for the grammars.
3. makePol - To compile the pronunciation pool files that the engine uses.

Note that the output from all of these utilities must be present to have a complete vocabulary for your application. Be sure to test your grammars when your application is complete. Refer to Testing Your Phone Grammars.
Sample Grammar Vocabularies - U.S. English

This section describes the words and phrases that are supported in the sample grammars for U.S. English that are included with the ViaVoice Telephony Tools. The following sample grammars are provided:

- Banking
- Commands
- Dates
- Digits and digComp
- Dollars
- Natural Numbers
- Spelling
- Times
- Voice Activated Network Control
- Voice Mail
- Yes/No
Banking - U.S. English

The purpose of the banking sample grammar is to recognize words or phrases which are commonly found in banking applications.

In U.S. English, the following words and phrases are supported:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>205</td>
</tr>
<tr>
<td>Auto</td>
<td>218</td>
</tr>
<tr>
<td>Automobile</td>
<td>219</td>
</tr>
<tr>
<td>Balance</td>
<td>209</td>
</tr>
<tr>
<td>Car</td>
<td>220</td>
</tr>
<tr>
<td>CD</td>
<td>221</td>
</tr>
<tr>
<td>Certificate</td>
<td>217</td>
</tr>
<tr>
<td>Certificate of deposit</td>
<td>222</td>
</tr>
<tr>
<td>Charge card</td>
<td>201</td>
</tr>
<tr>
<td>Checking</td>
<td>206</td>
</tr>
<tr>
<td>Credit</td>
<td>229</td>
</tr>
<tr>
<td>Day</td>
<td>215</td>
</tr>
<tr>
<td>Deposit</td>
<td>204</td>
</tr>
<tr>
<td>Dividends</td>
<td>227</td>
</tr>
<tr>
<td>Fixed</td>
<td>210</td>
</tr>
<tr>
<td>Information</td>
<td>216</td>
</tr>
<tr>
<td>Interest</td>
<td>228</td>
</tr>
<tr>
<td>Investment</td>
<td>230</td>
</tr>
<tr>
<td>Loan</td>
<td>202</td>
</tr>
<tr>
<td>Month</td>
<td>214</td>
</tr>
<tr>
<td>Term</td>
<td>Page</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>Mortgage</td>
<td>203</td>
</tr>
<tr>
<td>Rate</td>
<td>212</td>
</tr>
<tr>
<td>Regular</td>
<td>223</td>
</tr>
<tr>
<td>Savings</td>
<td>207</td>
</tr>
<tr>
<td>Special</td>
<td>224</td>
</tr>
<tr>
<td>Statement</td>
<td>231</td>
</tr>
<tr>
<td>Stocks</td>
<td>225</td>
</tr>
<tr>
<td>Transfer</td>
<td>208</td>
</tr>
<tr>
<td>Variable</td>
<td>211</td>
</tr>
<tr>
<td>Withdrawals</td>
<td>226</td>
</tr>
<tr>
<td>Year</td>
<td>213</td>
</tr>
</tbody>
</table>
The commands grammar recognizes commonly used command words. The following commands are supported in U.S. English:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
<td>101</td>
</tr>
<tr>
<td>Cancel</td>
<td>107</td>
</tr>
<tr>
<td>Exit</td>
<td>105</td>
</tr>
<tr>
<td>Help</td>
<td>106</td>
</tr>
<tr>
<td>Quit</td>
<td>104</td>
</tr>
<tr>
<td>Repeat</td>
<td>103</td>
</tr>
<tr>
<td>Stop</td>
<td>102</td>
</tr>
</tbody>
</table>
Dates - U.S. English

The purpose of the dates grammar is to recognize different date phrases. A date phrase takes the following format:

[DAY] MONTH ["the"] DAY_NUMBER [[CENTURY] YEAR]

Where:

**DAY** can be any day name: "Sunday" - "Saturday"

**MONTH** can be any month name: "January" - "December"

**DAY_NUMBER** can be any number: "one" - "thirty one" or "first" - "thirty first"

**CENTURY** can be one of the following: "nineteen", "nineteen hundred", "twenty", "two thousand" or "two thousand and".

**YEAR** can be one of the following:

- A number from "eleven" through "ninety-nine", which is interpreted 1911 - 1999
- A number from "one" through "ten" which is interpreted as 2000 - 2010

Items in square brackets ([ ]) are optional.

When this grammar is used, a special parser is activated that produces, in addition to the recognized text, an easy to use parsed string.

When date utterances are recognized, they are parsed and returned to the application as an annotated string. For more information on the parsing process, see Grammar Parser.
Digits and digComp Grammars - U.S. English

The digits grammars are designed to provide recognition of digit strings of varied length. These include telephone numbers, credit card numbers, personal identification numbers, and other digit sequences. Dollar amounts and natural numbers are not included in this grammar.

In all of these grammars, please note that [DIGIT] can be any number between "zero" and "nine", including "oh". Any number can occur in any position. No grouped teens, hundreds, or thousands are allowed in these grammars.

The digits grammars are designed to be used by applications that recognize some numbers but primarily recognize text. The digComp grammars are designed for use by digit-only applications and provide precision recognition of digits. The number of grammars, names of the grammars, and annotations returned are the same for digits and digComp. The digits grammars reside in the `<VTT_HOME>\app\En_US\digit` directory and the digComp grammars reside in the `<VTT_HOME>\app\En_US\digComp` directory. The digComp applet has a baseforms.special file that must be used during an application build.

Use of the digits grammars versus the digComp grammars is determined in part by the value of the `taskId` resource. Ensure that the resource is set properly for the grammars you want to use. A mismatch in grammar selection and/or resource value may cause system performance degradation.
**Dollars - U.S. English**

The dollars grammars are designed to provide recognition of dollar amounts between $1 and $999,999 and dollar and cents amounts from $0.01 through $999,999.99. In all the dollars grammars, please note that the digit strings that comprise the numbers are not recognized. Grouped teens, hundreds, or thousands are allowed in these grammars.

The following format is used:

[DOLLAR_AMOUNT ["dollars" or "dollar"] ["and"]] CENTS_AMOUNT ["cents" or "cent"]

Where:

**DOLLAR_AMOUNT** must be the name of a number from "zero" or "oh" through "nine hundred ninety-nine thousand nine hundred ninety-nine". "two three five four two" is not recognized as a dollar amount.

**CENTS_AMOUNT** must be "no", or "zero" through "ninety-nine".

Some examples of valid dollar utterances are "eight dollars," "twenty-three dollars and zero cents," "one twenty-four and six cents."

The dollars and cents (DAC) grammar and the whole dollar amount (WDA) grammars can be found in the `<VTT_HOME>\app\digit` directory.
Natural Numbers - U.S. English

The natural numbers grammar recognizes any integer number from 1 to 999,999. The user must say the number properly using hundreds and thousands to indicate the positions. Using this grammar, for example, it is not proper to say twenty-two hundred. It is correct to say two thousand two hundred.

The format of a natural number is:

**NUMBER**

Where:

**NUMBER** must be the name of an integer from "one" through "nine hundred ninety-nine thousand nine hundred and ninety-nine".

Some examples of valid natural number utterances are "forty two," "five thousand five hundred and two," and "one hundred and forty eight."

Some examples of invalid natural number utterances are "one oh three" and "fifty five hundred and two."
Spelling Mode - U.S. English

The spelling mode grammars allow the user to spell the words (and phrases) that are to be recognized. Such a mode is particularly important when the user doesn't know how to pronounce the word (or phrase) correctly.

The ViaVoice Telephony Tools contain spelling grammars for all of the sample grammars. The following are examples for U.S. English:

<table>
<thead>
<tr>
<th>Utterance</th>
<th>Recognized Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>A L B E R T E I N S T E I N</td>
<td>albert einstein</td>
</tr>
<tr>
<td>B E G I N</td>
<td>begin</td>
</tr>
<tr>
<td>J I M M Y C A R T E R</td>
<td>jimmy carter</td>
</tr>
</tbody>
</table>
Times - U.S. English

The purpose of the times grammar is to recognize different time phrases. In U.S. English, a time phrase takes the following format:

[APPROX] HOUR1 ["O'CLOCK or MINUTE1] [TIME1] [ZONE]

or

[APPROX] MINUTE2 WORD1 HOUR2 [TIME2] [ZONE]

Where:

APPROX must be one of the following: "approximately", around", "about" or "roughly".

HOUR1 must be a number from "one" through "twelve".

MINUTE1 must be a number from "zero" through "fifty nine".

TIME1 must be one of the following: "A.M.", "P.M.", "in the morning", "in the evening", "in the afternoon" or "at night".

ZONE must be one of the following: "eastern time", "eastern standard time", "central time", "central standard time", "rocky mountain time", "rocky mountain standard time", "mountain time", "mountain standard time", "pacific time", "pacific standard time", "western time", "western standard time", "daylight time", "daylight saving time" or "daylight savings time".

MINUTE2 must be a number from "zero" through "fifty nine" followed by "minutes", or one of the following: "quarter", "a quarter" or "half".

WORD1 must be one of the following: "til", "to", "of", "past", "before", "until" or "after". ("Before" and "until" are not allowed following "quarter", "a quarter" or "half".)

HOUR2 must be a number from "one" through "twelve", optionally followed by "o'clock", or one of the following: "noon", "midnight", "twelve noon" or "twelve midnight".

TIME2 must be one of the following: "A.M.", "P.M.", "in the morning", "in the evening", "in the afternoon" or "at night". ("A.M." and "P.M." are not allowed following "noon", "midnight", "twelve noon" or "twelve midnight").

Items in square brackets ([ ]) are optional.

When this grammar is used, a special parser is activated that produces, in addition to the recognized text, an easy to use parsed string.

When time utterances (in either format) are recognized, they are parsed returned to the application as an annotated string. For more information on the grammar parsing process, see Grammar Parser.
Voice Activated Network Control - U.S. English

The purpose of the voice activated network control grammar is to recognize the command words used to control network features provided by the telephone companies. Also included with this grammar are words common to personal vocabularies. The following words and phrases are supported in U.S. English:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add name</td>
<td>40</td>
</tr>
<tr>
<td>Call screening</td>
<td>26</td>
</tr>
<tr>
<td>Cancel call forwarding</td>
<td>25</td>
</tr>
<tr>
<td>Cancel call waiting</td>
<td>24</td>
</tr>
<tr>
<td>Call forwarding</td>
<td>20</td>
</tr>
<tr>
<td>Call return</td>
<td>21</td>
</tr>
<tr>
<td>Call trace</td>
<td>30</td>
</tr>
<tr>
<td>Delete name</td>
<td>41</td>
</tr>
<tr>
<td>Disable anonymous caller rejection</td>
<td>29</td>
</tr>
<tr>
<td>Enable anonymous caller rejection</td>
<td>28</td>
</tr>
<tr>
<td>Name that number</td>
<td>22</td>
</tr>
<tr>
<td>Per call blocking</td>
<td>27</td>
</tr>
<tr>
<td>Repeat call</td>
<td>23</td>
</tr>
<tr>
<td>Review list</td>
<td>42</td>
</tr>
</tbody>
</table>
**Voice Mail - U.S. English**

The purpose of the voice mail sample grammar is to recognize words or phrases commonly found in voice mail applications. In U.S. English, the following words are supported for voice mail:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>327</td>
</tr>
<tr>
<td>Assistant</td>
<td>325</td>
</tr>
<tr>
<td>Asterisk</td>
<td>336</td>
</tr>
<tr>
<td>Attributes</td>
<td>328</td>
</tr>
<tr>
<td>Audio name</td>
<td>346</td>
</tr>
<tr>
<td>Call</td>
<td>315</td>
</tr>
<tr>
<td>Comment</td>
<td>344</td>
</tr>
<tr>
<td>Copy</td>
<td>313</td>
</tr>
<tr>
<td>Delete</td>
<td>308</td>
</tr>
<tr>
<td>Disconnect</td>
<td>320</td>
</tr>
<tr>
<td>Distribution</td>
<td>345</td>
</tr>
<tr>
<td>Diversion</td>
<td>324</td>
</tr>
<tr>
<td>E-mail</td>
<td>340</td>
</tr>
<tr>
<td>Envelope</td>
<td>322</td>
</tr>
<tr>
<td>Expert</td>
<td>329</td>
</tr>
<tr>
<td>Fax</td>
<td>339</td>
</tr>
<tr>
<td>Forward</td>
<td>304</td>
</tr>
<tr>
<td>Future</td>
<td>332</td>
</tr>
<tr>
<td>Goodbye</td>
<td>348</td>
</tr>
<tr>
<td>Greeting</td>
<td>302</td>
</tr>
<tr>
<td>Hash</td>
<td>337</td>
</tr>
<tr>
<td>Header</td>
<td>321</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Language</td>
<td>326</td>
</tr>
<tr>
<td>Listen</td>
<td>317</td>
</tr>
<tr>
<td>Menu</td>
<td>343</td>
</tr>
<tr>
<td>Message</td>
<td>303</td>
</tr>
<tr>
<td>Notification</td>
<td>333</td>
</tr>
<tr>
<td>Operator</td>
<td>331</td>
</tr>
<tr>
<td>Outgoing</td>
<td>319</td>
</tr>
<tr>
<td>Pager</td>
<td>335</td>
</tr>
<tr>
<td>Password</td>
<td>323</td>
</tr>
<tr>
<td>Pause</td>
<td>314</td>
</tr>
<tr>
<td>Personal</td>
<td>316</td>
</tr>
<tr>
<td>Phone</td>
<td>334</td>
</tr>
<tr>
<td>Play</td>
<td>306</td>
</tr>
<tr>
<td>Pound</td>
<td>338</td>
</tr>
<tr>
<td>Receive</td>
<td>318</td>
</tr>
<tr>
<td>Record</td>
<td>301</td>
</tr>
<tr>
<td>Replay</td>
<td>312</td>
</tr>
<tr>
<td>Reverse</td>
<td>305</td>
</tr>
<tr>
<td>Review</td>
<td>310</td>
</tr>
<tr>
<td>Save</td>
<td>307</td>
</tr>
<tr>
<td>Send</td>
<td>309</td>
</tr>
<tr>
<td>Skip</td>
<td>311</td>
</tr>
<tr>
<td>Speak</td>
<td>341</td>
</tr>
<tr>
<td>Standard</td>
<td>330</td>
</tr>
<tr>
<td>Star</td>
<td>347</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>Who from</td>
<td>342</td>
</tr>
</tbody>
</table>
Yes/No - U.S. English

The purpose of the yes_no grammar is to recognize words or phrases which mean yes or no. The following words are supported in U.S. English:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>0</td>
</tr>
<tr>
<td>NOPE</td>
<td>0</td>
</tr>
<tr>
<td>OK</td>
<td>1</td>
</tr>
<tr>
<td>SURE</td>
<td>1</td>
</tr>
<tr>
<td>YEAH</td>
<td>1</td>
</tr>
<tr>
<td>YEP</td>
<td>1</td>
</tr>
<tr>
<td>YES</td>
<td>1</td>
</tr>
</tbody>
</table>
Sample Grammar Vocabularies - U.K. English

This section describes the words and phrases that are supported in the sample grammars for UK English and are included with the ViaVoice Telephony Tools. The following sample grammars are provided:

- Banking
- Commands
- Dates
- Digits and digComp
- Spelling
- Times
- Voice Activated Network Control
- Voice Mail
- Yes/No
Banking - U.K. English

The purpose of the banking sample grammar is to recognize words or phrases which are commonly found in banking applications.

In U.K. English, the following words and phrases are supported:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT NUMBER</td>
<td>205</td>
</tr>
<tr>
<td>AUTO</td>
<td>218</td>
</tr>
<tr>
<td>AUTOMOBILE</td>
<td>219</td>
</tr>
<tr>
<td>BALANCE</td>
<td>209</td>
</tr>
<tr>
<td>CAR</td>
<td>220</td>
</tr>
<tr>
<td>CD</td>
<td>221</td>
</tr>
<tr>
<td>CERTIFICATE</td>
<td>217</td>
</tr>
<tr>
<td>CERTIFICATE OF DEPOSIT</td>
<td>222</td>
</tr>
<tr>
<td>CHARGE CARD</td>
<td>201</td>
</tr>
<tr>
<td>CHECKING</td>
<td>206</td>
</tr>
<tr>
<td>CREDIT</td>
<td>229</td>
</tr>
<tr>
<td>DAY</td>
<td>215</td>
</tr>
<tr>
<td>DEPOSIT</td>
<td>204</td>
</tr>
<tr>
<td>DIVIDENDS</td>
<td>227</td>
</tr>
<tr>
<td>FIXED</td>
<td>210</td>
</tr>
<tr>
<td>INFORMATION</td>
<td>216</td>
</tr>
<tr>
<td>INTEREST</td>
<td>228</td>
</tr>
<tr>
<td>INVESTMENT</td>
<td>230</td>
</tr>
<tr>
<td>LOAN</td>
<td>202</td>
</tr>
<tr>
<td>MONTH</td>
<td>214</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>MORTGAGE</td>
<td>203</td>
</tr>
<tr>
<td>RATE</td>
<td>212</td>
</tr>
<tr>
<td>REGULAR</td>
<td>223</td>
</tr>
<tr>
<td>SAVINGS</td>
<td>207</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>224</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>231</td>
</tr>
<tr>
<td>STOCKS</td>
<td>225</td>
</tr>
<tr>
<td>TRANSFER</td>
<td>208</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>211</td>
</tr>
<tr>
<td>WITHDRAWALS</td>
<td>226</td>
</tr>
<tr>
<td>YEAR</td>
<td>213</td>
</tr>
</tbody>
</table>
The commands grammar recognizes commonly used command words. The following commands are supported in U.K. English:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>101</td>
</tr>
<tr>
<td>CANCEL</td>
<td>107</td>
</tr>
<tr>
<td>EXIT</td>
<td>105</td>
</tr>
<tr>
<td>HELP</td>
<td>106</td>
</tr>
<tr>
<td>QUIT</td>
<td>104</td>
</tr>
<tr>
<td>REPEAT</td>
<td>103</td>
</tr>
<tr>
<td>STOP</td>
<td>102</td>
</tr>
</tbody>
</table>
Dates - U.K. English

The purpose of the dates grammar is to recognize different date phrases. A date phrase takes the following format:

\[ \text{<dates>} = \]

\[ \text{<day_name>? <the>? <day_number> <of> <month> <cent_year>?} \]

Where:

\[ \text{<DAY_NAME> can be any day name shown in the following chart.} \]

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUNDAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>MONDAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>TUESDAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>WEDNESDAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>THURSDAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>FRIDAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>SATURDAY</td>
<td>1_dates</td>
</tr>
</tbody>
</table>

\[ \text{<of> can be:} \]

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>of</td>
<td>1_dates</td>
</tr>
</tbody>
</table>

\[ \text{<MONTH> can be any month name shown in the following chart.} \]

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>JANUARY</td>
<td>1_dates</td>
</tr>
<tr>
<td>FEBRUARY</td>
<td>1_dates</td>
</tr>
<tr>
<td>MARCH</td>
<td>1_dates</td>
</tr>
<tr>
<td>APRIL</td>
<td>1_dates</td>
</tr>
<tr>
<td>MONTH</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>MAY</td>
<td>1_dates</td>
</tr>
<tr>
<td>JUNE</td>
<td>1_dates</td>
</tr>
<tr>
<td>JULY</td>
<td>1_dates</td>
</tr>
<tr>
<td>AUGUST</td>
<td>1_dates</td>
</tr>
<tr>
<td>SEPTEMBER</td>
<td>1_dates</td>
</tr>
<tr>
<td>OCTOBER</td>
<td>1_dates</td>
</tr>
<tr>
<td>NOVEMBER</td>
<td>1_dates</td>
</tr>
<tr>
<td>DECEMBER</td>
<td>1_dates</td>
</tr>
</tbody>
</table>

<the> as shown below:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE</td>
<td>1_dates</td>
</tr>
</tbody>
</table>

<DAY_NUMBER> can be specified as shown in the following chart.

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWO</td>
<td>1_dates</td>
</tr>
<tr>
<td>THREE</td>
<td>1_dates</td>
</tr>
<tr>
<td>FOUR</td>
<td>1_dates</td>
</tr>
<tr>
<td>FIVE</td>
<td>1_dates</td>
</tr>
<tr>
<td>SIX</td>
<td>1_dates</td>
</tr>
<tr>
<td>SEVEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>EIGHT</td>
<td>1_dates</td>
</tr>
<tr>
<td>NINE</td>
<td>1_dates</td>
</tr>
<tr>
<td>Number</td>
<td>1_dates</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>TEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>ELEVEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWELVE</td>
<td>1_dates</td>
</tr>
<tr>
<td>THIRTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>FOURTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>FIFTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>SIXTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>SEVENTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>EIGHTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>NINETEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY ONE</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY TWO</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY THREE</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY FOUR</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY FIVE</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY SIX</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY SEVEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY EIGHT</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY NINE</td>
<td>1_dates</td>
</tr>
<tr>
<td>THIRTY</td>
<td>1_dates</td>
</tr>
<tr>
<td>THIRTY ONE</td>
<td>1_dates</td>
</tr>
<tr>
<td>FIRST</td>
<td>1_dates</td>
</tr>
<tr>
<td></td>
<td>1_dates</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
</tr>
<tr>
<td>SECOND</td>
<td></td>
</tr>
<tr>
<td>THIRD</td>
<td></td>
</tr>
<tr>
<td>FOURTH</td>
<td></td>
</tr>
<tr>
<td>FIFTH</td>
<td></td>
</tr>
<tr>
<td>SIXTH</td>
<td></td>
</tr>
<tr>
<td>SEVENTH</td>
<td></td>
</tr>
<tr>
<td>EIGHTH</td>
<td></td>
</tr>
<tr>
<td>NINTH</td>
<td></td>
</tr>
<tr>
<td>TENTH</td>
<td></td>
</tr>
<tr>
<td>ELEVENTH</td>
<td></td>
</tr>
<tr>
<td>TWELFTH</td>
<td></td>
</tr>
<tr>
<td>THIRTEENTH</td>
<td></td>
</tr>
<tr>
<td>FOURTEENTH</td>
<td></td>
</tr>
<tr>
<td>FIFTEENTH</td>
<td></td>
</tr>
<tr>
<td>SIXTEENTH</td>
<td></td>
</tr>
<tr>
<td>SEVENTEENTH</td>
<td></td>
</tr>
<tr>
<td>EIGHTEENTH</td>
<td></td>
</tr>
<tr>
<td>NINTEENTH</td>
<td></td>
</tr>
<tr>
<td>TWENTIETH</td>
<td></td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates FIRST</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates SECOND</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates THIRD</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates FOURTH</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates FIFTH</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates SIXTH</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates SEVENTH</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates EIGHTH</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates NINTH</td>
</tr>
<tr>
<td>THIRTIETH</td>
<td>1_dates</td>
</tr>
<tr>
<td>THIRTY</td>
<td>1_dates FIRST</td>
</tr>
</tbody>
</table>

<CENT_YEAR> can be any number as specified in the following:

<cent_year> = <cent> <year>

<cent> =

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NINETEEN</td>
<td>1_dates</td>
<td></td>
</tr>
<tr>
<td>NINETEEN</td>
<td>1_dates HUNDRED</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWENTY</td>
<td>1_dates</td>
<td></td>
</tr>
<tr>
<td>TWO</td>
<td>1_dates THOUSAND</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWO</td>
<td>1_dates THOUSAND</td>
<td>1_dates AND</td>
</tr>
</tbody>
</table>

<YEAR> is a combination of 2 number categories.

year = <num1> and <num2>

Where <num1> can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWO</td>
<td>1_dates</td>
</tr>
<tr>
<td>THREE</td>
<td>1_dates</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>FOUR</td>
<td>1_dates</td>
</tr>
<tr>
<td>FIVE</td>
<td>1_dates</td>
</tr>
<tr>
<td>SIX</td>
<td>1_dates</td>
</tr>
<tr>
<td>SEVEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>EIGHT</td>
<td>1_dates</td>
</tr>
<tr>
<td>NINE</td>
<td>1_dates</td>
</tr>
<tr>
<td>TEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>ELEVEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>TWELVE</td>
<td>1_dates</td>
</tr>
<tr>
<td>THIRTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>FOURTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>FIFTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>SIXTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>SEVENTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>EIGHTEEN</td>
<td>1_dates</td>
</tr>
<tr>
<td>NINETEEN</td>
<td>1_dates</td>
</tr>
</tbody>
</table>

<num2> = <num3> and <num4>

<num3> =

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWENTY</td>
<td>1_dates</td>
</tr>
<tr>
<td>THIRTY</td>
<td>1_dates</td>
</tr>
<tr>
<td>FORTY</td>
<td>1_dates</td>
</tr>
<tr>
<td>FIFTY</td>
<td>1_dates</td>
</tr>
<tr>
<td>SIXTY</td>
<td>1_dates</td>
</tr>
</tbody>
</table>
When this grammar is used, a special parser is activated that produces, in addition to the recognized text, an easy to use parsed string.

When date utterances are recognized, they are parsed and returned to the application as an annotated string. For more information on the parsing process, see Grammar Parser.
Digits and digComp Grammars - U.K. English

The digits grammars are designed to provide recognition of digit strings of varied length. These include telephone numbers, credit card numbers, personal identification numbers, and other digit sequences. Dollar amounts and natural numbers are not included in this grammar.

In all of these grammars, please note that [DIGIT] can be any number between "zero" and "nine", including "oh". Any number can occur in any position. No grouped teens, hundreds, or thousands are allowed in these grammars.

The digits grammars are designed to be used by applications that recognize some numbers but primarily recognize text. The digComp grammars are designed for use by digit-only applications and provide precision recognition of digits. The number of grammars, names of the grammars, and annotations returned are the same for digits and digComp. The digits grammars reside in the `<VTT_HOME>\app\En_UK\digit` directory and the digComp grammars reside in the `<VTT_HOME>\app\En_UK\digComp` directory. The digComp applet has a baseforms.special file that must be used during an application build.

Use of the digits grammars versus the digComp grammars is determined in part by the value of the `taskid` resource. Ensure that the resource is set properly for the grammars you want to use. A mismatch in grammar selection and/or resource value may cause system performance degradation.

\[\text{<price>} = \text{<dollars>} | \text{<dollars>} \text{ and? } \text{<cents>} | \text{<cents>}\]

Where \text{<dollars>} can be:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>1_pound</td>
<td>1D</td>
</tr>
<tr>
<td>02_999999</td>
<td>1_dates</td>
<td>1D</td>
</tr>
</tbody>
</table>

Where \text{<cents>} can be:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>1_pence</td>
<td>1C</td>
</tr>
<tr>
<td>02_99</td>
<td>1_pence</td>
<td>1C</td>
</tr>
</tbody>
</table>

Where \text{<02_99>} = \text{<02_9>} | \text{<10_99>}.
Where `<02_9>` can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>three</td>
<td>3</td>
</tr>
<tr>
<td>four</td>
<td>4</td>
</tr>
<tr>
<td>five</td>
<td>5</td>
</tr>
<tr>
<td>six</td>
<td>6</td>
</tr>
<tr>
<td>seven</td>
<td>7</td>
</tr>
<tr>
<td>eight</td>
<td>8</td>
</tr>
<tr>
<td>nine</td>
<td>9</td>
</tr>
</tbody>
</table>

Where `<1_9>` can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>2</td>
</tr>
<tr>
<td>three</td>
<td>3</td>
</tr>
<tr>
<td>four</td>
<td>4</td>
</tr>
<tr>
<td>five</td>
<td>5</td>
</tr>
<tr>
<td>six</td>
<td>6</td>
</tr>
<tr>
<td>seven</td>
<td>7</td>
</tr>
<tr>
<td>eight</td>
<td>8</td>
</tr>
<tr>
<td>nine</td>
<td>9</td>
</tr>
</tbody>
</table>
Where $<10\_99> = <10\_19> \mid <20\_99>.$

$<10\_19>$ can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ten</td>
<td>10</td>
</tr>
<tr>
<td>eleven</td>
<td>11</td>
</tr>
<tr>
<td>twelve</td>
<td>12</td>
</tr>
<tr>
<td>thirteen</td>
<td>13</td>
</tr>
<tr>
<td>fourteen</td>
<td>14</td>
</tr>
<tr>
<td>fifteen</td>
<td>15</td>
</tr>
<tr>
<td>sixteen</td>
<td>16</td>
</tr>
<tr>
<td>seventeen</td>
<td>17</td>
</tr>
<tr>
<td>eighteen</td>
<td>18</td>
</tr>
<tr>
<td>nineteen</td>
<td>19</td>
</tr>
</tbody>
</table>

Where $<20\_90>$ can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>twenty</td>
<td>20</td>
</tr>
<tr>
<td>thirty</td>
<td>30</td>
</tr>
<tr>
<td>forty</td>
<td>40</td>
</tr>
<tr>
<td>fifty</td>
<td>50</td>
</tr>
<tr>
<td>sixty</td>
<td>60</td>
</tr>
<tr>
<td>seventy</td>
<td>70</td>
</tr>
<tr>
<td>eighty</td>
<td>80</td>
</tr>
<tr>
<td>ninety</td>
<td>90</td>
</tr>
</tbody>
</table>
\(<20\_99> = \langle 20\_90 \rangle <1\_9>\).

\(<100\_999> = \langle 1\_99\rangle?\text{hundred} \langle \text{after}_100\rangle\).

\(<\text{after}_100> = \text{and}\? <1\_99>\).

\(<1\_99> = <1\_9> | <10\_19> | <20\_99>.

\(<1\_999> = <1\_99> | <100\_999>.

\(<1000\_999999> = <1\_999>\?\text{thousand}\ <1\_999>\).

\(<02\_999999> = <02\_99> | <100\_999> | <1000\_999999>\).
Spelling Mode - U.K. English

The spelling mode grammars allow the user to spell the words (and phrases) that are to be recognized. Such a mode is particularly important when the user doesn't know how to pronounce the word (or phrase) correctly.

The ViaVoice Telephony Tools contain spelling grammars for all of the sample grammars. The following are examples for U.K. English.

**Banking:**

<table>
<thead>
<tr>
<th>Utterance</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNT NUMBER</td>
<td>205</td>
</tr>
<tr>
<td>AUTO</td>
<td>218</td>
</tr>
<tr>
<td>AUTO</td>
<td>218</td>
</tr>
<tr>
<td>MOBILE</td>
<td>219</td>
</tr>
<tr>
<td>BALANCE</td>
<td>209</td>
</tr>
<tr>
<td>CAR</td>
<td>220</td>
</tr>
<tr>
<td>CD</td>
<td>221</td>
</tr>
<tr>
<td>CERTIFICATE</td>
<td>217</td>
</tr>
<tr>
<td>CERTIFICATE OF DEPOSIT</td>
<td>222</td>
</tr>
<tr>
<td>CHARGE CARD</td>
<td>201</td>
</tr>
<tr>
<td>CHECKING</td>
<td>206</td>
</tr>
<tr>
<td>CREDIT</td>
<td>229</td>
</tr>
<tr>
<td>DAY</td>
<td>215</td>
</tr>
<tr>
<td>DEPOSIT</td>
<td>204</td>
</tr>
<tr>
<td>DIVIDENDS</td>
<td>227</td>
</tr>
<tr>
<td>FIXED</td>
<td>210</td>
</tr>
<tr>
<td>INFORMATION</td>
<td>216</td>
</tr>
<tr>
<td>INTEREST</td>
<td>228</td>
</tr>
<tr>
<td>Utterance</td>
<td>Annotation</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>INVESTMENT</td>
<td>230</td>
</tr>
<tr>
<td>LOAN</td>
<td>202</td>
</tr>
<tr>
<td>MONTH</td>
<td>214</td>
</tr>
<tr>
<td>MORTGAGE</td>
<td>203</td>
</tr>
<tr>
<td>RATE</td>
<td>212</td>
</tr>
<tr>
<td>REGULAR</td>
<td>223</td>
</tr>
<tr>
<td>SAVINGS</td>
<td>207</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>224</td>
</tr>
<tr>
<td>STATEMENT</td>
<td>231</td>
</tr>
<tr>
<td>STOCKS</td>
<td>225</td>
</tr>
<tr>
<td>TRANSFER</td>
<td>208</td>
</tr>
<tr>
<td>VARIABLE</td>
<td>211</td>
</tr>
<tr>
<td>WITHDRAWALS</td>
<td>226</td>
</tr>
<tr>
<td>YEAR</td>
<td>213</td>
</tr>
</tbody>
</table>

Commands:

<table>
<thead>
<tr>
<th>Utterance</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN</td>
<td>101</td>
</tr>
<tr>
<td>CANCEL</td>
<td>107</td>
</tr>
<tr>
<td>EXIT</td>
<td>105</td>
</tr>
<tr>
<td>HELP</td>
<td>106</td>
</tr>
<tr>
<td>QUIT</td>
<td>104</td>
</tr>
<tr>
<td>REPEAT</td>
<td>103</td>
</tr>
<tr>
<td>STOP</td>
<td>102</td>
</tr>
</tbody>
</table>

278
### Voice Mail:

<table>
<thead>
<tr>
<th>Utterance</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>327</td>
</tr>
<tr>
<td>ASSISTANT</td>
<td>325</td>
</tr>
<tr>
<td>ASTERISK</td>
<td>336</td>
</tr>
<tr>
<td>ATTRIBUTES</td>
<td>328</td>
</tr>
<tr>
<td>AUDIO NAME</td>
<td>346</td>
</tr>
<tr>
<td>CALL</td>
<td>315</td>
</tr>
<tr>
<td>COMMENT</td>
<td>344</td>
</tr>
<tr>
<td>COPY</td>
<td>313</td>
</tr>
<tr>
<td>DELETE</td>
<td>308</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>320</td>
</tr>
<tr>
<td>DISTRIBUTION</td>
<td>345</td>
</tr>
<tr>
<td>DIVERSION</td>
<td>324</td>
</tr>
<tr>
<td>EMAIL</td>
<td>340</td>
</tr>
<tr>
<td>ENVELOPE</td>
<td>322</td>
</tr>
<tr>
<td>EXPERT</td>
<td>329</td>
</tr>
<tr>
<td>FAX</td>
<td>339</td>
</tr>
<tr>
<td>FORWARD</td>
<td>304</td>
</tr>
<tr>
<td>FUTURE</td>
<td>332</td>
</tr>
<tr>
<td>GOODBYE</td>
<td>348</td>
</tr>
<tr>
<td>GREETING</td>
<td>302</td>
</tr>
<tr>
<td>HASH</td>
<td>337</td>
</tr>
<tr>
<td>HEADER</td>
<td>321</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>326</td>
</tr>
<tr>
<td>Utterance</td>
<td>Annotation</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>LISTEN</td>
<td>317</td>
</tr>
<tr>
<td>MENU</td>
<td>343</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>303</td>
</tr>
<tr>
<td>NOTIFICATION</td>
<td>333</td>
</tr>
<tr>
<td>OPERATOR</td>
<td>331</td>
</tr>
<tr>
<td>OUTGOING</td>
<td>319</td>
</tr>
<tr>
<td>PAGER</td>
<td>335</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>323</td>
</tr>
<tr>
<td>PAUSE</td>
<td>314</td>
</tr>
<tr>
<td>PERSONAL</td>
<td>316</td>
</tr>
<tr>
<td>PHONE</td>
<td>334</td>
</tr>
<tr>
<td>PLAY</td>
<td>306</td>
</tr>
<tr>
<td>POUND</td>
<td>338</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>318</td>
</tr>
<tr>
<td>RECORD</td>
<td>301</td>
</tr>
<tr>
<td>REPLAY</td>
<td>312</td>
</tr>
<tr>
<td>REVERSE</td>
<td>305</td>
</tr>
<tr>
<td>REVIEW</td>
<td>310</td>
</tr>
<tr>
<td>SAVE</td>
<td>307</td>
</tr>
<tr>
<td>SEND</td>
<td>309</td>
</tr>
<tr>
<td>SKIP</td>
<td>311</td>
</tr>
<tr>
<td>SPEAK</td>
<td>341</td>
</tr>
<tr>
<td>STANDARD</td>
<td>330</td>
</tr>
<tr>
<td>STAR</td>
<td>347</td>
</tr>
<tr>
<td>Utterance</td>
<td>Annotation</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>WHO FROM</td>
<td>342</td>
</tr>
</tbody>
</table>

Yes/No:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>0000</td>
</tr>
<tr>
<td>NOPE</td>
<td>0000</td>
</tr>
<tr>
<td>OK</td>
<td>0001</td>
</tr>
<tr>
<td>SURE</td>
<td>0001</td>
</tr>
<tr>
<td>YEAH</td>
<td>0001</td>
</tr>
<tr>
<td>YEP</td>
<td>0001</td>
</tr>
<tr>
<td>YES</td>
<td>0001</td>
</tr>
</tbody>
</table>
The purpose of the times grammar is to recognize different time phrases. In U.K. English, a time phrase takes the following format:

\[ \text{<times>} = \text{<times1>} | \text{<times2>} \]

Where \( \text{<times1>} = \text{<word1>}? \text{<hour>}? \text{<minute>}? \text{<word2>}? \text{<zone>}? \)

Where \( \text{<times2>} = \text{<word1>}? \text{<minute2>} \text{<hour2>}? \text{<zone>}? \)

Where \( \text{<minute1>} \) can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>O'Clock</td>
<td>1_times</td>
</tr>
<tr>
<td></td>
<td>&lt;minute&gt;</td>
</tr>
</tbody>
</table>

Where \( \text{<word1>} \) can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>approximately</td>
<td>1_times</td>
</tr>
<tr>
<td>around</td>
<td>1_times</td>
</tr>
<tr>
<td>about</td>
<td>1_times</td>
</tr>
<tr>
<td>roughly</td>
<td>1_times</td>
</tr>
</tbody>
</table>

\( \text{<hour2>} \) can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;hour&gt; O'Clock</td>
<td>1_times</td>
</tr>
<tr>
<td>noon</td>
<td>1_times</td>
</tr>
<tr>
<td>midnight</td>
<td>1_times</td>
</tr>
<tr>
<td>twelve</td>
<td>1_times noon</td>
</tr>
<tr>
<td></td>
<td>1_times midnight</td>
</tr>
</tbody>
</table>

Where \( \text{<word2>} \) can be any of the following:
<table>
<thead>
<tr>
<th>UTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>in:</td>
<td>1_times the:</td>
<td>1_times morning</td>
<td>1_times</td>
</tr>
<tr>
<td>in:</td>
<td>1_times the:</td>
<td>1_times evening</td>
<td>1_times</td>
</tr>
<tr>
<td>in:</td>
<td>1_times the:</td>
<td>1_times afternoon</td>
<td>1_times</td>
</tr>
<tr>
<td>at:</td>
<td>1_times night:</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>&quot;a.m.&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;p.m.&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where `<minute2> = <minute><word3a> | <word3b> <word3c>`

<word3a> can be any of the following:

<table>
<thead>
<tr>
<th>UTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>minutes</td>
<td>1_times till:</td>
<td>1_times</td>
</tr>
<tr>
<td>minutes</td>
<td>1_times to:</td>
<td>1_times</td>
</tr>
<tr>
<td>minutes</td>
<td>1_times of:</td>
<td>1_times</td>
</tr>
<tr>
<td>minutes</td>
<td>1_times past:</td>
<td>1_times</td>
</tr>
<tr>
<td>minutes</td>
<td>1_times after:</td>
<td>1_times</td>
</tr>
<tr>
<td>minutes</td>
<td>1_times before:</td>
<td>1_times</td>
</tr>
<tr>
<td>minutes</td>
<td>1_times until:</td>
<td>1_times</td>
</tr>
</tbody>
</table>

<word3b> can be any of the following:

<table>
<thead>
<tr>
<th>UTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>quarter</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>a:</td>
<td>1_times quarter:</td>
<td>1_times</td>
</tr>
<tr>
<td>half</td>
<td>1_times</td>
<td></td>
</tr>
</tbody>
</table>
<word3c> can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>till:</td>
<td>1_times</td>
</tr>
<tr>
<td>to:</td>
<td>1_times</td>
</tr>
<tr>
<td>of:</td>
<td>1_times</td>
</tr>
<tr>
<td>past:</td>
<td>1_times</td>
</tr>
<tr>
<td>after:</td>
<td>1_times</td>
</tr>
</tbody>
</table>

<zone> can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>eastern:</td>
<td>1_times</td>
<td>time:</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>eastern:</td>
<td>1_times</td>
<td>standard:</td>
<td>1_times</td>
<td>time:</td>
</tr>
<tr>
<td>central:</td>
<td>1_times</td>
<td>time:</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>central:</td>
<td>1_times</td>
<td>standard::</td>
<td>1_times</td>
<td>time:</td>
</tr>
<tr>
<td>rocky:</td>
<td>1_times</td>
<td>mountain:</td>
<td>1_times</td>
<td>time:</td>
</tr>
<tr>
<td>rocky:</td>
<td>1_times</td>
<td>mountain:</td>
<td>1_times</td>
<td>standard:</td>
</tr>
<tr>
<td>mountain:</td>
<td>1_times</td>
<td>time:</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>mountain:</td>
<td>1_times</td>
<td>standard:</td>
<td>1_times</td>
<td>time:</td>
</tr>
<tr>
<td>pacific:</td>
<td>1_times</td>
<td>time:</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>pacific:</td>
<td>1_times</td>
<td>standard:</td>
<td>1_times</td>
<td>time:</td>
</tr>
<tr>
<td>western:</td>
<td>1_times</td>
<td>time:</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>western:</td>
<td>1_times</td>
<td>standard:</td>
<td>1_times</td>
<td>time:</td>
</tr>
<tr>
<td>daylight</td>
<td>1_times saving</td>
<td>1_times time</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>--------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>daylight</td>
<td>1_times saving</td>
<td>1_times time</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>greenwich</td>
<td>1_times mean</td>
<td>1_times time</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>british</td>
<td>1_times summer</td>
<td>1_times time</td>
<td>1_times</td>
<td></td>
</tr>
<tr>
<td>daylight</td>
<td>1_times savings</td>
<td>1_times time</td>
<td>1_times</td>
<td></td>
</tr>
</tbody>
</table>

`<hour>` can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>1_times</td>
</tr>
<tr>
<td>two</td>
<td>1_times</td>
</tr>
<tr>
<td>three</td>
<td>1_times</td>
</tr>
<tr>
<td>four</td>
<td>1_times</td>
</tr>
<tr>
<td>five</td>
<td>1_times</td>
</tr>
<tr>
<td>six</td>
<td>1_times</td>
</tr>
<tr>
<td>seven</td>
<td>1_times</td>
</tr>
<tr>
<td>eight</td>
<td>1_times</td>
</tr>
<tr>
<td>nine</td>
<td>1_times</td>
</tr>
<tr>
<td>ten</td>
<td>1_times</td>
</tr>
<tr>
<td>eleven</td>
<td>1_times</td>
</tr>
<tr>
<td>twelve</td>
<td>1_times</td>
</tr>
</tbody>
</table>
Where `<minute>` = `<num1p> <num1a> | <num1a> | <num1b> <num2>`

Where `<num1p>` can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>oh</td>
<td>1_times</td>
</tr>
<tr>
<td>zero</td>
<td>1_times</td>
</tr>
<tr>
<td>one</td>
<td>1_times</td>
</tr>
<tr>
<td>two</td>
<td>1_times</td>
</tr>
<tr>
<td>three</td>
<td>1_times</td>
</tr>
<tr>
<td>four</td>
<td>1_times</td>
</tr>
<tr>
<td>five</td>
<td>1_times</td>
</tr>
</tbody>
</table>

Where `<num1a>` can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>oh</td>
<td>1_times</td>
</tr>
<tr>
<td>zero</td>
<td>1_times</td>
</tr>
<tr>
<td>one</td>
<td>1_times</td>
</tr>
<tr>
<td>two</td>
<td>1_times</td>
</tr>
<tr>
<td>three</td>
<td>1_times</td>
</tr>
<tr>
<td>four</td>
<td>1_times</td>
</tr>
<tr>
<td>five</td>
<td>1_times</td>
</tr>
<tr>
<td>six</td>
<td>1_times</td>
</tr>
<tr>
<td>seven</td>
<td>1_times</td>
</tr>
<tr>
<td>eight</td>
<td>1_times</td>
</tr>
<tr>
<td>nine</td>
<td>1_times</td>
</tr>
</tbody>
</table>
Where $\textbf{<num1b>}$ can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ten</td>
<td>1_times</td>
</tr>
<tr>
<td>eleven</td>
<td>1_times</td>
</tr>
<tr>
<td>twelve</td>
<td>1_times</td>
</tr>
<tr>
<td>thirteen</td>
<td>1_times</td>
</tr>
<tr>
<td>fourteen</td>
<td>1_times</td>
</tr>
<tr>
<td>fifteen</td>
<td>1_times</td>
</tr>
<tr>
<td>sixteen</td>
<td>1_times</td>
</tr>
<tr>
<td>seventeen</td>
<td>1_times</td>
</tr>
<tr>
<td>eighteen</td>
<td>1_times</td>
</tr>
<tr>
<td>nineteen</td>
<td>1_times</td>
</tr>
</tbody>
</table>

Where $\textbf{<num2b>} = \textbf{<num3>}$ and $\textbf{<num4>}$?

$\textbf{<num3>}$ can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>twenty</td>
<td>1_times</td>
</tr>
<tr>
<td>thirty</td>
<td>1_times</td>
</tr>
<tr>
<td>forty</td>
<td>1_times</td>
</tr>
<tr>
<td>fifty</td>
<td>1_times</td>
</tr>
</tbody>
</table>
<num4> can be any of the following:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>1_times</td>
</tr>
<tr>
<td>two</td>
<td>1_times</td>
</tr>
<tr>
<td>three</td>
<td>1_times</td>
</tr>
<tr>
<td>four</td>
<td>1_times</td>
</tr>
<tr>
<td>five</td>
<td>1_times</td>
</tr>
<tr>
<td>six</td>
<td>1_times</td>
</tr>
<tr>
<td>seven</td>
<td>1_times</td>
</tr>
<tr>
<td>eight</td>
<td>1_times</td>
</tr>
<tr>
<td>nine</td>
<td>1_times</td>
</tr>
</tbody>
</table>

When this grammar is used, a special parser is activated that produces, in addition to the recognized text, an easy to use parsed string.

When time utterances (in either format) are recognized, they are parsed returned to the application as an annotated string. For more information on the grammar parsing process, see Grammar Parser.
Voice Activated Network Control - U.K. English

The purpose of the voice activated network control grammar is to recognize the command words used to control network features provided by telephone companies. Also included with this grammar are words common to personal vocabularies. The following words and phrases are supported in U.K. English:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add name</td>
<td>40</td>
</tr>
<tr>
<td>Call screening</td>
<td>26</td>
</tr>
<tr>
<td>Cancel call forwarding</td>
<td>25</td>
</tr>
<tr>
<td>Cancel call waiting</td>
<td>24</td>
</tr>
<tr>
<td>Call forwarding</td>
<td>20</td>
</tr>
<tr>
<td>Call return</td>
<td>21</td>
</tr>
<tr>
<td>Call trace</td>
<td>30</td>
</tr>
<tr>
<td>Delete name</td>
<td>41</td>
</tr>
<tr>
<td>Disable anonymous caller rejection</td>
<td>29</td>
</tr>
<tr>
<td>Enable anonymous caller rejection</td>
<td>28</td>
</tr>
<tr>
<td>Name that number</td>
<td>22</td>
</tr>
<tr>
<td>Per call blocking</td>
<td>27</td>
</tr>
<tr>
<td>Repeat call</td>
<td>23</td>
</tr>
<tr>
<td>Review list</td>
<td>42</td>
</tr>
</tbody>
</table>
The purpose of the voice mail sample grammar is to recognize words or phrases which are commonly found in voice mail applications. In U.K. English, the following words are supported for voice mail:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>327</td>
</tr>
<tr>
<td>Assistant</td>
<td>325</td>
</tr>
<tr>
<td>Asterisk</td>
<td>336</td>
</tr>
<tr>
<td>Attributes</td>
<td>328</td>
</tr>
<tr>
<td>Audio name</td>
<td>346</td>
</tr>
<tr>
<td>Call</td>
<td>315</td>
</tr>
<tr>
<td>Comment</td>
<td>344</td>
</tr>
<tr>
<td>Copy</td>
<td>313</td>
</tr>
<tr>
<td>Delete</td>
<td>308</td>
</tr>
<tr>
<td>Disconnect</td>
<td>320</td>
</tr>
<tr>
<td>Distribution</td>
<td>345</td>
</tr>
<tr>
<td>Diversion</td>
<td>324</td>
</tr>
<tr>
<td>E-mail</td>
<td>340</td>
</tr>
<tr>
<td>Envelope</td>
<td>322</td>
</tr>
<tr>
<td>Expert</td>
<td>329</td>
</tr>
<tr>
<td>Fax</td>
<td>339</td>
</tr>
<tr>
<td>Forward</td>
<td>304</td>
</tr>
<tr>
<td>Future</td>
<td>332</td>
</tr>
<tr>
<td>Goodbye</td>
<td>348</td>
</tr>
<tr>
<td>Greeting</td>
<td>302</td>
</tr>
<tr>
<td>Hash</td>
<td>337</td>
</tr>
<tr>
<td>Header</td>
<td>321</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>Language</td>
<td>326</td>
</tr>
<tr>
<td>Listen</td>
<td>317</td>
</tr>
<tr>
<td>Menu</td>
<td>343</td>
</tr>
<tr>
<td>Message</td>
<td>303</td>
</tr>
<tr>
<td>Notification</td>
<td>333</td>
</tr>
<tr>
<td>Operator</td>
<td>331</td>
</tr>
<tr>
<td>Outgoing</td>
<td>319</td>
</tr>
<tr>
<td>Pager</td>
<td>335</td>
</tr>
<tr>
<td>Password</td>
<td>323</td>
</tr>
<tr>
<td>Pause</td>
<td>314</td>
</tr>
<tr>
<td>Personal</td>
<td>316</td>
</tr>
<tr>
<td>Phone</td>
<td>334</td>
</tr>
<tr>
<td>Play</td>
<td>306</td>
</tr>
<tr>
<td>Pound</td>
<td>338</td>
</tr>
<tr>
<td>Receive</td>
<td>318</td>
</tr>
<tr>
<td>Record</td>
<td>301</td>
</tr>
<tr>
<td>Replay</td>
<td>312</td>
</tr>
<tr>
<td>Reverse</td>
<td>305</td>
</tr>
<tr>
<td>Review</td>
<td>310</td>
</tr>
<tr>
<td>Save</td>
<td>307</td>
</tr>
<tr>
<td>Send</td>
<td>309</td>
</tr>
<tr>
<td>Skip</td>
<td>311</td>
</tr>
<tr>
<td>Speak</td>
<td>341</td>
</tr>
<tr>
<td>Standard</td>
<td>330</td>
</tr>
<tr>
<td>Star</td>
<td>347</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>Who</td>
<td>342</td>
</tr>
</tbody>
</table>
Yes/No - U.K. English

The purpose of the yes/no grammar is to recognize words or phrases which mean yes or no. The following words are supported in U.K. English:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>0000</td>
</tr>
<tr>
<td>NOPE</td>
<td>0000</td>
</tr>
<tr>
<td>OK</td>
<td>0001</td>
</tr>
<tr>
<td>SURE</td>
<td>0001</td>
</tr>
<tr>
<td>YEAH</td>
<td>0001</td>
</tr>
<tr>
<td>YEP</td>
<td>0001</td>
</tr>
<tr>
<td>YES</td>
<td>0001</td>
</tr>
</tbody>
</table>
Sample Grammar Vocabularies - French

This section describes the words and phrases that are supported in the sample grammars for French that are included with the ViaVoice Telephony Tools. The following sample grammars are provided:

- Banking
- Commands
- Voice Activated Network Control
- Voice Mail
- Yes/No
Banking - French

The purpose of the banking sample grammar is to recognize words or phrases which are commonly found in banking applications.

In French, the following words and phrases are supported:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>année</td>
<td>213</td>
</tr>
<tr>
<td>auto</td>
<td>218</td>
</tr>
<tr>
<td>automobile</td>
<td>219</td>
</tr>
<tr>
<td>carte de crédit</td>
<td>201</td>
</tr>
<tr>
<td>CD</td>
<td>221</td>
</tr>
<tr>
<td>certificat</td>
<td>217</td>
</tr>
<tr>
<td>certificat</td>
<td>217</td>
</tr>
<tr>
<td>de dépôt</td>
<td>222</td>
</tr>
<tr>
<td>courant</td>
<td>206</td>
</tr>
<tr>
<td>crédit</td>
<td>229</td>
</tr>
<tr>
<td>dépôt</td>
<td>204</td>
</tr>
<tr>
<td>dividendes</td>
<td>227</td>
</tr>
<tr>
<td>épargne</td>
<td>207</td>
</tr>
<tr>
<td>fixe</td>
<td>210</td>
</tr>
<tr>
<td>hypothèque</td>
<td>203</td>
</tr>
<tr>
<td>informations</td>
<td>216</td>
</tr>
<tr>
<td>intérêts</td>
<td>228</td>
</tr>
<tr>
<td>investissement</td>
<td>230</td>
</tr>
<tr>
<td>jour</td>
<td>215</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>mois</td>
<td>214</td>
</tr>
<tr>
<td>normal</td>
<td>223</td>
</tr>
<tr>
<td>numéro de compte</td>
<td>205</td>
</tr>
<tr>
<td>prêt</td>
<td>202</td>
</tr>
<tr>
<td>relève</td>
<td>231</td>
</tr>
<tr>
<td>retraits</td>
<td>226</td>
</tr>
<tr>
<td>solde</td>
<td>209</td>
</tr>
<tr>
<td>spécial</td>
<td>224</td>
</tr>
<tr>
<td>taux</td>
<td>212</td>
</tr>
<tr>
<td>titres</td>
<td>225</td>
</tr>
<tr>
<td>transfert</td>
<td>208</td>
</tr>
<tr>
<td>variable</td>
<td>211</td>
</tr>
<tr>
<td>voiture</td>
<td>220</td>
</tr>
</tbody>
</table>
The commands grammar recognizes commonly used command words. The following commands are supported in French:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>abandon</td>
<td>104</td>
</tr>
<tr>
<td>aide</td>
<td>106</td>
</tr>
<tr>
<td>annulation</td>
<td>107</td>
</tr>
<tr>
<td>arrêt</td>
<td>102</td>
</tr>
<tr>
<td>début</td>
<td>101</td>
</tr>
<tr>
<td>répétition</td>
<td>103</td>
</tr>
<tr>
<td>sortie</td>
<td>105</td>
</tr>
</tbody>
</table>
Voice Activated Network Control - French

The purpose of the voice activated network control grammar is to recognize the command words used to control network features provided by the telephone companies. Also included with this grammar are words common to personal vocabularies.

The following words and phrases are supported in French:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>activation</td>
<td>28</td>
</tr>
<tr>
<td>ajout nom</td>
<td>40</td>
</tr>
<tr>
<td>annulation renvoi automatique</td>
<td>25</td>
</tr>
<tr>
<td>annulation signal d'appel</td>
<td>24</td>
</tr>
<tr>
<td>blocage sur appel</td>
<td>27</td>
</tr>
<tr>
<td>désactivation rejet correspondants anonymes</td>
<td>29</td>
</tr>
<tr>
<td>filtrage appels</td>
<td>26</td>
</tr>
<tr>
<td>indication du numéro</td>
<td>22</td>
</tr>
<tr>
<td>rappel automatique</td>
<td>21</td>
</tr>
<tr>
<td>répétition appel</td>
<td>23</td>
</tr>
<tr>
<td>revoir automatique d'appel</td>
<td>20</td>
</tr>
<tr>
<td>suppression nom</td>
<td>41</td>
</tr>
<tr>
<td>trace appels</td>
<td>30</td>
</tr>
<tr>
<td>visualisation liste</td>
<td>42</td>
</tr>
</tbody>
</table>
Voice Mail - French

The purpose of the voice mail sample grammar is to recognize words or phrases which are commonly found in voice mail applications. In French, the following words are supported for voice mail:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>adresse</td>
<td>327</td>
</tr>
<tr>
<td>appel</td>
<td>315</td>
</tr>
<tr>
<td>assistant</td>
<td>325</td>
</tr>
<tr>
<td>astérisque</td>
<td>336</td>
</tr>
<tr>
<td>attributs</td>
<td>328</td>
</tr>
<tr>
<td>au revoir</td>
<td>348</td>
</tr>
<tr>
<td>audio</td>
<td>346</td>
</tr>
<tr>
<td>bienvenue</td>
<td>302</td>
</tr>
<tr>
<td>commentaire</td>
<td>344</td>
</tr>
<tr>
<td>conversation</td>
<td>341</td>
</tr>
<tr>
<td>copie</td>
<td>313</td>
</tr>
<tr>
<td>déconnexion</td>
<td>320</td>
</tr>
<tr>
<td>dièse</td>
<td>338</td>
</tr>
<tr>
<td>diffusion</td>
<td>345</td>
</tr>
<tr>
<td>écoute</td>
<td>317</td>
</tr>
<tr>
<td>email</td>
<td>340</td>
</tr>
<tr>
<td>enregistrement</td>
<td>301</td>
</tr>
<tr>
<td>en-tête</td>
<td>321</td>
</tr>
<tr>
<td>enveloppe</td>
<td>322</td>
</tr>
<tr>
<td>envoi</td>
<td>309</td>
</tr>
<tr>
<td>étoile</td>
<td>347</td>
</tr>
<tr>
<td>futur</td>
<td>332</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>langue</td>
<td>326</td>
</tr>
<tr>
<td>lecture</td>
<td>306</td>
</tr>
<tr>
<td>menu</td>
<td>343</td>
</tr>
<tr>
<td>message</td>
<td>303</td>
</tr>
<tr>
<td>mot de passe</td>
<td>323</td>
</tr>
<tr>
<td>normal</td>
<td>330</td>
</tr>
<tr>
<td>notification</td>
<td>333</td>
</tr>
<tr>
<td>opérateur</td>
<td>331</td>
</tr>
<tr>
<td>pager</td>
<td>335</td>
</tr>
<tr>
<td>parasites</td>
<td>337</td>
</tr>
<tr>
<td>pause</td>
<td>314</td>
</tr>
<tr>
<td>PCV</td>
<td>305</td>
</tr>
<tr>
<td>personnel</td>
<td>316</td>
</tr>
<tr>
<td>qui</td>
<td>342</td>
</tr>
<tr>
<td>réception</td>
<td>318</td>
</tr>
<tr>
<td>relecture</td>
<td>312</td>
</tr>
<tr>
<td>renvoi</td>
<td>304</td>
</tr>
<tr>
<td>saut</td>
<td>311</td>
</tr>
<tr>
<td>sauvegarde</td>
<td>307</td>
</tr>
<tr>
<td>sortant</td>
<td>319</td>
</tr>
<tr>
<td>spécialiste</td>
<td>329</td>
</tr>
<tr>
<td>suppression</td>
<td>308</td>
</tr>
<tr>
<td>télécopie</td>
<td>339</td>
</tr>
<tr>
<td>téléphone</td>
<td>334</td>
</tr>
<tr>
<td>transfert</td>
<td>324</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>visualisation</td>
<td>310</td>
</tr>
</tbody>
</table>
**Yes/No - French**

The purpose of the yes/no grammar is to recognize words or phrases which mean yes or no. In French, the following words are supported:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>d'accord</td>
<td>0001</td>
</tr>
<tr>
<td>OK</td>
<td>0001</td>
</tr>
<tr>
<td>ouais</td>
<td>0001</td>
</tr>
<tr>
<td>nan</td>
<td>0000</td>
</tr>
<tr>
<td>non</td>
<td>0000</td>
</tr>
</tbody>
</table>
Sample Grammar Vocabularies - German

This section describes the words and phrases that are supported in the sample grammars for German that are included with the ViaVoice Telephony Tools. The following sample grammars are provided:

- Banking
- Commands
- Dates
- Digits and digComp
- Spelling
- Times
- Voice Activated Network Control
- Voice Mail
- Yes/No
The purpose of the banking sample grammar is to recognize words or phrases which are commonly found in banking applications.

In German, the following words and phrases are supported:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABHEBUNG</td>
<td>226</td>
</tr>
<tr>
<td>AKTIEN</td>
<td>225</td>
</tr>
<tr>
<td>ANZAHLUNG</td>
<td>204</td>
</tr>
<tr>
<td>AUTO</td>
<td>219</td>
</tr>
<tr>
<td>BESTÄTIGUNG</td>
<td>217</td>
</tr>
<tr>
<td>BILANZ</td>
<td>209</td>
</tr>
<tr>
<td>CD</td>
<td>221</td>
</tr>
<tr>
<td>DARLEHEN</td>
<td>202</td>
</tr>
<tr>
<td>DIVIDENDE</td>
<td>227</td>
</tr>
<tr>
<td>EINLAGE</td>
<td>204</td>
</tr>
<tr>
<td>ERSPARNISSE</td>
<td>207</td>
</tr>
<tr>
<td>FIXIERT</td>
<td>210</td>
</tr>
<tr>
<td>GUTHABEN</td>
<td>204</td>
</tr>
<tr>
<td>HYPOTHEK</td>
<td>202</td>
</tr>
<tr>
<td>INFORMATION</td>
<td>216</td>
</tr>
<tr>
<td>INVESTITION</td>
<td>230</td>
</tr>
<tr>
<td>JAHR</td>
<td>213</td>
</tr>
<tr>
<td>KONTOAUSZUG</td>
<td>231</td>
</tr>
<tr>
<td>KONTOBESTÄTIGUNG</td>
<td>217</td>
</tr>
<tr>
<td>KONTONUMMER</td>
<td>205</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>KREDITKARTE</td>
<td>201</td>
</tr>
<tr>
<td>KUNDENKREDITKARTE</td>
<td>201</td>
</tr>
<tr>
<td>KURS</td>
<td>212</td>
</tr>
<tr>
<td>MONAT</td>
<td>214</td>
</tr>
<tr>
<td>RECHNUNG</td>
<td>206</td>
</tr>
<tr>
<td>SPEZIELL</td>
<td>224</td>
</tr>
<tr>
<td>STAMMKUNDE</td>
<td>223</td>
</tr>
<tr>
<td>TAG:</td>
<td>215</td>
</tr>
<tr>
<td>ÜBERWEISEN</td>
<td>208</td>
</tr>
<tr>
<td>VERÄNDERLICH</td>
<td>211</td>
</tr>
<tr>
<td>WAGEN</td>
<td>218</td>
</tr>
<tr>
<td>ZINSEN</td>
<td>228</td>
</tr>
</tbody>
</table>
Commands - German

The commands grammar recognizes commonly used command words. The following commands are supported in German:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABBRECHEN</td>
<td>105</td>
</tr>
<tr>
<td>ABBRUCH</td>
<td>107</td>
</tr>
<tr>
<td>BEENDEN</td>
<td>102</td>
</tr>
<tr>
<td>BEGINNEN</td>
<td>101</td>
</tr>
<tr>
<td>HILFE</td>
<td>106</td>
</tr>
<tr>
<td>STARTEN</td>
<td>101</td>
</tr>
<tr>
<td>STOP</td>
<td>102</td>
</tr>
<tr>
<td>VERLASSEN</td>
<td>104</td>
</tr>
<tr>
<td>WIEDERHOLEN</td>
<td>103</td>
</tr>
</tbody>
</table>
Dates - German

The purpose of the dates grammar is to recognize different date phrases. A date phrase takes the following format:
DAY ["der"] DAY_NUMBER MONTH [[CENTURY] YEAR]

Where:

**DAY** can be any day name: "Sonntag" through "Samstag"

**DAY_NUMBER** can be any number: "erste" - "einunddreißigste" or "erster" - "einunddreißigster"

**MONTH** can be any month name: "Januar" through "Dezember"

**CENTURY** can be one of the following: "neunzehn", "neunzehnhundert", "zweitausend" or "zweitausend und"

**YEAR** can be one a number from "eins" through "neunundneunzig"

Items in square brackets ([[]]) are optional.

When this grammar is used, a special parser is activated that produces, in addition to the recognized text, an easy to use parsed string.

When date utterances are recognized, they are parsed and returned to the application as an annotated string. For more information on the parsing process, see Grammar Parser.
Digis and digComp Grammars - German

The digis grammars are designed to provide recognition of digit strings of varied length. These include telephone numbers, credit card numbers, personal identification numbers, and other digit sequences. Dollar amounts and natural numbers are not included in this grammar.

In all of these grammars, please note that [DIGIT] can be any number between "NULL" and "NEUN". Any number can occur in any position. No grouped teens, hundreds, or thousands are allowed in these grammars.

The digis grammars are designed to be used by applications that recognize some numbers but primarily recognize text. The digComp grammars are designed for use by digit-only applications and provide precision recognition of digits. The number of grammars, names of the grammars, and annotations returned are the same for digis and digComp. The digis grammars reside in the <VTT_HOME>\app\Gr_GR\digit directory and the digComp grammars reside in the <VTT_HOME>\app\Gr_GR\digcomp directory. The digComp applet has a baseforms.special file that must be used during an application build.

Use of the digis grammars versus the digComp grammars is determined in part by the value of the taskId resource. Ensure that the resource is set properly for the grammars you want to use. A mis-match in grammar selection and resource value may cause system performance degradation.
Spelling Mode - German

The spelling mode grammars allow the user to spell the words (and phrases) that are to be recognized. Such a mode is particularly important when the user doesn't know how to pronounce the word (or phrase) correctly.

The ViaVoice Telephony Tools contain spelling grammars for all of the sample grammars. The following are examples for the German Language:

<table>
<thead>
<tr>
<th>Utterance</th>
<th>Recognized Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adresse</td>
<td>adresse</td>
</tr>
<tr>
<td>Assistant</td>
<td>assistent</td>
</tr>
<tr>
<td>Stern</td>
<td>stern</td>
</tr>
</tbody>
</table>
**Times - German**

The purpose of the times grammar is to recognize different time phrases. In German, a time phrase takes the following format:

\[ [\text{APPROX}] \ \text{HOUR1} \ ["\text{Uhr}"] \ ["\text{und}"] \ [\text{MINUTE1} \ "\text{Minuten}"] \ [\text{ZONE}] \]

or

\[ [\text{APPROX}] \ \text{MINUTE2} \ \text{HOUR2} \ [\text{ZONE}] \]

Where:

- **APPROX** must be one of the following: "beinahe" or "ungefähr"
- **HOUR1** must be a number from "ein" through "dreiundzwanzig".
- **MINUTE1** must be a number from "eine" through "neunundfünfzig".
- **ZONE** must be one of the following: "Sommerzeit", "Winterzeit", or "mitteleuropäische Zeit".
- **MINUTE2** must be a number from "eine" through "neunundfünfzig" followed by "Minuten 'bis' or 'nach' or 'vor'"
- **HOUR2** must be one of the following: "Mittag", "Mitternacht", "null Uhr" or "zwölf Uhr".

Items in square brackets ([[]]) are optional.

When this grammar is used, a special parser is activated that produces, in addition to the recognized text, an easy to use parsed string.

When time utterances (in either format) are recognized, they are parsed returned to the application as an annotated string. For more information on the grammar parsing process, see [Grammar Parser](#).
Voice Activated Network Control - German

The purpose of the voice activated network control grammar is to recognize the command words used to control network features provided by the telephone companies. Also included with this grammar are words common to personal vocabularies.

The following words and phrases are supported in German:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANRUF WIEDERHOLEN</td>
<td>23</td>
</tr>
<tr>
<td>ANRUF ZURÜCKLEITEN</td>
<td>21</td>
</tr>
<tr>
<td>LISTE PRÜFEN</td>
<td>42</td>
</tr>
<tr>
<td>NAME HINZUFÜGEN</td>
<td>40</td>
</tr>
<tr>
<td>NAME LÖSCHEN</td>
<td>41</td>
</tr>
<tr>
<td>TRACEMODUS AKTIVIEREN</td>
<td>30</td>
</tr>
<tr>
<td>ÜBERPRÜFUNG AKTIVIEREN</td>
<td>26</td>
</tr>
<tr>
<td>UMLEITUNG AKTIVIEREN</td>
<td>20</td>
</tr>
<tr>
<td>UMLEITUNG BEENDEN</td>
<td>25</td>
</tr>
<tr>
<td>WARTEMODUS BEENDEN</td>
<td>24</td>
</tr>
</tbody>
</table>
Voice Mail - German

The purpose of the voice mail sample grammar is to recognize words or phrases which are commonly found in voice mail applications. In German, the following words are supported for voice mail:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSPIELEN</td>
<td>306</td>
</tr>
<tr>
<td>ADRESSE</td>
<td>327</td>
</tr>
<tr>
<td>ANRUF</td>
<td>315</td>
</tr>
<tr>
<td>ASSISTENT</td>
<td>325</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>328</td>
</tr>
<tr>
<td>AUDIO</td>
<td>346</td>
</tr>
<tr>
<td>AUF WIEDERSEHEN</td>
<td>348</td>
</tr>
<tr>
<td>AUFZEICHNEN</td>
<td>301</td>
</tr>
<tr>
<td>AUSGEHEND</td>
<td>319</td>
</tr>
<tr>
<td>BENACHRICHTIGUNG</td>
<td>333</td>
</tr>
<tr>
<td>BRIEFKOPF</td>
<td>321</td>
</tr>
<tr>
<td>E-MAIL</td>
<td>340</td>
</tr>
<tr>
<td>EMPFANGEN</td>
<td>318</td>
</tr>
<tr>
<td>EXPERTE</td>
<td>329</td>
</tr>
<tr>
<td>FAX</td>
<td>339</td>
</tr>
<tr>
<td>GRUSS</td>
<td>302</td>
</tr>
<tr>
<td>HÖREN</td>
<td>317</td>
</tr>
<tr>
<td>KOMMENTAR</td>
<td>344</td>
</tr>
<tr>
<td>KOPIEREN</td>
<td>313</td>
</tr>
<tr>
<td>LÖSCHEN</td>
<td>308</td>
</tr>
<tr>
<td>MENÜ</td>
<td>343</td>
</tr>
<tr>
<td>NACHRICHT</td>
<td>303</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>NUMMERZEICHEN</td>
<td>337</td>
</tr>
<tr>
<td>NUMMERZEICHEN</td>
<td>338</td>
</tr>
<tr>
<td>PAGER</td>
<td>335</td>
</tr>
<tr>
<td>PASSWORT</td>
<td>323</td>
</tr>
<tr>
<td>PAUSE</td>
<td>314</td>
</tr>
<tr>
<td>PERSÖNLICH</td>
<td>316</td>
</tr>
<tr>
<td>PRÜFEN</td>
<td>310</td>
</tr>
<tr>
<td>SENDEN</td>
<td>309</td>
</tr>
<tr>
<td>SPEICHERN</td>
<td>307</td>
</tr>
<tr>
<td>SPRACHE</td>
<td>326</td>
</tr>
<tr>
<td>SPRECHEN</td>
<td>341</td>
</tr>
<tr>
<td>STANDARD</td>
<td>330</td>
</tr>
<tr>
<td>STERN</td>
<td>336</td>
</tr>
<tr>
<td>STERN</td>
<td>347</td>
</tr>
<tr>
<td>TELEFON</td>
<td>334</td>
</tr>
<tr>
<td>TRENEN</td>
<td>320</td>
</tr>
<tr>
<td>ÜBERSPRINGEN</td>
<td>311</td>
</tr>
<tr>
<td>UMKEHREN</td>
<td>305</td>
</tr>
<tr>
<td>UMLEITUNG</td>
<td>324</td>
</tr>
<tr>
<td>UMSCHLAG</td>
<td>322</td>
</tr>
<tr>
<td>VERMITTLUNG</td>
<td>331</td>
</tr>
<tr>
<td>VERTEILER</td>
<td>345</td>
</tr>
<tr>
<td>VERTEILUNG</td>
<td>345</td>
</tr>
<tr>
<td>WER</td>
<td>342</td>
</tr>
<tr>
<td>WIEDERGEBEN</td>
<td>312</td>
</tr>
<tr>
<td>UTTERANCE</td>
<td>ANNOTATION</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>WEITERLEITEN</td>
<td>304</td>
</tr>
<tr>
<td>ZUKUNFT</td>
<td>332</td>
</tr>
</tbody>
</table>
Yes/No - German

The purpose of the yes/no grammar is to recognize words or phrases which mean yes or no. In German, the following words are supported:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>ANNOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA</td>
<td>1</td>
</tr>
<tr>
<td>NE</td>
<td>0</td>
</tr>
<tr>
<td>NEIN</td>
<td>0</td>
</tr>
<tr>
<td>NOE</td>
<td>0</td>
</tr>
<tr>
<td>OK</td>
<td>1</td>
</tr>
<tr>
<td>SICHER</td>
<td>1</td>
</tr>
</tbody>
</table>
Grammar Parser

The grammar parser is used to decode the recognized text into a simple string that can be used by the application. The application is given both the recognized text and the new "parsed string."

Currently the parser is used with the following sample grammars:

- Dates
- Times (both formats)

The parser takes the recognized string and generates a new string (the parsed string) which contains special fields. These fields represent the recognized string in a compact way that then can be used by the application programmer. The last field of the parsed string (called parsed_phrase field) contains the recognized text in a fix format.

The parsed string is attached to the recognized text as the annotation of this text. The different fields of the parsed string are separated by the "#" character.

The following sections describe the parser results for the dates and times grammars.

Dates

The resulting annotation string has the following format:

week_day#month#month_day#year#parsed_phrase

Where:

- month is the month as a number (January = 1)
- month_day is the day in the month (1 through 31)
- week_day is the day of the week as a number (Sunday = 1)
- year is the year.

Some examples of how dates are parsed:

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>RESULTING ANNOTATION STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday March the thirtieth two thousand</td>
<td>2#3#30#2000#Monday_March_30_2000</td>
</tr>
<tr>
<td>February twenty first</td>
<td>-1#2#21#-1#February_21</td>
</tr>
<tr>
<td>October twentieth oh three</td>
<td>-1#10#20#2003#October_20_2003</td>
</tr>
<tr>
<td>Sunday April the first</td>
<td>1#4#1#-1#Sunday_April_1</td>
</tr>
</tbody>
</table>
**Times**

For the times grammar, the parser returns an annotation string in the following format:

\[ \text{approx}\#\text{hour}\#\text{minute}\#\text{is\_am}\#\text{zone}\#\text{parsed\_phrase} \]

Where:

- **approx** is set to 1 if the utterance contains a word from APPROX
- **hour** is the hour
- **minute** is the minute past the hour
- **is\_am** is set to 1 if the time is A.M.
- **zone** is the index to the zone-table and can be:
  - 0 = "eastern time" "eastern standard time"
  - 1 = "central time" "central standard time"
  - 2 = "rocky mountain time" "rocky mountain standard time"
  - 3 = "mountain time" mountain standard time"
  - 4 = "pacific time," "pacific standard time"
  - 5 = "western time" "western standard time"
  - 6 = "daylight time" "daylight saving time" "daylight savings time"

<table>
<thead>
<tr>
<th>UTTERANCE</th>
<th>RESULTING ANNOTATION STRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>seventeen minutes before ten</td>
<td>-1#9#43#-1#-1#9:43</td>
</tr>
<tr>
<td>about half past eight in the evening</td>
<td>1#8#30#0#-1#about_8:30_P.M.</td>
</tr>
<tr>
<td>a quarter til four A.M. central time</td>
<td>-1#3#45#1#1#3:45_A.M._central_standard_time</td>
</tr>
</tbody>
</table>
The Utilities

There are several commands and utilities provided with the ViaVoice Telephony Tools that can help you develop and test your phone recognition applications. These tools are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DialWatch</strong></td>
<td>Displays the current status of a call line and the output of the Logger and StatLog processes.</td>
</tr>
<tr>
<td><strong>listres</strong></td>
<td>Lists the current values of the specified resources in VTTDefaults.</td>
</tr>
<tr>
<td><strong>logtail</strong></td>
<td>A command-line utility for monitoring system events and errors.</td>
</tr>
<tr>
<td><strong>setdbg</strong></td>
<td>Sets the debug level for the specified processes.</td>
</tr>
<tr>
<td><strong>text2pcm</strong></td>
<td>Converts text into speech output.</td>
</tr>
<tr>
<td><strong>tsmcon</strong></td>
<td>Controls the tsmRouter on a host machine.</td>
</tr>
</tbody>
</table>

**DialWatch**

The DialWatch utility displays the current status of the call line. It also displays selected output of the logger and statlog processes. Tsm event tags from the ernestine library are sent to statlog. If the event has a single underscore, it is stripped away and displayed in Dialwatch. If the event has a double underscore, it is stripped away and sent to a file.
There are four types of status displayed by **DialWatch**, each in a dedicated window:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Status</td>
<td>The events currently being handled on the call line.</td>
</tr>
<tr>
<td>Error Messages</td>
<td>Any error messages reported by the logger.</td>
</tr>
<tr>
<td>Statistics</td>
<td>Reports all events launched or stopped by the system.</td>
</tr>
<tr>
<td>Informational Messages</td>
<td>All messages reported by the statlog process.</td>
</tr>
</tbody>
</table>

**Channel Status Window**

The Channel Status Window contains 6 possible columns. Each column head can be dragged to increase the width of the respective column. The columns are:

- **Channel** - Gives the respective active channel number. In the Dialwatch diagram, only one channel is currently active (Channel 6).

- **Host** - Provides the server name hosting the session. In the Dialwatch example, the server named Henry is the server hosting the Channel 6 session.

![Channel Status Window](image)

[Table of Channel Status]

<table>
<thead>
<tr>
<th>Channel</th>
<th>Host</th>
<th>Call</th>
<th>Time</th>
<th>Start</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>HENRY</td>
<td>0</td>
<td>12:56:46</td>
<td>12:55:32</td>
<td>DONE</td>
</tr>
</tbody>
</table>

[Diagram of Channel Status Window]
Call - The number of calls handled on the respective channel since the process was initiated.

Time - Gives the time that the event completed. See the Events Table for a listing of possible Events.

Start - Gives the time that the event started. See the Events Table for a listing of possible Events.

Event - An action started or completed by the system. Here is a table of possible events:

### DIALWATCH EVENTS TABLE

<table>
<thead>
<tr>
<th>Event Abbreviation</th>
<th>Definition</th>
<th>Event Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_UTTCMPNC</td>
<td>Utterance complete no confirmation needed</td>
<td>The utterance is complete. No confirmation is required.</td>
</tr>
<tr>
<td>_UTTCMP</td>
<td>Utterance complete</td>
<td>The utterance is complete.</td>
</tr>
<tr>
<td>_UTTCMPRJ</td>
<td>Utterance complete rejected</td>
<td>The utterance, although marked complete, has been rejected by the system.</td>
</tr>
<tr>
<td>_READY</td>
<td>Channel just started ready for requests</td>
<td>The telephony channel is ready to receive utterances.</td>
</tr>
<tr>
<td>_STARTAPP</td>
<td>TsmStartAppl was received</td>
<td>TsmStartAppl has been received by the system.</td>
</tr>
<tr>
<td>_ENDAPP</td>
<td>TsmEndAppl was received</td>
<td>TsmEndAppl has been received by the system.</td>
</tr>
<tr>
<td>_LDCTX</td>
<td>TsmLoadContext</td>
<td>TsmLoadContext has been received by the system.</td>
</tr>
<tr>
<td>_UNLDCTX</td>
<td>TsmUnloadContext</td>
<td>TsmUnloadContext has been received by the system.</td>
</tr>
<tr>
<td>_ENAVOC</td>
<td>TsmEnableVocab</td>
<td>TsmEnableVocab has been received by the system.</td>
</tr>
<tr>
<td>_DISVOC</td>
<td>TsmDisableVocab</td>
<td>TsmDisableVocab has been received by the system.</td>
</tr>
<tr>
<td>_STARTREC</td>
<td>TsmStartReco</td>
<td>TsmStartReco has been received by the system.</td>
</tr>
<tr>
<td>_STOPREC</td>
<td>TsmStopReco</td>
<td>TsmStopReco has been received by the system.</td>
</tr>
<tr>
<td>_GETTEXT</td>
<td>TsmGetText</td>
<td>TsmGetText has been received by the system.</td>
</tr>
<tr>
<td>Event Abbreviation</td>
<td>Definition</td>
<td>Event Explanation</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><em>TRAIN</em></td>
<td>TsmTrain</td>
<td>TsmTrain has been received by the system.</td>
</tr>
<tr>
<td><em>SAVCTX</em></td>
<td>TsmSaveContext</td>
<td>TsmSaveContext has been received by the system.</td>
</tr>
<tr>
<td><em>DELWRD</em></td>
<td>TsmDeleteWord</td>
<td>TsmDeleteWord has been received by the system.</td>
</tr>
<tr>
<td><em>WRDDDATA</em></td>
<td>TsmWordData</td>
<td>TsmWordData has been received by the system.</td>
</tr>
<tr>
<td><em>TERMAPPL</em></td>
<td>TsmTermAppl</td>
<td>TsmTermAppl has been received by the system.</td>
</tr>
<tr>
<td><em>UNKEXIT</em></td>
<td>Unknown command received</td>
<td>An unknown command has been received by the system.</td>
</tr>
<tr>
<td><em>RECREJ</em></td>
<td>Reco rejected</td>
<td>The system received an utterance, but does not understand, therefore the utterance has been rejected. Check the Error Messages panel as well as the logger and statlog files for possible rejection reasons.</td>
</tr>
<tr>
<td><em>DONE</em></td>
<td>Acknowledgement</td>
<td>The system has successfully received and processed the utterance.</td>
</tr>
<tr>
<td><em>DTMF</em></td>
<td>Dual Tone Multi-frequency</td>
<td>The system has received dialing digits.</td>
</tr>
<tr>
<td><em>HANGUP</em></td>
<td>On-hook signal</td>
<td>The system has received an on-hook signal. The call has been terminated.</td>
</tr>
<tr>
<td><em>RECO</em></td>
<td>Speech reco</td>
<td>The system has successfully received utterances.</td>
</tr>
<tr>
<td><em>READY</em></td>
<td>Ready</td>
<td>The system is ready to receive and process events.</td>
</tr>
</tbody>
</table>

Additional events may be displayed, based upon the event tags defined by your tcl script.
Error Messages Window

The Error Messages window logs any error conditions encountered by the system. The recognized message levels for this window are:

1. Dbg = Debug
2. WRN = Warning
3. TRC = Trace
4. ADV = Advisory
5. ERROR = Error
6. FATAL = Fatal.

Statistics Window

The Statistics Window is used to log events, as well as the number of times an event was launched or stopped. In the previous Dialwatch diagram, the TsmDisableVocab command was received twice, and, the DONE event was issued 13 times (indicating that the 13 displayed processes were successfully completed).

Info Messages Window

The Info Messages Window displays internal informational messages from the speech engine. In the Dialwatch diagram, the Info Messages Window shows that a process started on 08-06 at 12:56:46 that used channel 6 and the VttDispatcher process. It also shows the addition and location of each memory block handle. In addition to messages that may be used for debugging (indicated by DBG), there are also messages used for general informational purposes (indicated by INF). These messages give the state of the engine at any given time.

DialWatch Parameters

Note that DialWatch can be run remotely. From the command line, specify:


Where:

--help - Displays a message box that describes the DialWatch parameters.

-h - Specifies the name of the host to monitor. The default is the name of the local host.

-l - Specifies the filename to which trace info from DialWatch is logged.

-p - Specifies the protocol to use when monitoring a remote host. On the local host, Windows messages
-f - Specifies the font size (in points) to use in the DialWatch window. The default font size is 12 points.

--u LL - Specifies the language of the DialWatch user interface. By default, the language is U.S. English. Specify this parameter whenever the language of the application is not U.S. English. LL can be one of the following:

- En_US for U.S. English
- En_UK for U.K. English
- Fr_FR for French
- Gr_GR for German

listres

The listres command displays the current values for the specified resources. The format of the listres command is:

```
listres [--help] [-v] [-s] [-g] [-r] process[[[[.id].class].objectid].name]
```

or

```
listres [-v] <(g|r)> <pattern>
```

where:

--help - Displays the listres help parameters.

-v – Lists the source of the resource value (file or defaults). Specifies that all resources obtained from VTTDefaults are listed.

-s – Lists resources matching the specified scope. Only the resources matching the given scope will be listed.

-g – Lists resources matching the <pattern> global expression.

-r – Lists resources matching the <pattern> regular expression.

**NOTE:** The listres command is a Tcl script. Before you can use listres for the first time, you need to generate the file pkgindex.tcl. To do this, type the following at the command line:

```
tclch
```

At the prompt, type:

```
exit
```

This generates the file, and you can now run the listres as described. This procedure only needs to be done the first time you use listres.
The following example lists the current values for all resources defined in the VTTDefaults files:

```
listres -s *.*.*
```

This example lists the current values for all of the Tsm speech recognition resources:

```
listres -v chp.*.Tsmapi
```

This example lists all resources defined for TelLine objects.

```
listres -g *TellLine*
```

This example lists resources defined for chp processes 0 and 1.

```
listres -r chp.[0-1]
```

### logtail

The **logtail** command displays the output of the logger and statlog processes. The format of the **logtail** command is:

```
logtail [--help] [-t] [-f filter] [-h hostname] [-r interval] [-p protocol] [-l (yes|no)] [-s (yes|no)]
```

where:

- **--help** - Displays a message box that describes the logtail parameters.
- **-t** - Prints the local time stamp.
- **-m** - Mutes (does not print) the preamble.
- **-l** - Prints logger output. The default is yes.
- **-s** - Prints statlog output. The default is no.
- **-f filter** - Specifies the scope of the command using process[.cloneId][.className]. Note that process can be specified as [hostname:]<name|ALL>[.cloneId].

- **-h hostname** - Specifies the host to monitor. The default is the local host.

- **-r interval** - Specifies the reconnect interval in milliseconds. The default is 5000 milliseconds.

- **-p protocol** - A string which specifies the protocol to use, which can be either msg, udp, or tcp. The default for local host is msg; the default for a remote host is tcp.

The following example monitors the call line activity for all of the telephony processes on the local host, displaying only statlog information:

```
logtail -l no -s yes
```

Note that **logtail** can be run remotely. To do so, simply use the -h parameter followed by the name of the remote host. For example:

```
logtail -h ernestine
```

Monitors the call line activity on the ernestine host.
**setdbg**

The `setdbg` utility sets the debug level for the specified processes on the specified host. The format of the `setdbg` command is:

```
setdbg [--help] [-s category] [-h hostname] [-l level]
```

Where:

`--help` - Displays a message box that describes the `setdbg` parameters.

`-s category` – Specifies the category for which to set the debug level. Specify category as `<process>.<cloneid>.<classname>.<resourcename>`.

`-h hostname` – Specifies the name of the host on which the processes are running.

`-l level` – Specifies the debug level to set. Supported values for the debug level are:

<table>
<thead>
<tr>
<th>Level</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Advise</td>
</tr>
<tr>
<td>d</td>
<td>Debug</td>
</tr>
<tr>
<td>e</td>
<td>Error</td>
</tr>
<tr>
<td>f</td>
<td>Fatal</td>
</tr>
<tr>
<td>t</td>
<td>Information</td>
</tr>
<tr>
<td>w</td>
<td>Warning</td>
</tr>
</tbody>
</table>

The following example sets all registered objects to their current debug level as defined in VTTDefaults:

```
setdbg -s *.*.*
```

This example turns trace on for all of the objects in process `chp.0`:

```
setdbg -s chp.0.* -l t
```

Finally, this example sets the debug level to warning for all semaphore objects in all processes:

```
setdbg -s *.*.* sema -l w
```

**tsmcon**

The `tsmcon` utility monitors the tsmRouter process on any host system. The format of the `tsmcon` utility is:

```
tsmcon [--help] [-h hostname] [-b channel | -f channel]
```

where:

`--help` - Displays a message box that describes the `tsmcon` parameters.
- **h hostname** - Specifies the host to monitor. If not specified, this defaults to the local host.

- **b channel** - Specifies the channels that should be busied out. To busy out all channels, specify "all" for the channel parameter.

- **f channel** - Specifies the busied channels that should be freed. To free all channels, specify "all" for the channel parameter.

The following example displays the current status for the host telephony1:

```
  tsmcon -h telephony1
```

Router: telephony1

<table>
<thead>
<tr>
<th>Entry</th>
<th>Status</th>
<th>Server</th>
<th>Count</th>
<th>Type</th>
<th>Language</th>
<th>Last_Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>1</td>
<td>17</td>
<td>Dialer</td>
<td>En_US</td>
<td>14:34:50.343</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>1</td>
<td>15</td>
<td>Dialer</td>
<td>En_US</td>
<td>14:33:43.718</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>1</td>
<td>17</td>
<td>Dialer</td>
<td>En_US</td>
<td>14:34:49.875</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>0</td>
<td>16</td>
<td>Dialer</td>
<td>En_US</td>
<td>14:34:50.375</td>
</tr>
</tbody>
</table>

The information that is displayed is:

- **Entry** - The channel number.
- **Status** - Describes the status of the engine connection from the client side. Client status can either be "A" (allocated) or "F" (free).
- **Server** - Describes the status of the engine connection from the server side. Server status can either be "0" (free) or "1" (allocated).
- **Count** - The number of times the engine was allocated.
- **Type** - The type of engine as defined in VTTDefaults.
- **Language** - The language of the engine.
- **Last_Allocated** - The time the engine was last allocated.

The following example busies out all of the channels on the ernestine server:

```
  tsmcon -h ernestine -b all
```

Similarly, the following command frees all of the channels on ernestine (the current server):

```
  tsmcon -f all
```

Issuing tsmcon with no parameters displays the status of the current server.
Text-to-Speech Conversion Utility

ViaVoice Telephony Tools provides mechanisms for conversion of text to speech output in Pulse Code Modulation (PCM) or .WAV format. Use the text2pcm utility or the Tts() class for conversion.

The text2pcm Utility

The text2pcm utility converts text into speech output using Pulse Code Modulation (PCM) or .WAV format. The text2pcm utility is useful in generating text and then playing it. Your application can also make use of it at run time (for example, to play an unrecorded prompt).

The utility has two command line formats:

(1) text2pcm -t "text" -w -p <output filename> -s <speed> -v <volume> -e .<extension> -q -l <Ll_LL>

or

(2) text2pcm -f <input filename> -w -s <speed> -v <volume> -e .<extension> -q -l <Ll_LL>

Use the first format to convert one line of text at a time. Use the second format of the utility to perform batch conversion of several lines of text in one invocation. This utility also allows you to generate and play mulaw (*.ulw) file formats by using the -cm option.

The parameters of text2pcm are:

-t = The text to be converted. The text string must be enclosed in quotes if the string contains embedded blanks. After the open quotation marks you must include the ' mark (hex symbol 60 or ASCII symbol 96). NOTE: Do not use the single quotation mark. The maximum string length is 512 characters.

-f = The name of the file containing the batch of lines of text to be converted. Each line in the batch must have the format:

<ofilename>@ <text><LF>

where ofilename is the name of the output file into which the converted data is placed and LF is the end-of-line character. ofilename is assumed to be in the ViaVoicePhone directory.

-w = Return the output in WAV format.

-c = Specifies the type of output Coding. All -c variations use mulaw (.ulw) with the exception of -c a|A which forces A-law. Since U.S. English systems typically produce mulaw output by default, although the "-c" can be used, it is not necessary.

-cm = Specifies the type of output Coding. -cm Allows you to generate and play mulaw (*.ulw) text-to-speech file formats.

-c a|A = Specifies the type of output Coding. Forces A-law encoding. Since non-U.S. English systems typically produce A-law output by default, although the "-ca" can be used, it is not necessary.
-p = Indicates the name of the output file. The ViaVoicePhone directory is the default.

-s = Set the speed for the speech resulting from the conversion. The valid speed range is 1 to 350. The default is 150.

-v = Set the volume for the speech resulting from the conversion. The valid range is 1 to 100. The default is 80.

-e = The output file extension. Specify ULW for a file in PCM format and WAV for a file in wav format. The default is .WAV.

-q = Operate in quiet mode. Do not log messages to the console.

-l = Specifies the language to be used to convert the text. By default, the language is U.S. English. Specify this parameter whenever the language of the application is not U.S. English. LI_LL can be one of the following:

   En_US for U.S. English
   En_UK for U.K. English
   Fr_FR for French
   Gr_GR for German

Embedded Text-to-Speech (tts) Tags

The Run Time product supports standard embedded text-to-speech annotation tags. These tags are inserted into the input text to affect how it sounds when read aloud. However, when using embedded text-to-speech tags, the \*:Tts.allowAnnotations resource must be set to 1 (or yes) and the embedded tag must start with a single quotation mark "". Here are some embedded tag examples that are used to set the type of character spelling mode (\xSpl=<value>):

<table>
<thead>
<tr>
<th>Tag</th>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\xSpl=off\</td>
<td>'ts0</td>
<td>No special interpretation (default setting).</td>
</tr>
<tr>
<td>\xSpl=alphanumeric\</td>
<td>'ts1</td>
<td>Pronounce only alphanumeric characters by name.</td>
</tr>
<tr>
<td>\xSpl=allchars\</td>
<td>'ts2</td>
<td>Pronounce all characters individually by name.</td>
</tr>
<tr>
<td>\xSpl=radio\</td>
<td>'ts3</td>
<td>Pronounce alphabetic characters according to the International Radio Alphabet.</td>
</tr>
</tbody>
</table>

For a listing of all supported embedded tts tags, please refer to the "ViaVoicePhone\tts\ENUref.hlp" online help file that comes with the ViaVoice Outloud Run Time product.
Glossary

A

annotation
An alphanumeric string used to mark a grammar when it is defined. When the grammar is used in an application, both the word and the alphanumeric string are returned to the application.

B

baseforms
The set of phonetic pronunciations associated with a grammar. In ViaVoice, the IBM dictionary of pronunciations is used.

bnf
(1) Abbreviation for Backus-Naur Form, used to describe the syntax of a given language and its notation. In speech recognition, a special adaptation of grammar representation specified by Speech Recognition Control Language (abbreviated SRCL and pronounced "circle").
(2) In ViaVoice, the extension of a file that contains grammar specifications in source form. Such a file is called a .bnf file.

C

channel process
The telephony process that manages the call flow for a voice channel. A channel process is created for each voice channel supported.

D

dynamic vocabulary
A vocabulary that is defined while an application is running.

E

engine
See speech engine.

engine channel process
A separate instance of the ViaVoice speech recognition engine. It is associated with a single Tcl channel process.

F

fsg
Abbreviation for finite state grammar. In ViaVoice, the extension of a file that contains grammar specifications in compiled, binary form. It is generated from a .bnf file and is called an .fsg file.
G

grammar, phone
A structured collection of words and phrases bound together by rules. It defines the set of all words, phrases, and sentences that may be spoken by a caller, and are recognized by the telephony recognition engine.

H

I

IVR
Abbreviation for interactive voice response; uses pre-recorded voice responses to supply information in response to voice input from a telephone caller.

J - L

M

mumble
Non-speech noise that a user interjects while speaking.

N

Nbest
The term for obtaining a list of alternatives from the speech recognition engine for the recognized text.

NMS
Abbreviation for telephony hardware manufacturer Natural Microsystems.

O

P

phone grammar
See grammar, phone

phone recognition
Communicating with a computer using voice via a telephone, over a telephone line. The computer application recognizes what was said and takes appropriate action.

Process Manager
The process that manages the interaction of all telephony system processes. The Process Manager is also known as pm.

procman.cfg
The file through which the Process Manager is configured.

pronunciation
The possible phonetic representations of a word. A word can have multiple pronunciations; for example, "the" has at least two pronunciations, "thee" and "thuh."

pronunciation dictionary
A file that contains the phonetic representation of all of the words, phrases, and sentences for an application's grammars. Also called baseforms.

pronunciation pool
The pronunciation files used by the ViaVoice speech recognition engine.

Q - R

S
silence
A brief pause between utterances.

speech engine
Part of ViaVoice; an application that performs all speech processing tasks and maintains the acoustic and word-usage models.

speech recognition
Primary function of the speech engine; the engine recognizes speech input and translates it into text that an application understands.

Speech Recognition Control Language (SRCL)
A structured syntax and notation used to define speech grammars; defines annotations, repetitions, words, phrases, and associated rules.

subscriber.db
A text file in a prescribed format used for defining vocabularies for ViaVoice Telephony.

T
T1
Digital telephony standard that provides 24 voice channels which are multiplexed on a single cable.

Tcl/Tk
Abbreviation for Tool Command Language/Toolkit; a scripting language that provides generic programming facilities and a toolkit of graphical objects for building user interfaces.

Tcl channel process
In a Tcl/Tk application, a process that manages a single voice channel.

text-to-speech engine
Provides synthesized speech output. It is part of the ViaVoice Outloud Run Time.
U

**utterance**
In ViaVoice, any stream of speech between two periods of silence.

V

**vocabulary**
The set of words, phrases, and sentences that a caller can say and be recognized by the engine.

**VTTDefaults**
The file through which a ViaVoice Telephony system is configured.

W - Z