AT&T Natural Voices™

Text-To-Speech Engines

System Developer's Guide

Server, Server-Lite, and Desktop Editions

US English, Latin American Spanish, UK English, German, and French

Release 1.4

http://www.naturalvoices.att.com

help@naturalvoices.att.com

AT&T

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Contents

1 Introduction

1.1. AT&T Natural Voices Text-To-Speech

AT&T Natural Voices Text-to-Speech (TTS) Engines provide synthesis services in multiple languages for application builders creating desktop applications. The application requests services from the TTS engine using either a Microsoft SAPI 4 COM object, SAPI 5.1 COM object, a Java program, or using a convenient set of C++ APIs that are included with the product. High-quality male and female voices are included in 8 KHz μlaw and CCITT G.711 alaw for telephony applications. Additional voices and higher quality 16 KHz voices are available for non-telephony applications. The AT&T Natural Voices TTS product is currently available in three configurations: the **Server Edition**, the **Server-Lite Edition**, and the **Desktop Edition**. The Server Edition provides a scalable, client/server architecture that's ideal for supporting many simultaneous client connections to a single server or a large server farm of TTS servers. The Server-Lite Edition offers the same TTS technology, but on a single server where both the application and the TTS run on a single computer making the Server-Lite Edition ideal for small office applications requiring fewer than about 10 simultaneous TTS channels[1]. The Desktop Edition provides the same AT&T Natural Voices TTS technology on a desktop computer but limits the TTS engine to a single channel. All three configurations are intended for application developers to use to build TTS-enabled applications, either for scalable applications using the Server

[1] The actual number of channels you may be able to support with the Server-Lite Edition depends upon the other applications that share the computer's processing power and memory. Text To Speech Synthesis tends to be memory intensive rather than CPU intensive so adding physical memory may allow you to run additional channels.

Edition, small to mid-sized applications using the Server-Lite Edition, or end user desktop applications using the Desktop Edition.

The following table shows the differences between the three editions of the AT&T Natural Voices product.

Feature/Requirement	Server	Server-Lite	Desktop
Architecture	Client/Server	Single computer	Single computer
Scalability (simultaneous channels)	Unlimited channels across multiple TTS servers	Several channels on one server	Single channel
Design goals	Large channel capacity	Single computer with smaller memory requirements	Single computer with smaller memory requirements
Recommended Memory	512 MB (256 Minimum)	256 MB (128 Minimum)	256 MB (128 Minimum)
Minimum CPU	500 MHz PIII	300 MHz PIII	300 MHz III
Disk Space	500 MB	500 MB	500 MB
Platforms ▪ Windows NT, 2000, XP ▪ Windows 98, ME ▪ Linux 6.1, 6.2, 7.2 ▪ Solaris/SPARC 2.7, 2.8 ▪ HPUX 11.0 ▪ AIX 4.3	 √ Client only √ √ √ √	 √ √ √ √ √ √	 √ √ √
SNMP Support	√		
Multi-Lingual	√	√	√
Voice Fonts	√	√	√
Development SDK and redistributable TTS engine	√	√	√
SAPI 4.0	√	√	√
SAPI 5.0 & 5.1[2]	√	√	√
SSML	√	√	√
JSML	√	√	√
Custom dictionaries; Word and sentence notifications; user-defined bookmarks	√	√	√

This document describes the AT&T Natural Voices Text-To-Speech Development SDK which application developers use to create TTS-enabled applications using US English, German, and Latin American Spanish, UK English, and French voices both for large capacity and desktop environments. The SDK is the same for all three editions, the only code difference being in creating the engine.

[2] SAPI 5.0 does not support Spanish. You must upgrade to SAPI 5.1 to use the multi-lingual features of the AT&T Natural Voices TTS 1.4 Release.

1.2 The Text-to-Speech Synthesis Problem

Text-to-Speech Synthesis (TTS) is the creation of audible speech from machine-readable text. TTS technology is useful whenever a computer application needs to communicate with a person. Although recorded voice prompts can sometimes meet this need, they provide limited flexibility and can be prohibitively expensive for high-volume applications. Thus TTS is especially helpful in telephone services, providing general information such as stock market quotes and sports scores, and reading e-mail or Web pages from the Internet over a telephone.

High quality speech synthesis is a technically demanding task. A TTS system has to model both the generic, phonetic features that make speech intelligible, and the idiosyncratic, acoustic characteristics that make it human. While text is rich in phonetic information, it contains little or nothing about the vocal qualities that denote emotional states, moods, and variations in emphasis or attitude. The elements of prosody—register, accentuation, intonation, and speed of delivery—are barely represented in the orthography (written representation) of a text. Yet without them, a synthesized voice sounds monotonous and unnatural. Inadequate or poorly crafted prosody also can lead to incorrect interpretation and listener fatigue during long texts.

The problem of generating speech from written text can be divided into two main tasks:

Text/linguistic analysis. The text must be converted into a linguistic representation that includes the phonemes to be produced, their duration, the location of phrase boundaries, and pitch/frequency contours for each phrase.

Synthesis. The information obtained in the linguistic analysis stage must be converted into an acoustic waveform.

Until quite recently, most commercial text-to-speech synthesizers produced audio that was too poor in quality to gain widespread acceptance. The most popular approach is concatenation of speech elements. AT&T's first-generation speech synthesizer relied on diphone concatenation using linear predictive coding (LPC).[3] Such systems produce speech with a high degree of intelligibility, but do not sound particularly natural. Typically, only one choice is available for each diphone transition, and the voices produced tend to sound "buzzy". The advantage, however, is that the parametric representation of speech allows a high degree of runtime control over fundamental frequency, speed, volume, and other characteristics, since the parameters for synthesis can be varied.

To improve naturalness, AT&T Natural Voices TTS system uses unit selection synthesis, which requires a set of pre-recorded speech units that can be classified into a small number of categories, each having sufficient examples of each unit to make statistical selection viable. Further, by using half phones as the basic units and employing sophisticated search and joining techniques, AT&T has been able to achieve an unprecedented degree of naturalness, while retaining intelligibility and real-time performance.

---

[3] Sproat, Richard W., and Olive, Joseph P. 1995. Text to Speech Synthesis. *The AT&T Technical Journal* (March/April 1995): 35-44.

1.3 Release 1.4 Features

This document describes the Server, Server-Lite, and Desktop Editions of Release 1.4 which builds on the previous releases, and provides a number of lexical analysis improvements for the US English voices including:

- Better end of sentence and paragraph detection to add pauses following sentences and paragraphs
- Better handling of dates and times
- Support for ranges of numbers, e.g. 3-5, dates, e.g. 4/3/99 – 4/20/99, and times, e.g. 3:00-4:00
- Better support for lists
- Better handling of words in capital letters.
- Support for multi-word dictionary items, e.g. “Los Angeles”
- Better handling of “decorative” characters
- The Desktop and Server-Lite Editions for Microsoft Windows platforms no longer use TCP/IP sockets which has caused confusion and performance problems for users with software firewalls.

The AT&T Natural Voices architecture supports many voices in multiple languages including US English, German, and Latin American Spanish, UK English, and French. More voices and languages will be released in the coming months.

AT&T Natural Voices includes the following features:

- The Server Edition supports a client/server architecture where the TTS client and server applications run on a variety of operating system and hardware platforms.
- The Server-Lite and Desktop Editions support an architecture where the TTS engine runs on the same computer as the application software and provides the same great TTS technology as the Server Edition but in a smaller footprint with lower channel capacity.
- All three editions provide support for Microsoft SAPI 4.0 and 5.1 for Microsoft Windows 98, ME, NT, 2000, and XP clients.
- All three editions include a Java language SDK that allows application developers to control the TTS engine.
- All three editions include a C++ language SDK that allows application developers to control the TTS engine. The Client SDK provided with the product includes source code for sample applications that demonstrates how applications interact with the TTS engine.
- The TTS engine runs on both single and multi-processor computers.
- The TTS engine accepts text input and returns 8 KHz μLaw, CCITT G.711 alaw, and wav audio with synthesized speech corresponding to the text input. 16 KHz voices are also available for even higher quality audio output.
- SNMP support for remote monitoring and administration is provided in the Server Edition.

- The Server Edition TTS engine is available as a Windows NT, Windows 2000, or Windows XP Service.

All three editions support US English, UK English, French, German, and Spanish. All versions allow you to add additional voices, languages, and sample rates, including 16 KHz versions of the voices. More voices and languages will be offered in the near future. Please visit www.naturalvoices.att.com for the latest information.

The TTS engines support the Java Speech Markup Language, the Speech Synthesis Markup Language component of the Voice XML standard, the Microsoft SAPI 4.0 Markup language, and the Microsoft SAPI 5.1 markup languages. These markup languages allow client applications to include special instructions within the input text that may change the default behavior of the text synthesizer including the following features (not all features are supported in all languages):

- Specify a set of phonemes to be spoken, allowing the application to control the exact pronunciation of a word or phrase;
- Change the default behavior for synthesizing numbers including support for fractions, measurements, cardinal numbers, decimals, money, and simple mathematical equations;
- Synthesize text containing dates, addresses, phone numbers, and times;
- Change the speaking rate, volume, and voice.
- Supports custom pronunciation databases for applications and users to change the default pronunciation of words and phrases. Release 1.4 also supports speaker-dependent dictionaries to allow applications to fine tune pronunciations for a specific voice.
- Provides user-defined bookmarks which allow the application to be notified when specific areas in the text are spoken.
- Allows optional notifications of events including word boundaries and phoneme boundaries.

See the chapters on the SAPI 4, SAPI 5, SSML, and JSML for more details.

The following table lists the major features and identifies the first product release which supports the feature:

Feature	**1.0 Feature**	**1.1 Feature**	**1.2 Feature**	**1.2.1 Feature**	**1.3 Feature**	**1.4 Feature**
Client/Server architecture with SNMP	√		√	√	√	√
Server-Lite & Desktop architecture		√	√	√	√	√
SAPI 5.0 & 5.1 Support	√	√	√	√	√	√
SAPI 4					√	√
SSML support	√	√	√	√	√	√
JSAPI/JSML support					√	√
C++ SDK	√	√	√	√	√	√

Single and multiple CPU	√	√	√	√	√	√
PCM, μLaw, alaw, and wav audio	√	√	√	√	√	√
US English	√	√	√	√	√	√
Spanish			√	√	√	√
German				√	√	√
UK English				√	√	
French				√	√	√
Voice-specific custom dictionaries			√	√	√	√
Language-specific custom dictionaries					√	√
Transcription API and GUI					√	√
Multi-word dictionary items						√
Lexical analysis improvements for US English including better handling of upper case characters, dates, times, etc.						√

1.4 System Components and Features

The following software libraries and executables are included in the AT&T Natural Voices TTS 1.4 products:

- **Installation package**. Installs the AT&T Natural Voices TTS engine, documentation, tools, class libraries, sample applications, and demo applications onto the target system. See the Installation chapter for more information.
- **AT&T Natural Voices TTS Engines**. The Server Edition TTS engine is provided as a standalone executable and supporting data files. The Server-Lite and Desktop Editions provide the TTS engine as a set of libraries and proxy applications that can be linked with your application or run as a proxy.
- **SDK**. This tool kit is an integrated collection of a C++ and Java classes and libraries to help developers integrate Text-To-Speech capabilities into their applications with ease.
- **Sample Applications**: The SDK includes sample applications that can be used to explore potential uses of the SDK and TTS server.
- **Nuance Integration Package**. The Server Edition for Microsoft Windows and Solaris operating systems include utilities to allow you to integrate Nuance ASR features with AT&T Natural Voices TTS.

1.5 Supported Languages

AT&T Natural Voices TTS engine, Release 1.4 is configured for US English for American English and includes both a male and a female voice. Latin American, Spanish female, German male and female, UK English male and female, and Parisian French male voices are also available. All configurations allow you to add languages and voices to the initial set of languages and voices. Editions tailored for other languages including Canadian French and other languages and more custom voices will become available in future releases. Visit http://www.naturalvoices.att.com for the latest information.

1.6 This Guide

The remainder of this Developer's Guide discusses the basic development of AT&T Natural Voices TTS applications. Topics include the following:

- Installation Guidelines – Learn how to install the client and server software.
- Server Guidelines – Learn how to administer the AT&T Natural Voices TTS server and understand what to expect from it as it runs including performance guidelines and SNMP features.
- Server-Lite and Desktop Guidelines – Learn about the Server-Lite and Desktop engines and applications you can use to access the TTS engines.
- SDK description – Explore the features included in the C++ SDK that will help you to write your applications.
- Accessing the engine from Java – Learn how to access the TTS engine from a Java application.
- Text Markup Languages – Use XML tags to change the default behavior of the TTS engine.
- Using custom dictionaries to tailor pronunciations.
- Performance Guidelines – Find heuristics for estimating system performance and hints for improving channel capacity.
- IPA Phonetic Alphabet – SAPI 4 and JSML use IPA to specify pronunciations.
- DARPA and SAPI Phonetic Alphabets – You may specify exact pronunciations for English words using DARPA or SAPI phonemes.
- SAMPA Spanish Phonetic Alphabets – You may specify exact pronunciations for Spanish words using SAMPA phonemes.
- SAMPA German Phonetic Alphabets – You may specify exact pronunciations for German words using SAMPA phonemes.
- SAPI Compliance – Check our compliance with the SAPI 4.0 and SAPI 5.1 specifications.

1.7 Other Sources of Information

The Microsoft Speech API 5 Developer's Guide contains detailed information on Microsoft's specification for speech applications and hardware. It is available from Microsoft from http://www.microsoft.com/speech.

Microsoft SAPI 4 is available for download from http://activex.microsoft.com/activex/controls/sapi/spchapi.exe

The Speech Synthesis Markup Language component of Voice XML is described at http://www.w3.org/TR/speech-synthesis.

More information about the Java Speech API is available at http://java.sun.com/products/java-media/speech/.

Support for AT&T Natural Voices TTS is available via our website, by electronic mail, or by telephone.

For technical support:

- See http://www.naturalvoices.att.com where you'll find the latest voices, languages, software examples, patches, and the most up-to-date information about the product.
- Participate in our user forums at http://forum.naturalvoices.com.
- Send email to help@naturalvoices.att.com. A technical expert will respond promptly to your email.
- Call (877) 741-4321 toll-free within the United States to speak to our technical support team. International callers should call (973) 360-8513.

For sales information, call (877) 741-4321 or (973) 360-8513.

2 Installation

2.1 Installing the AT&T Natural Voices TTS Software

The AT&T Natural Voices TTS Server Edition software is available for the following platforms:

- Microsoft NT 4.0, Service Pack 5 on Intel platforms
- Microsoft Windows 2000, Service Pack 2 on Intel platforms
- Microsoft Windows XP on Intel Platforms
- RedHat Linux 6.1, 6.2, and 7.1 on Intel platforms
- Sun Solaris 2.7 and 2.8 on Intel and Sparc platforms
- HP HPUX 11.0 on PA/RISC processors
- IBM AIX Version 4.3 on RS/6000 processors

The Server-Lite Edition is available for the following platforms:

- Microsoft NT 4.0, Service Pack 5 on Intel platforms
- Microsoft Windows 2000, Service Pack 2 on Intel platforms
- Microsoft Windows 98, ME, and XP on Intel Platforms
- RedHat Linux 6.1, 6.2, and 7.1 on Intel platforms
- Sun Solaris 2.7 and 2.8 on Intel and Sparc platforms
- HP HPUX 11.0 on PA/RISC processors
- IBM AIX Version 4.3 on RS/6000 processors

The AT&T Natural Voices Desktop Edition is available for the following platforms:

- Microsoft Windows 98 and ME
- Microsoft NT 4.0, Service Pack 5 on Intel platforms
- Microsoft Windows 2000, Service Pack 2 on Intel platforms
- Microsoft XP
- RedHat Linux 6.1, 6.2, and 7.1 on Intel platforms

Need a different platform? Let us know by sending email to help@naturalvoices.att.com and we'll be glad to discuss the details with you.

The installation procedure is different for Windows platforms and UNIX platforms. Please see the appropriate sections for Windows and UNIX platforms.

2.2 Disk Requirements

All three Editions of the AT&T Natural Voices TTS Engine application and associated data files require approximately 500 MB of disk space if you choose to install the complete package. You can choose to install only one voice which will save about 200MB of disk space. You will require some additional space to store server logs if running the Server Edition.

The TTS Engine uses large data files to support speech synthesis. The data files that are installed should always be stored on a local hard drive rather than on network file system to optimize system performance.

2.3 Memory Requirements

We recommend at least 256MB of physical RAM for the Server-Lite and Desktop Editions although the engine will work in 128 MB of memory. We recommend at least 512MB of physical RAM for each CPU running the Server Edition. Additional memory will provide better performance for all three Editions if you use the TTS engine while other applications are running. This provides sufficient memory to allow the operating system to load most of the time critical data structures into memory rather than paging them from disk. The Performance Guidelines chapter describes the memory and performance of the system in more detail.

2.4 Installing AT&T Natural Voices TTS Engines on Microsoft Windows

The installation procedure for the Server, Server-Lite, and Desktop Editions are very similar and differ only in the final steps. The AT&T Natural Voices TTS software is distributed on one CD for Microsoft platforms. The AT&T Natural Voices TTS software can be used with Microsoft SAPI 4.0 or SAPI 5.1 or you may choose to build your application using the C++ or JAVA APIs and not rely on SAPI 4 or SAPI 5. The install package included on the CD optionally installs the C++ SDK, the JAVA SDK, the Microsoft SAPI 4 COM object, and the Microsoft SAPI 5.1 COM object. You'll need to install the Microsoft SAPI SDK separately but the SAPI 4 and 5.1 Installation packages are included on the AT&T Natural Voices TTS 1.4 distribution CDs for your convenience.

Note: SAPI 5.0 does not support German or Spanish so you must install or upgrade to SAPI 5.1 if you wish to access the German or Spanish voice from a SAPI 5-compliant application. SAPI 4 supports US English, Spanish, and German.

A major difference in the installation procedure for the Server Edition is that the Server Edition allows you to install the client and server on the same or different machines. Installing the client for the Server Edition also includes an extra step for specifying the name and port number of the server to which the client should attach.

Use the following procedure to install the Server, Server-Lite, and Desktop Editions of the TTS software on your computer. You will need the AT&T Natural Voices TTS CD-ROM and about 500 MB of free hard-disk space.

Step 1. Insert the AT&T Natural Voices TTS Server, Server-Lite, or Desktop CD-ROM in a CD-ROM drive. If AUTORUN is enabled for your CD-ROM drive, the install will automatically run. You may skip to Step 3.

Step 2. If the installation does not begin automatically, use the **Add/Remove Programs** item of the **Control Panel** to run the SETUP.EXE application located on the AT&T Natural Voices TTS Server, Server-Lite, or Desktop CD-ROM. Follow the instructions on the screen.

Step 3. You may choose to set your default voice for Microsoft SAPI 5.1 applications. This step is not required but will save you time if you use the Microsoft SAPI 5.1 applications frequently, such as the TTSApp. You can set the default voice from the Speech applet on the Microsoft Windows Control Panel.

Step 4. If you are installing the Server-Lite or Desktop Editions then the software is now installed and you may want to skip to the next section on setting the default SAPI 5.1 voice. If you're installing the Server Edition, please read on for more instructions.

Step 5. If you are installing the Client for the Server Edition, the final step of the installation process runs an application called `WinVoiceEdit.exe` which will ask you to specify a TTS Server name and a port for each of the SAPI 5 voices. If you install the SAPI 4 client, you'll also be asked to specify the server name and port for each of the SAPI 4 voices. You'll need to know the name of the server and the port number for the server you wish to access. The port number must match the port number that the TTS Server specifies when starting. If you later decide to change the server and/or port number, you can run the `WinVoiceEdit.exe` or WinVoiceEdit4.exe application, located in the bin directory or from

```
Start->Programs->AT&T Natural Voices 1.4->ClientServer->
      Client->WinVoiceEdit
```

for SAPI 5 clients or

```
Start->Programs->AT&T Natural Voices 1.4->ClientServer->
      Client->WinVoiceEdit4
```

for SAPI 4 clients.

Figure 2.4.1. TTS SAPI 5.1 Voice Setup Dialog

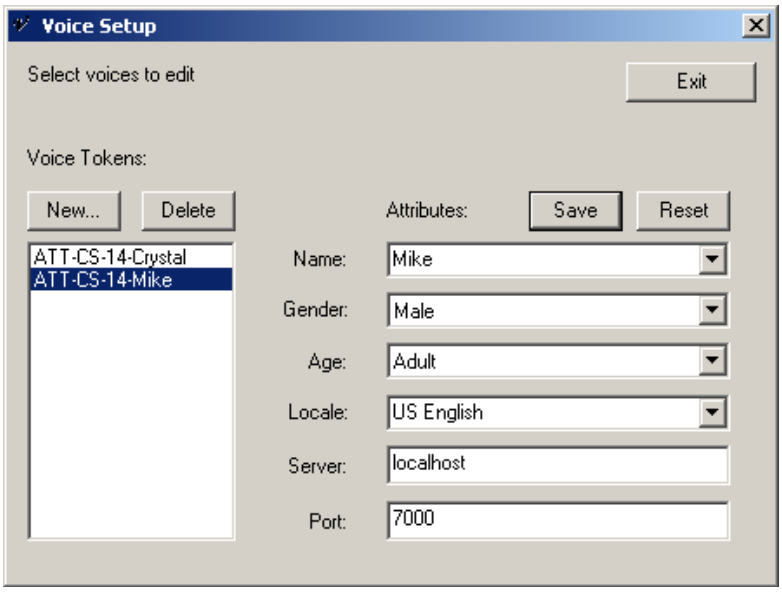
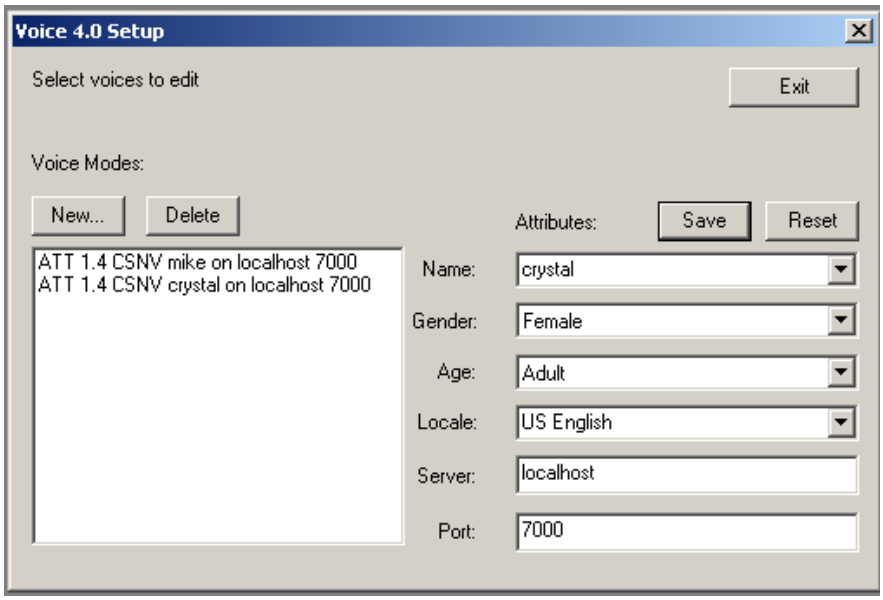


Figure 2.4.1 shows the dialog from WinVoiceEdit.exe that appears as the final step in the client installation if you installed the SAPI 5.1 client. The dialog allows you to set or change the server name and port number for the voices that included in the 1.4 Release. This example shows the Rosa voice running on the same server as the client using the default ports. Note that the Locale for Rosa is Spanish, while the Locale for Crystal and Mike is English. The locale for Reiner and Klara, the male and female German voices is German. The install script will set this for you. You may specify any server name and port number to match your environment. You may also use the WinVoiceEdit tool to create links to several different servers, e.g. you may have a TTS Server running the Crystal voice on two different servers. The Voice Setup tool allows you to add new entries that specify the voice, the server, and the port number. There is no limit to the number of servers that you can access from SAPI using the WinVoiceEdit application.

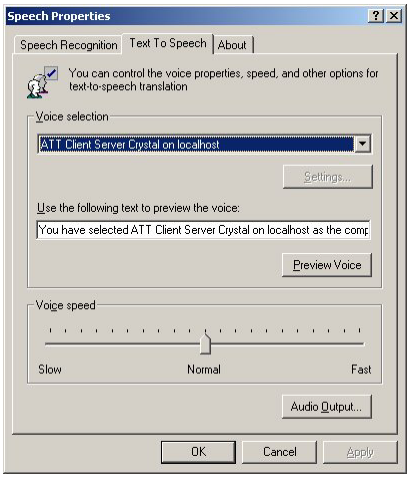
The WinVoiceEdit4.exe application for SAPI 4 clients is very similar to the WinVoiceEdit.exe application and is shown in Figure 2.4.2.

Figure 2.4.2. TTS SAPI 4 Voice Setup Dialog



Step 6. You may choose to set your default voice for the Microsoft SAPI 5 applications, including the TTSApp which is included in the Microsoft SAPI 5 SDK. This step is not required but will save you time if you use the TTSApp or other SAPI-compliant application frequently. You can set the default voice from the Speech applet on the Microsoft Windows Control Panel.

Figure 2.4.3. The Speech Control Panel Applet



2.5 Installing the AT&T Natural Voices TTS Software on UNIX Machines

The Server, Server-Lite, and Desktop Editions are delivered for UNIX platforms on a single CD. The Server Edition includes separate files for the client and server functionality along with a README file. This allows you to install the client and server

functionality on the same or separate machines. The Server-Lite and Desktop Editions include a single tar file.

2.5.1 Installing the AT&T Natural Voices Server-Lite and Desktop Editions

The TTS package is delivered for UNIX platforms on a single CD with the TTS functionality along with a README file. The tar files will unpack into a single directory called tts.<platform>.desktop or tts.<platform>.serverlite where <platform> is linux, sun, suni, aix, or hpux.

You may safely unpack the server tar file in any directory that has at least 500 MB of free space. To extract the files from the tar file, use the command:

```
tar xvf <CDRomDrive>/tts.linux.desktop.tar
```

where `<CDRomDrive>` is your CD drive, e.g. /dev/cdrom.

For example,

```
cd /usr/local
tar xvf /dev/cdrom/tts.linux.desktop.tar
```

creates a new directory, **`/usr/local/ATTNaturalVoices/TTS1.4/desktop`**, and installs all of the files in that directory.

Be sure to read the README file on the CD to find the latest information. The Server-Lite and Desktop Editions require no additional set up.

2.5.2 Installing the Server Edition TTS Server on UNIX Servers

You may safely unpack the server tar file in any directory that has at least 500 MB of free space. To extract the files from the tar file, use the command:

```
tar xvf <CDRomDrive>/tts.<platform>.server.tar
```

where `<CDRomDrive>` is your CD drive, e.g. /dev/cdrom, and `<platform>` is *linux, suni, sun, aix, or hpux.*

For example,

```
cd /usr/local
tar xvf /dev/cdrom/tts.linux.server.tar
```

creates a new directory, **/usr/local/ATTNaturalVoices/TTS1.4**, and installs all of the server files in that directory.

Be sure to read the README file on the CD to find the latest information. See Running the Server for information on running the TTS Client and The Client SDK for information about writing client applications.

2.5.3 Installing the Server Edition TTS Client on UNIX Machines

The Server Edition TTS client package may be installed on the same machine as the TTS server or on any other machine. You may safely unpack the client tar file in any directory that has at least 20 MB of free space. To extract the files from the tar file, use the command:

```
tar xvf <CDRomDrive>/tts.<platform>.client.tar
```

where *<CDRomDrive>* is your CD drive, e.g. /dev/cdrom, and *<platform>* is *linux, suni, sun, aix, or hpux.*

For example,

```
cd /usr/local
tar xvf /dev/cdrom/tts.linux.client.tar
```

creates a new directory, **/usr/local/ ATTNaturalVoices/TTS1.4,** and installs all of the client files in that directory. You may safely install both the client and server packages in the same directory on a single machine.

Be sure to read the README file on the CD to find the latest information. See Running the Server for information on running the TTS Client and The Client SDK for information about writing client applications.

3 Using the TTS Server Engines

This chapter explains how to run the Server Edition of the AT&T Natural Voices TTS engines, explains how to set up the client and server, and describes the sample applications that are included with the Server Edition. Chapter 5 details the APIs that are available for your application to use.

The AT&T Natural Voices Server Edition supports a client/server architecture where the client and server may run on the same or different computers. First, we'll explain how to start the server, then we'll describe the different client applications that are included with the product.

3.1 Server Command Line Arguments

The AT&T Natural Voices TTS Server may be started on the command line using arguments to specify the maximum number of ports, the voice, and the port number to listen for requests. The command line arguments are same for both Microsoft and UNIX platforms.

Synopsis:

```
TTSServer -r rootFilePath -c connectionPort
        [-x voiceName] [-y snmpPort] [-vV?][-v0] [-v1] [-v2]
         [-v3] [-d AudioFile] [-m maxClients]
         [-l logFilePath]
```

Options:

Option	Argument	Action
-r	Path of configuration data (required)	File path of the TTS data files. Typically the `data` subdirectory where you installed the Server Edition package.

-c	Connection port (required)	Identifies the port on which TTSServer will be listening for client connections. Note that the client application must be configured to use this same port.
-x	Default voice selection: - crystal - mike - rich - rosa - claire - klara - reiner	Specify the default voice: *Crystal*, the female voice, and *Mike,* the male voice, are the two voices provided with the 1.4 release. Rosa is the female Spanish voice, Klara and Reiner are the female and male German voices. You may specify only one voice as the default. You may specify any voice that you have installed on the server.
-v0	Minimal Server trace level	Prints only error messages.
-v1	Default Server trace level	Prints error messages, voice information, and process start/stop information.
-v2	Verbose Server trace level	Prints error messages, voice information, process start/stop information, and request and reply packets. Beware that-v2 mode generates extensive information and should be used sparingly.
-v3	Verbose mode	Prints error messages, voice information, process start/stop information, request and reply packets, and all notifications. Beware that-v3 mode generates extensive information and should generally not be used.
-v	equivalent to –v2	Prints error messages, voice information, process start/stop information, and request and reply packets. Beware that-v2 mode generates extensive information and should be used sparingly.
-V	Version	Print the TTSServer version.
-?	Help	Print command line options.
-d	Debug mode	The synthesized audio from each request is written to a local file on the server with the file name `AudioFile` with the child process ID appended. The audio is sent to the audio output socket as usual.
-m	Maximum simultaneous clients (default = 32)	Maximum number of clients. Attempts to allocate additional channels will fail.
-y	SNMP Port	Access this port for SNMP.
-l	Log file path	Path of a file where messages are logged.

UNIX Examples:

```
/usr/local/ATTNaturalVoices/TTS1.4/server/bin/TTSServer \
      -y 8000 -r /usr/local/ATTNaturalVoices/TTS1.4/data \
      -c 7000 -x crystal -l ttslog
/usr/local/ATTNaturalVoices/TTS1.4/server/bin/TTSServer \
      -y 8000 -r /usr/local/ATTNaturalVoices/TTS1.4/data \
      -c 7000 -x rosa -l ttslog
```

Windows Examples:

```
C:\program
files\ATTNaturalVoices\TTS1.4\ClientServer\bin\TTSServer \
      -r "C:\program
files\ATTNaturalVoices\TTS1.4\ClientServer\data" \
      -c 7000 -x crystal -y 8000 -l ttslog
C:\program
files\ATTNaturalVoices\TTS1.4\ClientServer\bin\TTSServer \
      -r "C:\program
files\ATTNaturalVoices\TTS1.4\ClientServer\data" \
      -c 7000 -x rosa -y 8000 -l ttslog
```

3.2 Running the TTS Server as a Microsoft Service

The TTS Server, `TTSServer.exe`, can also run as a Windows NT, 2000, or XP service which can optionally start automatically when your server boots and automatically restart the TTSServer if the process dies unexpectedly. A separate executable, `TTSService.exe` manages this process.

The `TTSService.exe` application is used to install and uninstall the service and to modify the service parameters.

3.2.1 Installing the ATT_TTSService Service.

By default, the AT&T Natural Voices TTS engine installation package installs the package for you. If you should ever need to install it, the first step in setting up the TTSServer as a service is to install the service. This is accomplished by running the TTSService.exe application with the -i option. (See Modifying the ATT_TTSService parameters for additional options to the -i command.)

For example:

```
c:\program
files\ATTNaturalVoices\TTS1.4\ClientServer\bin\ttsservice -i
ATT_TTSService 1.4 installed.
Updating parameters.
```

```
Parameter update succeeded.
```

With no additional options, the ATT_TTSService will run the TTSServer with default command line arguments, using Crystal and the default voice and running on port 7000.

3.2.2 Running the ATT_TTSService service

Once the service has been installed, it is started, paused, continued, and stopped using the Microsoft Management Console, or the command line. The Microsoft Management console is available from the Services item on the Administrative Tools Applet in the Control Panel.

By default, the AT&T Natural Voices TTS product installation package installs the service for you but specifies that the service must be started manually. You can change this attribute so the service is automatically started for you when your server starts using the Services component of the Microsoft Management Console which is available from the Windows Control Panel's Administrative Services Applet under Services. Here you can change the parameters and specify that the service be started automatically.

From the Command line:

```
c:> net start ATT_TTSService14
c:> net stop ATT_TTSService14
c:> net pause ATT_TTSService14
c:> net continue ATT_TTSService14
```

3.2.3 Uninstalling the ATT_TTSService

The TTSService is uninstalled by running the TTSService.exe application with the -u argument. For example:

```
c:\program files\ATTNaturalVoices\TTS1.4\ClientServer\bin>
ttsservice -u

ATT_TTSService14 uninstalled.
```

3.2.4 Modifying the ATT_TTSService Parameters

The TTSService application runs the `TTSServer.exe` application to handle client requests. It creates the command line for executing the TTSServer application using the additional arguments given with the `TTSService.exe –i` command line argument. To change these arguments, simply run `TTSService.exe` with the `–i` argument again and a different set of options.

Synopsis:

```
TTSService
   [-c connectionPort][-y snmpPort] [-i | -ia]
   [-m maxClients] [ -v ]  [-d AudioFilePath] [-r rootPath]
   [-p executablePath][-l logPath]
```

Options:

Option	Argument	Action
-i or –ia	none	Install the service either as manual start (-i) or autostart (-ia) service.
-c	Connection port	Identifies the port on which TTSServer will be listening for client connections.
-y	SNMP port	Identifies the port number the service listens to for SNMP connections.
-m	Maximum clients	Sets the maximum number of concurrent connections the service will allow.
-v	Verbose	Sets the service to run in verbose mode.
-d	Directory path for storing audio files for debugging	Sets the root path/file-name for dumping audio output to a file
-p	Executable path	Sets the path and executable name for the TTSServer application
-l	Log file path	Path of a file where messages are logged.

Windows Example:

```
c:\program files\ATTNaturalVoices\TTS1.4\ClientServer\bin> ttsservice
  -ia –c 7000 –y 8000 –m 32
  -r "\program files\ATTNaturalVoices\TTS1.4\ClientServer\data"
  -p "\program
Files\ATTNaturalVoices\TTS1.4\ClientServer\bin\TTSServer.exe"
Updating parameters.
Parameter update succeeded.
```

3.3 Supporting Multiple Voices and Languages

The AT&T Natural Voices TTS engine provides supports for multiple languages, including US English, UK English, French, German and Spanish. Unlike other TTS engine vendors who require that you run a separate instance of the TTS server for each language, the AT&T Natural Voices TTS engines allow you to mix languages on a single server simply by specifying a different voice. Users can specify a voice and language in a control tag that is included in the input text or make a Java or C++ API call and switch between voices and languages seamlessly. This architecture also allows us to deliver new voices in a variety of languages without requiring a new TTS engine executable.

The TTS Server allows you to specify a default voice when you start the TTSServer. That voice also specifies the default language as the language associated with the

default voice. An application can switch voices either from the SDK or by using control tags in the input text. Note that each voice requires about 75 MB of data to be accessed by the server so we recommend that you have at least 1 GB of memory per server CPU if you start more than one voice.

The AT&T Natural Voices TTS Release 1.4 Engine supports US English, UK English, French, German, and Spanish voices. To use the Spanish voice, simply choose the optional "Rosa" voice which is the Spanish female voice. Likewise, choose "Reiner" for the German male voice and "Klara" for the German female voice. You can even mix languages in a single input file by specifying the voice to use to speak the text. Note that any voice will attempt to speak whatever text is presented, i.e. Rosa will attempt to synthesize English text as Spanish words.

8 KHz and 16 KHz versions of all voices are available separately. The 8 KHz voices are ideally suited for telephony applications. The 16 KHz voices offer higher quality output for desktop applications including web site applications and wav files. You can install any combination of 8 KHz and 16 KHz voices on a single server. The voice naming convention is as follows:

Voice Name	Description	8 KHz Name	16 KHz Name
Crystal	US English female	crystal	crystal16
Mike	US English male	mike	mike16
Claire	US English female	claire	claire16
Rich	US English male	rich	rich16
Rosa	Spanish female	rosa	rosa16
Klara	German female	klara	klara16
Reiner	German male	reiner	reiner16
Audrey	UK English female	audrey	audrey16
Charles	UK English male	charles	charles16
Alain	French	alain	alain16

3.4 Running the Client

Once the server is up and running, you'll want to be able to send text to the TTS server to try it out. You can do with using either a Microsoft SAPI 4 or 5.1 compatible applications, one of the sample applications we include in the distribution, or you can write your own application in C++ or Java.

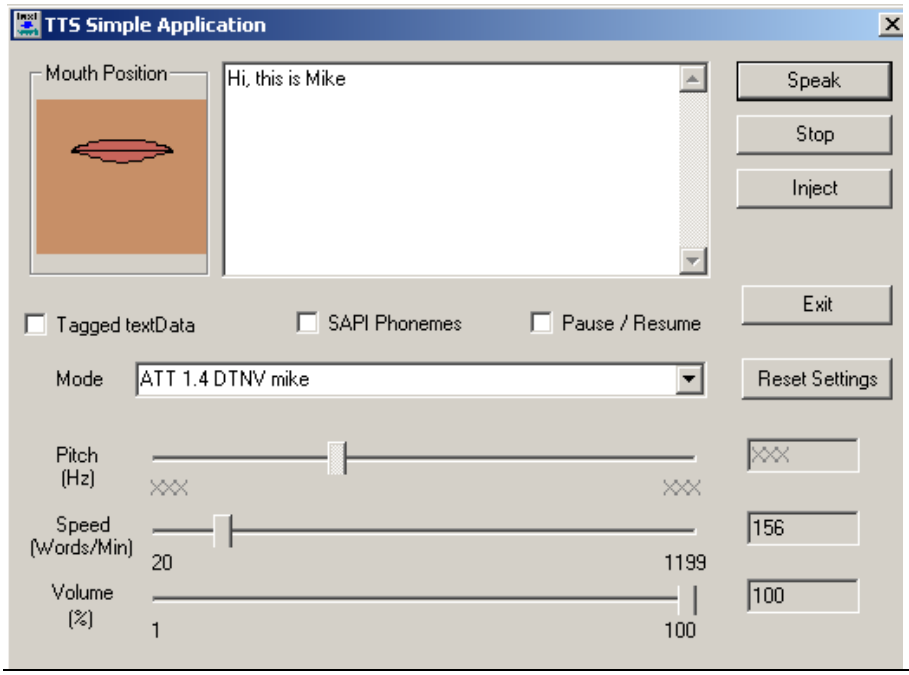
3.4.1 Microsoft Windows SAPI 4.0 TTSApp

If you're running on a Microsoft Windows 98, ME, NT, 2000, or XP client and you've installed the Microsoft SAPI 4 SDK then you may want to use the TTSApp that is included with Microsoft SAPI 4. This GUI client makes it easy to synthesize text that you type in. You'll find the TTSApp at

```
C:\Program Files\Microsoft Speech SDK\TTS\ttsapp.exe
```

Choose one of the AT&T Natural Voices TTS voices, Mike, Crystal, Rosa, Klara, or Reiner from the “Mode” selector as shown in Figure 3.4.1. You’ll also need to check off the “Tagged textDate” box if you’d like to include SAPI 4 control tags.

Figure 3.4.1. The SAPI 4 TTSApp



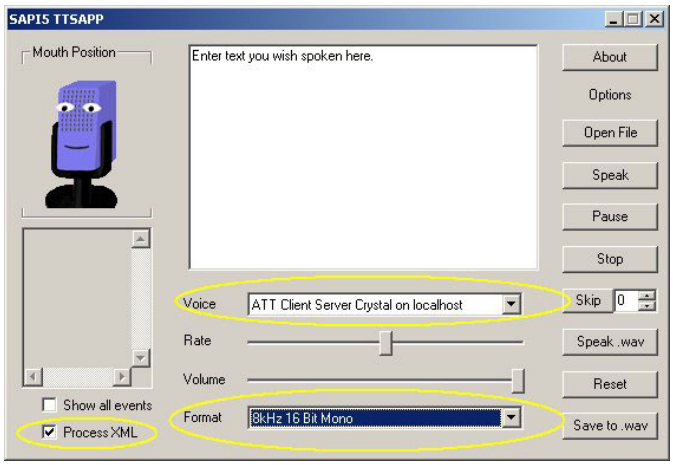
3.4.2 Microsoft Windows SAPI 5.1 TTSApp

If you're running on a Microsoft Windows 98, ME, NT, 2000, or XP client and you've installed the Microsoft SAPI 5.1 SDK then you may want to use the TTSApp that is included with Microsoft SAPI 5.1. This GUI client makes it easy to synthesize text that you type in or load from a file. You'll find the TTSApp at

Start->Programs->Microsoft Speech SDK 5.1->Tools->TTSApp

Choose one of the AT&T Natural Voices TTS voices, Mike, Crystal, Audrey, Charles, Alain, Rosa, Klara, or Reiner from the "Voice" selector and set the format to "8kHz 16 bit mono" as shown in Figure 3.1.4.1. Set the format to "16kHz 16bit mono" if you choose to use the Mike16, Crystal16, Rosa16, Klara16, or Reiner16 voices. You'll also need to check off the "Process XML" box if you'd like to include control tags.

Figure 3.4.2. The SAPI 5.1 TTSApp



3.4.3 Command Line TTS Clients

Two command line TTS clients, TTSClientPlayer and TTSClientFile, are included in the bin directory of the AT&T Natural Voices TTS distributions for both Microsoft Windows and Unix platforms. TTSClientPlayer allows you to synthesize text files and play them through a sound card. TTSClientFile synthesizes text files and writes the linear 16 bit PCM output as a wav file. These applications are useful for testing. We also include source code for these two applications to help you understand how to use the TTS engine in your applications.

Synopsis:

```
TTSClientPlayer -p connectionPort -s TTSServer
     [-a] [-xml] [-f inputFile] [-l fileList]
     [-du userDictionary] [-da applicationDictionary]
     [-phoneset att_darpa_english | att_sampa_spanish |
        att_sampa_german | att_sampa_ukenglish |
        att_sampa_french ]
     [ -r 8000 | 16000 ] [-mulaw | -alaw]

TTSClientFile -p connectionPort -s TTSServer
     -o audioOutputFile
     [-a] [-xml] [-f inputFile] [-l fileList]
     [-du userDictionary] [-da applicationDictionary]
     [-phoneset att_darpa_english | att_sampa_spanish |
        att_sampa_german | att_sampa_ukenglish |
        att_sampa_french]
     [ -r 8000 | 16000 ] [-queue] [-mulaw | -alaw]
```

Options:

Option	Argument	Action
-p	Connection port (required)	Identifies the port on which TTSServer will be listening for client connections.
-s	Server Name (required)	Specify the name or IP address of the TTS server.
-a	Asynchronous engine mode	Use asynchronous engine mode. The default is synchronous.
-xml	Parse XML tags as SAPI or SSML control tags	Default is not to parse XML tags.
-f	Input file name	The path of the text file to synthesize.
-l	File with a list of files to synthesize	Specify a file with a list of text files to synthesize.
-du	User dictionary file	Specify a user dictionary file.
-da	Application dictionary file	Specify an application dictionary file.
-phoneset	`att_darpabet_english`, `att_sampa_spanish`, `att_sampa_german`	Specify the set of phonemes that are used in the dictionary.
-r	Audio bit rate	Must be either 8000 for 8KHz or 16000 for 16KHz
-mulaw		Generate 8KHz, 8bit mulaw output
-alaw		Generate 8KHz, 8bit alaw output
-o	AudioOutputFile (TTSClientFile only)	Specify the output file where the audio will be written.

UNIX Example:

```
/usr/local/ATTNaturalVoices/TTS1.4/server/bin/TTSClientPlayer \
   -p 7000 -s localhost -f sample.txt
```

synthesizes sample.txt using the TTS server running on the local machine and listening on port 7000 and writes the audio to a sound card.

```
/usr/local/ATTNaturalVoices/TTS1.4/server/bin/TTSClientFile \
   -p 7000 -s localhost -f sample.txt -o audio.wav
play -t wav audio.wav
```

synthesizes the text in sample.txt, capturing the audio output to audio.wav then uses the UNIX play command to play the audio.

Windows Example:

```
C:\program files\ATTNaturalVoices\TTS1.4\ClientServer\
bin\TTSClientPlayer \
       -p 7000 -s localhost sample.txt
```

reads the contents of `sample.txt` using the TTS server running on the same computer, listening on port 7000.

```
C:\program
files\ATTNaturalVoices\TTS1.4\ClientServer\bin\TTSClientFile \
   -p 7000 -s localhost -f sample.txt -o audio.wav
```

synthesizes the text in sample.txt, capturing the audio output to audio.wav. Double click on audio.wav to play the audio. Note that Microsoft Windows does not allow you to write audio to stdout, thus you must use the –o option to write the audio to a file.

3.5 Server Output and Error Messages

The TTS Server writes all output and error messages to the log file specified with the –l option.

All messages have the format

Wire#*child time message*

e.g. Wire#1 171.90 : wireline started

where *child#* is a sequential number of speak requests, where each speak request creates a new child process and *time* is the number of seconds since the server started.

3.5.1 Initialization Messages

The following messages are generated by the TTS Server during initialization. These set of messages indicate that a wireline server started up. Wireline is the application itself, server engine is the thread that receives wireline requests and sends wireline replies, the asynch notify server is the thread that sends wireline notifications, and the asynch standalone is the thread that processes speak requests.

Wire#1 171.90 : wireline started
Wire#1 171.91 : server engine started
Wire#1 172.81 : asynch notify server started
Wire#1 172.85 : asynch standalone started

3.5.2 Shutdown Messages

The following messages are generated by the TTS Server during shutdown and map directly to the corresponding Server Initialization Messages.

Wire#1 177.56 : asynch standalone finished
Wire#1 177.56 : asynch notify server finished
Wire#1 177.56 : server engine finished
Wire#1 177.60 : wireline finished

3.5.3 Voice Inventory Messages

The following messages enumerate the voices that are available on the Server, the default voice and the current voice:

Wire#1 172.93 : Installed voices
Rosa;es_us;female;adult;8000;16;pcm;
Crystal;en_us;female;adult;8000;16;pcm;
Crystal16;en_us;female;adult;16000;16;pcm;
Mike;en_us;male;adult;8000;16;pcm;
Mike16;en_us;male;adult;16000;16;pcm;
Rich;en_us;male;adult;8000;16;pcm;
Rich16;en_us;male;adult;16000;16;pcm;
Wire#1 172.94 : Default voice: Rich16;en_us;male;adult;16000;16;pcm;
Wire#1 172.94 : Current voice: Crystal;en_us;female;adult;8000;16;pcm;

3.5.4 Client Messages

These messages may be displayed when a client disconnects from the server unexpectedly.

Wire#1 368.83 : receive error 10054 ::Connection reset by peer

Wire#1 368.83 : sending notify packet::Connection reset by peer

Wire#1 368.90 : wireline finished::Connection reset by peer

Periodically, the server will clean up resources for child processes that have exited.

TTSWireServer 395.02 : reaped child 1

These messages are simply housekeeping messages and can be ignored.

3.6 SNMP

The AT&T Natural Voices TTS package includes support for remote server administration via SNMP. This makes it possible to control and monitor the TTS server using the SNMP-compatible Management software of your choice. SNMP support included with the TTS package consists of the DLLs or libraries necessary to interface the TTS server with a server-side SNMP Agent, such as Microsoft's SNMP Service (for Microsoft platforms only) or NET-SNMP (Unix platforms.) Additionally, MIB files are included for use with most SNMP-compatible management console systems.

3.6.1 Configuring SNMP on Windows NT/2000/XP

The files necessary to support TTS/SNMP are copied by default when the TTS Server is installed on your machine. If you are using Microsoft Windows NT Server or 2000 Server, the Microsoft SNMP service should have been installed with the Operating System.

3.6.2 Verifying that the Microsoft SNMP Service is installed

To insure that the SNMP service is installed correctly:

1. Open the Windows Control Panel.
2. If you are using Windows NT, open the “Services” dialog directly from the Control Panel and skip to step 3. If you are using Windows 2000 Server, under “Administrative Tools”, double-click to open the “Computer Management” dialog box. On the left side of the Computer Management dialog, select “Computer Management (Local) > Services and Applications > Services”. The right side of the window will display a list of available system services.
3. Scroll down to select “SNMP Service” (if it is not listed, you must install the SNMP Service Component on your machine; look for directions under “SNMP Service, installing” in the Windows online help system.) Double-click to open the Properties dialog. If the status is listed as “Stopped”, click the “Start” button to activate the service. Under “Startup type”, select “Automatic” to insure that the SNMP Service will activate when the system is booted.

Microsoft SNMP allows you to set access control and security options for SNMP connections. Detailed instructions on these functions and how they may be configured can be found in the Windows Online Help under “SNMP Service”. Once the TTS Server and the Microsoft SNMP Service have been configured and started, you may begin using the service.

3.6.3 Configuring SNMP on Unix systems

TTS includes support for SNMP on Linux and Solaris systems through an extension to the NET-SNMP package. Formerly UCD-SNMP, NET-SNMP is a free software project, and may be obtained from the NET-SNMP project page (http://www.net-snmp.com). Installation and configuration instructions can be also found at this location.

In order to extend the NET-SNMP server for use with TTS, an Extension Agent is included with the TTS distribution in the form of a dynamically loaded shared object file (ttsSNMPAgent.so, located in the “snmp” subdirectory after installation).

NOTE: NET-SNMP versions prior to 4.1 do not include support for shared-objects, and therefore cannot be used with the TTS Extension Agent.

3.6.4 Configuring the snmpd.conf configuration file

Once you have installed and configured the NET-SNMP package, you must inform the daemon as to the location of the TTS SNMP Extension Agent shared object file. This is accomplished through by adding a line to the snmpd.conf file:

dlmod ttsSNMPAgent <path>/ttsSNMPAgent.so

Where <path> is the full path to the extension agent, generally <install directory>/snmp. Be sure to restart the server before continuing.

For more information on the configuration of shared object extension agents with NET-SNMP, see the NET-SNMP tutorial.

3.6.5 TTS MIB files

TTS includes one or more MIB files for use with SNMP Management Consoles. These files are located in the “snmp/MIBs” subdirectory in the TTS Server directory

(created on installation.) The TTS MIBs define the set of variables that can be queried/modified via SNMP, along with a brief description of their purpose.

The following is a list of properties that can be accessed via SNMP:

System status – An indication of the current state of the TTS server

System uptime – The number of seconds since server was started

Wireline version – The version of the client-server speech-control protocol.

Engine version – The version number of the TTS server.

Databases loaded – A list of voice databases in memory.

Default Database – The database currently set for default use.

3.7 Using the Server Edition with Nuance

Nuance ASR users can integrate the Nuance 7.0.4 and Nuance 8.0 ASR engine and the Natural Voices TTS engine using the Nuance NuTTS interface. This feature is available only for the Server Edition on Windows and Solaris platforms. We will add new platforms as Nuance adds support for additional platforms.

NVIF_ATTNaturalVoices_704 and NVIF_ATTNaturalVoices_80 emulate the Nuance Vocalizer TTS engine and accepts synthesis requests from RecClient or a Nuance Resource Manager then hands those requests off to the AT&T Natural Voices TTS engine. This allows you to take full advantage of the Nuance Resource Manager and the AT&T Natural Voices TTS Engine. The AT&T Natural Voices Server must be running before starting NVIF_ATTNaturalVoices.

Synopsis:

`NVIF_ATTNaturalVoices_704 -s` *`NVServerNameOrIPAddress`* `-p` *`NVPort`* `–xml –channels` *`cnt NuanceArguments`*

or

`NVIF_ATTNaturalVoices_80 -s` *`NVServerNameOrIPAddress`* `-p` *`NVPort`* `–xml –channels` *`cnt NuanceArguments`*

where *NVServerNameOrIPAddress* is the server name or IP address of the server running the AT&T Natural Voices TTS Server, *NVPort* is the port number the TTS server is listening to for TTS requests, 7000 by default, *cnt* is the number of simultaneous channels (16 by default), and *`NuanceArguments`* are any additional arguments that should be passed along to Nuance.

NuanceArguments include:

- tts.port which specifies the port that NVIF_ATTNaturalVoices uses to listen for requests with a default value of 32323. This should **NOT** be the same port as specified with the -p option
- tts.ResourceName specifies the name of an instance of the TTS Server which allows clients to direct requests to a specific class of servers.
- rm.Addresses defines the name and port of one or more Nuance Resource Managers to which the TTSServer should attach. By default, no resource manager is used.

- watcher.DaemonDatagramPort contacts the Nuance Watcher on the specified port which defaults to 7890.

Please see the appropriate Nuance documentation for more information about the Nuance Resource Manager, RecClient, and relevant parameters.

Example:

$ NVIF_ATTNaturalVoices_704 -s localhost -p 7000 –xml rm.Addresses=localhost

Starts the NVIF_ATTNaturalVoices process which sendsTTS synthesis requests to the AT&T Natural Voices Server which is running on localhost, port 7000. NVIF_ATTNaturalVoices is controlled from the Nuance Resource Manager that is also running on localhost.

4 Running the Server-Lite and Desktop Edition

This section describes running the Server-Lite and Desktop Editions of the AT&T Natural Voices TTS engines and describes the sample applications that are included with Microsoft SAPI 4, SAPI 5.1, and with the AT&T Natural Voices TTS SDK. This chapter describes the command line interfaces to the applications and Chapter 5 details the APIs that are available for your application to use.

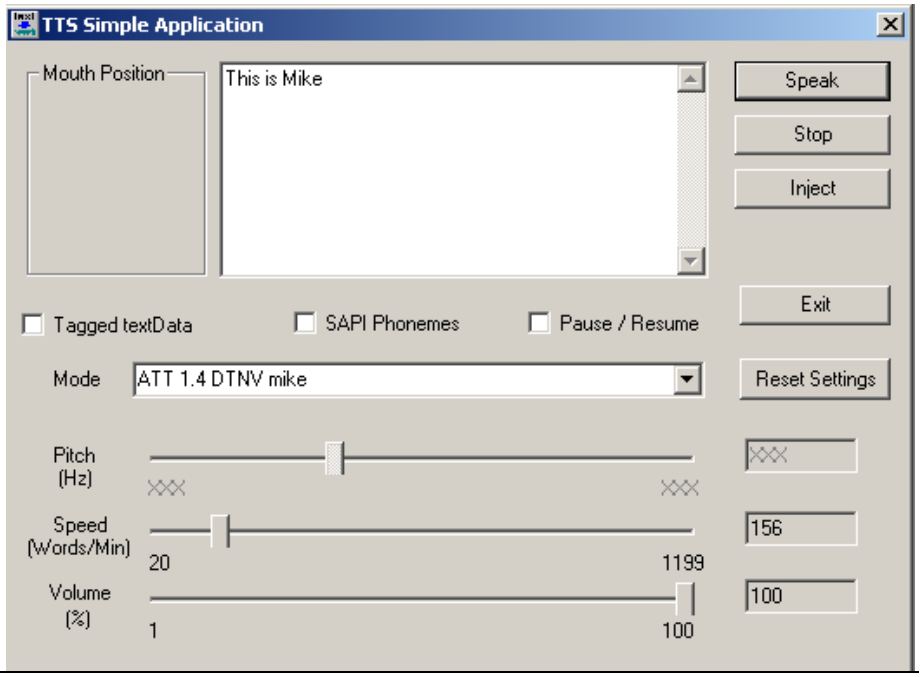
Microsoft Windows users have several choices when using the AT&T Natural Voices TTS Server-Lite or Desktop engines: you may write your application to the Microsoft SAPI 4 or SAPI 5.1 specification, you can invoke our C++ APIs, or you invoke the TTS engine from Java. Unix users must write to the C++ or Java APIs. If you write to the C++ APIs, you can choose either to link our TTS engine DLL or shared object with your application or make use of a TTS Proxy that starts and stops automatically.

4.1 Microsoft Windows SAPI 4.0 TTSApp

If you're running on Microsoft Windows 98, ME, NT, 2000, or XP and you've installed the Microsoft SAPI 4 SDK then you may want to use the TTSApp that is included with Microsoft SAPI 4. This GUI client makes it easy to synthesize text that you type in. You'll find the TTSApp at

```
C:\Program Files\Microsoft Speech SDK\TTS\ttsapp.exe
```

Choose one of the AT&T Natural Voices TTS voices, Mike, Crystal, Rich, Claire, Rosa, Klara, or Reiner from the "Mode" selector as shown below. You'll also need to check off the "Tagged textDate" box if you'd like to include SAPI 4 control tags.

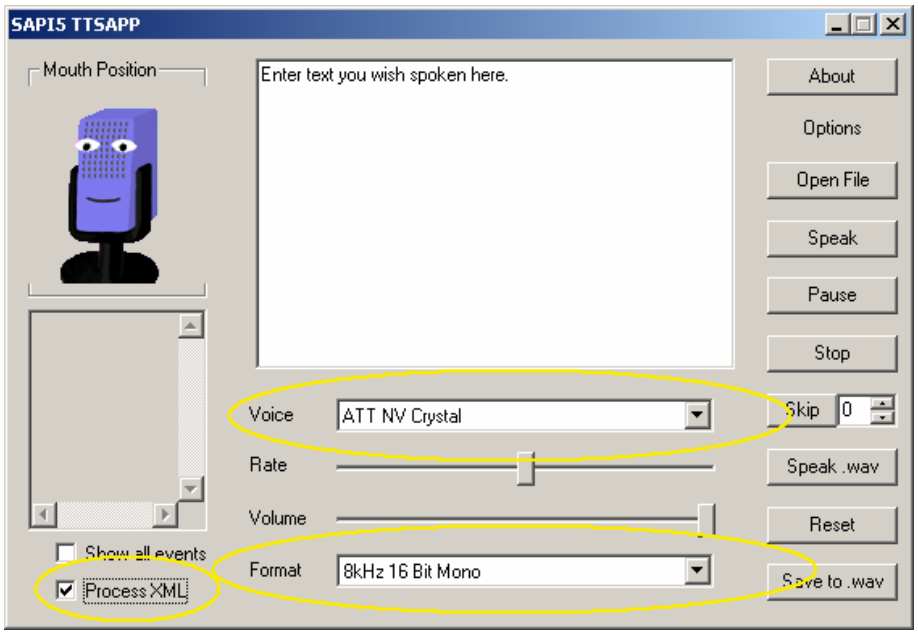


4.2 Microsoft Windows SAPI 5.1 TTSApp

If you've installed the Microsoft SAPI 5.1 SDK then you may want to use the TTSApp that is included with Microsoft SAPI 5.1. This GUI client makes it easy to synthesize text that you type in or load from a file. You'll find the TTSApp at

Start->Programs->Microsoft Speech SDK 5.1->Tools->TTSApp

Choose one of the AT&T Natural Voices TTS voices, either Mike or Crystal, from the "Voice" selector and set the format to "8kHz 16 bit mono" as shown below. You'll also need to check off the "Process XML" box if you'd like to include control tags.



4.3 Command Line TTS Applications

Four command line TTS clients, TTSDesktopPlayer, TTSDesktopFile, TTSStandalonePlayer, and TTSStandaloneFile are included in the bin directory of the AT&T Natural Voices TTS Server-Lite or Desktop directory. TTSDeskTopPlayer and TTSStandalonePlayer allow you to synthesize text files and play them through a sound card. TTSDesktopFile and TTSStandaloneFile synthesize text files and write the linear 16 bit PCM output to a file. The "Standalone" applications link the TTS library into the executable and allow at most one instance of a TTS engine per process. The "Desktop" applications allow you to start multiple TTS engines within a single process, however all synthesis requests are serialized by the Desktop Edition[4]. The "Desktop" applications link to a small library and use a TTS Proxy that is managed by the SDK to allow a single instance of the application to invoke.

	Output to Sound Card	**Output to File**
One TTS engine instance per process	TTSStandalonePlayer	TTSStandaloneFile
Multiple TTS engine instances per process	TTSDesktopPlayer	TTSDesktopFile

[4] The AT&T Natural Voices Desktop Edition allows you to create multiple engines but the requests for synthesis are serialized, effectively providing a single TTS channel. The Server-Lite Edition allows you to create multiple simultaneous engines and supports multiple simultaneous calls to Speak().

These applications are useful for testing the TTS engine. We provide source code for these four applications to help you understand how to use the TTS engine in your applications.

NOTE: The TTS Engine must be able to find data files that are included with the product to do synthesis. The path to these files is passed to the engine in one of four ways:

1. Both Microsoft Windows and Unix platforms can use an environmental variable called `NaturalVoicesPath` to specify the root of the directory tree. For example, on Windows the variable should be set to `c:\program files\attnaturalvoices\tts1.4\desktop` or `c:\program files\attnaturalvoices\tts1.4\serverlite` if the engine is installed in the default location or `/usr/local/attnaturalvoices/tts1.4/desktop` or `/usr/local/attnaturalvoices/tts1.4/desktop` on Unix systems.
2. If the variable is not set, Microsoft Windows platforms check the Registry for `HKEY_LOCAL_MACHINE\SOFTWARE\ATT\NaturalVoices\1.4\desktop` or `HKEY_LOCAL_MACHINE\SOFTWARE\ATT\NaturalVoices\1.4\serverlite` and retrieve the value of the string "`Pathdir`". This Registry key is set by the installation script and doesn't require change.
3. The sample applications allow you to specify the Root directory and the data subdirectory on the command line. See below for details.
4. You may specify the directory when you create in instance of the engine using the C++ classes. See Section 5.3.1 for more details.

The `NaturalVoicesPath` environmental variable is optional if you're writing applications for Microsoft Windows platforms since the applications will find the path in the registry. The environmental variable should be set for Linux unless your application specifies the path. If it is not set, the engine will fail to initialize. You can override the environmental variable using command line arguments to specify the root and bin directories as shown below.

All four commands accept the same command line arguments.

Synopsis:

```
TTSStandalonePlayer
     [-xml] [-f inputFile] [-l fileList]
     [-du userDictionary] [-da applicationDictionary]
     [-phoneset att_darpa_english | att_sampa_spanish |
        att_sampa_german | att_sampa_ukenglish |
        att_sampa_french ]
     [-mulaw] [-alaw]
     [-data dataPath] [-root rootPath]
     [-x voice]

TTSDesktopPlayer
     [-xml] [-f inputFile] [-l fileList]
     [-du userDictionary] [-da applicationDictionary]
     [-phoneset att_darpa_english | att_sampa_spanish |
        att_sampa_german | att_sampa_ukenglish |
        att_sampa_french]
     [-mulaw] [-alaw]
     [-data dataPath] [-root rootPath]
     [-x voice]

TTSStandaloneFile
     [-xml] [-f inputFile] [-l fileList]
     [-du userDictionary] [-da applicationDictionary]
     [-phoneset att_darpa_english | att_sampa_spanish |
        att_sampa_german | att_sampa_ukenglish |
        att_sampa_french]
     [-o audioOutputFile]
     [-mulaw] [-alaw]
     [-data dataPath] [-root rootPath]
     [-x voice]

TTSDesktopFile
     [-a] [-xml] [-f inputFile] [-l fileList]
     [-du userDictionary] [-da applicationDictionary]
     [-phoneset att_darpa_english | att_sampa_spanish |
        att_sampa_german | att_sampa_ukenglish |
        att_sampa_french ]
     [-o audioOutputFile]
     [-mulaw] [-alaw]
     [-data dataPath] [-root rootPath]
     [-x voice]
```

Options:

Option	Argument	Action
-xml	Parse XML tags as SAPI or SSML control tags	Default is not to parse XML tags.
-f	Input file name	The path of the text file to synthesize.
-l	File with a list of files to synthesize	Specify a file with a list of text files to synthesize.
-du	User dictionary file	Specify a user dictionary file.
-da	Application dictionary file	Specify an application dictionary file.

-phoneset	`att_darpabet_english`
	`att_sampa_spanish`
	`att_sampa_german`
		Specify the set of phonemes that are used in the dictionary.

-mulaw	none	Generate mulaw audio output
-alaw	none	Generate alaw audio output
-data	Path name	The directory path of the data subdirectory, typically c:\program files\attnaturalvoices\tts1.4\desktop\data or /usr/local/attnaturalvoices/tts1.4/desktop/data
-root	Path name	The directory path of the bin subdirectory of the Natural Voices directory, typically c:\program files\attnaturalvoices\tts1.4\desktop\bin or /usr/local/attnaturalvoices/tts1.4/desktop/bin
-x	mike, crystal, or other installed voice	Specify the default voice
-o	AudioOutputFile (TTSStandaloneFile and TTSDesktopFile only)	Specify the output file where the audio will be written.

UNIX Example:

```
NaturalVoices=/usr/local/ATTNaturalVoices/TTS1.4/desktop
Export NaturalVoices
TTSStandalonePlayer –f sample.txt
```

synthesizes sample.txt and writes the audio to a sound card.

```
TTSClientFile –f sample.txt > audio.wav
play –t wav audio.wav
```

synthesizes the text in sample.txt, capturing the audio output to audio.wav then uses the UNIX play command to play the audio.

Windows Example:

```
C:\program files\ATTNaturalVoices\TTS1.4\Desktop\bin\TTSDesktopPlayer
       -f sample.txt
```

reads the contents of sample.txt and plays the synthesized audio on the sound card.

```
C:\program files\ATTNaturalVoices\TTS1.4\Desktop\bin\TTSDesktopFile
   -f sample.txt -o audio.wav
```

synthesizes the text in sample.txt, capturing the audio output to audio.wav. Double click on audio.wav to play the audio.

You'll find the source code for all of these applications in the `samples` subdirectory.

5 Client C++ SDK

The AT&T Natural Voices TTS package includes a Software Developers Kit (SDK) for building client applications that work with the TTS server. The SDK contains C++ class definitions, libraries and sample applications that demonstrate the use of the SDK.

The Server, Server-Lite, and Desktop Editions of the AT&T Natural Voices TTS engine all use the same SDK. The only difference between using the SDK with the three Editions is in selecting the engine model when creating the engine. We'll explain the differences below.

5.1 Getting Started

The SDK contains the necessary header and library files that should be included in an application's build settings.

5.1.1 SDK Headers

There is only one header required for inclusion in a speech synthesis application. The header file TTSApi.h contains all the necessary C++ classes. Alternatively, an application can include an operating system-specific header, TTSWin32API.h for Windows platforms or TTSUnixAPI.h for UNIX platforms, which contains some additional classes that may be useful for building applications. TTSWin32API.h and TTSUnixAPI include the header file TTSApi.h.

Header	***Description***
TTSApi.h	Contains all the necessary C++ classes, definitions and functions. Also includes the headers TTSResult.h and TTSOSMacros.h.
TTSWin32API.h	Contains some optional Windows-specific C++ classes, headers and definitions. Includes TTSApi.h
TTSUnixAPI.h	Contains some optional Unix-specific C++ classes, headers and definitions. Includes TTSApi.h
TTSResult.h	This file contains the result codes that are returned from the SDK C++ classes and functions.
TTSOSMacros.h	This file contains operating system-specific macros necessary to provide thread-safe access to the SDK C++ classes.
TTSUTF8.h	This file contains string functions that handle UTF-8 strings.

5.1.2 Compiler Definitions

The following preprocessor definitions should be used when integrating the SDK with an application.

Operating System	***Definitions***
Windows	WIN32
Unix	UNIX

5.1.3 SDK Libraries

The SDK contains one library with which an application should link. The developer should also use an appropriate threading library.

Operating System	*Library*

Windows	The following libraries are built with Visual C++ v6.0 SP3 and are available for applications to use. They are located in the lib sub-directory. All versions are compiled using the Multithreaded run-time libraries, both DLL and static versions.
	lib/Debug client/server libraries:
	`TTSSDKLibraryCS.lib – ANSI compiled with /MTd`
	`TTSSDKLibraryCSU.lib – UNICODE compiled with /MTd`
	`TTSSDKLibraryCSMD.lib – ANSI compiled with /MDd`
	`TTSSDKLibraryCSUMD.lib – UNICODE compiled with /MDd`
	lib/Release client/server libraries:
	`TTSSDKLibraryCS.lib – ANSI compiled with /MT`
	`TTSSDKLibraryCSU.lib – UNICODE compiled with /MT`
	`TTSSDKLibraryCSMD.lib – ANSI compiled with /MD`
	`TTSSDKLibraryCSUMD.lib – UNICODE compiled with /MD`
	lib/Debug standalone libraries:
	`TTSSDKLibrarySA.lib – ANSI compiled with /MTd`
	`TTSSDKLibrarySAU.lib – UNICODE compiled with /MTd`
	`TTSSDKLibrarySAMD.lib – ANSI compiled with /MDd`
	`TTSSDKLibrarySAUMD.lib – UNICODE compiled with /MDd`
	lib/Release standalone libraries:
	`TTSSDKLibrarySA.lib – ANSI compiled with /MT`
	`TTSSDKLibrarySAU.lib – UNICODE compiled with`

	/MT TTSSDKLibrarySAMD.lib – ANSI compiled with /MD TTSSDKLibrarySAUMD.lib – UNICODE compiled with /MD
Unix	libTTSAPI.a – this is built with the GNU g++ compiler and uses pthread functionality. An application must also link with libpthread.a.

5.2 Programming Conventions

The SDK uses the following conventions with which a developer should become familiar before using the SDK.

5.2.1 Return Codes

The header TTSResult.h defines a result code type called `TTS_RESULT` and all the possible values it may contain. Most of the C++ global and class member functions return a `TTS_RESULT` code. An application may use the macros `FAILED()` and `SUCCEEDED()` to determine if a function failed or succeeded. There is also a static class `CTTSResult` that will return a descriptive English-language string for the result code. For example:

```
TTS_RESULT result = ttsCreateEngine(...);
if (FAILED(result)) {
        cout << CTTSResult::GetErrorString(result);
}
```

5.2.2 CTTSRefCntObject

All of the SDK C++ classes derive from the `CTTSRefCntObject` class. This class is used to provide reference counting for classes and manage pointers to the classes.

When a class object is created, its reference count is zero. An application should call `CTTSRefCntObject::AddRef` for every pointer to a class object it needs to obtain. `CTTSRefCntObject::AddRef` increases the reference count of the class by one.

An application should call `CTTSRefCntObject::Release` when it is finished with a pointer to a class object. `CTTSRefCntObject::Release` decreases the reference count of the object by one. When the reference count is zero, the object will free itself from memory.

These calls are symmetrical, for every `CTTSRefCntObject::AddRef` there should be a `CTTSRefCntObject::Release`.

5.2.3 UTF-8 Strings

The SDK supports only UTF-8 strings as both input and output. (UTF-8 is a multibyte character encoding. its major advantage is that it allows encoding of 7-bit US-ASCII characters in a single byte, thus allowing existing US-ASCII text data to be read as UTF-8 without modification.) The application should be conscious of functions within

the SDK that receive or return text in UTF-8 format. The header TTSApi.h defines some type definitions for UTF-8 strings.

```
typedef unsigned char                UTF8String;
typedef unsigned char *              PUTF8String;
typedef const unsigned char *        PCUTF8String;
```

5.2.4 C++ Standard Template Library

Some the TTS SDK class functions use C++ Standard Template Library (STL) list, vector, and string container classes. Developers should become familiar with using STL classes.

5.3 Creating An Engine

The SDK is the same for the Server, Server-Lite, and Desktop Editions of the AT&T Natural Voices TTS engines. The only difference in the SDK between three Editions is the engine model that is specified during engine configuration.

Creating an engine involves building a `TTSConfig` structure and calling the library function `ttsCreateEngine()`.

5.3.1 Build A Configuration Structure

The `TTSConfig` structure contains the information necessary for the SDK library to create a `CTTSEngine` object.

TTSConfig	***Description***
m_eEngineModel	Indicates which engine model to create: TTSENGINEMODEL_CLIENTSERVER – a client/server engine model. The SDK will determine what protocol the server is running. Supported only in the AT&T Natural Voices TTS Server Edition. TTSENGINEMODEL_CLIENTSERVER_PRERELEASE – a client/server engine model. The server is a pre-release of Natural Voices. Supported only in the AT&T Natural Voices TTS Server Edition. TTSENGINEMODEL_CLIENTSERVER _NV10 – a client/server engine model. The server is Natural Voices engine v1.0. Supported only in the AT&T Natural Voices TTS Server Edition. TTSENGINEMODEL_CLIENTSERVER CURRENT – a client/server engine model. The server is the current Natural Voices engine. Supported only in the AT&T Natural Voices TTS Server Edition. TTSENGINEMODEL_STANDALONE – a standalone engine model. This supports only one instance of the Natural Voices engine per process. Supported only in the AT&T Natural Voices TTS Server-Lite and Desktop Editions. TTSENGINEMODEL_DESKTOP – a desktop engine model which allows a single process to create multiple instances of the TTS engine. Supported only in the AT&T Natural Voices TTS Server-Lite and Desktop Editions.

m_eEngineBehavior	Determines the behavior of the engine: `TTSENGINE_SYNCHRONOUS` – the behavior is the same for all function calls as in the asynchronous mode except the `CTTSEngine::Speak()` call which will not return until all the audio and other notifications have been sent to the application's `CTTSSink` class. `TTSENGINE_ASYNCHRONOUS` – the engine will return after fully completing a function call. Most affected is the `CTTSEngine::Speak()` call which will return before all audio and other notifications have been sent to the applications `CTTSSink` class. `TTSENGINE_MESSAGEBASED` – the engine will return immediately from all function calls. The application should rely on the `CTTSSink` notification to determine if the call was successful or not.
m_lstServerConfig	This is an STL list of `TTSServerConfig` structures. This is applicable only to `TTSENGINEMODEL_CLIENTSERVER_*` engine mode with the AT&T Natural Voices TTS Server Edition. An application may have multiple TTS servers try to connect to during a `CTTSEngine::Initialize()` function call.
m_strRootDirectory	This field is only applicable to the `TTSENGINEMODEL_DESKTOP` engine model. It overrides the path to the installation of AT&T Natural Voices. By default, the value of the `NaturalVoicesPath` environmental variable is used if defined. Windows platforms next checks the registry key `HKEY_LOCAL_MACHINE\SOFTWARE\ATT\NaturalVoices\1.4\desktop\Pathdir with the Desktop Edition` or `HKEY_LOCAL_MACHINE\SOFTWARE\ATT\NaturalVoices\1.4\ServerLite\Pathdir`. The Pathdir value in the registry does not include the \bin subdirectory but m_strRootDirectory must specify the full directory name, e.g. `c:\program files\attnaturalvoices\tts1.4\Desktop\bin`, `c:\program files\attnaturalvoices\tts1.4\ServerLite\bin`, or `/usr/local/bin/attnaturalvoices/tts1.4/ServerLite/bin`.
m_strDataDirectory	This field is only applicable to the `TTSENGINEMODEL_DESKTOP` and `TTSENGINEMODEL_STANDALONE` engine models. It is an override to where the voice data is located.
m_strLogDirectory	This is an optional field for the desktop version to specify a log file for the desktop proxy server.
m_strTempDirectory	This is an optional field that tells the SDK where to store temporary files.

The `TTSServerConfig` structure is a description of a server, port and timeout that the client/server engine will use to connect to a TTS server. This structure is appropriate only for the `TTSENGINEMODEL_CLIENTSERVER`, `TTSENGINEMODEL_CLIENTSERVER_PRERELEASE`, `TTSENGINEMODEL_CLIENTSERVER _NV10`, and `TTSENGINEMODEL_CLIENTSERVER CURRENT` engine models.

TTSServerConfig	***Description***
m_nServerPort	The TTS server engine port to which the SDK should connect.
m_tvTimeout	The timeout value the SDK should use while communicating with a TTS server. If an application sets the timeval to {0, 0} the SDK will use an appropriate default.
m_szServer	The TTS server to which the SDK should connect. It can either be a DNS name or an explicit IP address.

All the engine modes and behaviors are multi-thread safe. The `TTSENGINE_ASYNCHRONOUS` engine is for applications that need to insure that function calls return the result from the TTS server before returning control to the application. In addition, they rely on separate threads in the SDK to process notification and audio data from the TTS server. The `TTSENGINE_SYNCHRONOUS` engine is for applications that do not want extra threads running in the system and need `CTTSEngine::Speak()` calls to run to completion before returning. The `TTSENGINE_MESSAGEBASED` engine is for applications that run in a state machine and rely on the asynchronous notifications in the `CTTSSink` to drive the application.

5.3.2 Creating an Engine

The process for creating an engine is slightly different for the AT&T Natural Voices TTS Server Edition than for the Server-Lite and Desktop Editions.

5.3.3 Creating an Engine for the Server Edition

Invoking the `ttsCreateEngine()` function will result in a `CTTSEngine` pointer being returned to the application. If the call is successful, the application should `AddRef()` the returned `CTTSEngine` pointer. When the application no longer needs the `CTTSEngine` pointer, it should call its `Release()` member function.

The format of the `ttsCreateEngine()` function is:

```
TTS_RESULT ttsCreateEngine(CTTSEngine **ppEngine,
                                        const TTSConfig &ttsConfig);
CTTSEngine *pEngine = 0;
TTSConfig ttsConfig;
TTSServerConfig ttsServerConfig;
//
// Setup our server configuration
//
ttsServerConfig.m_nServerPort = nPort;
ttsServerConfig.m_szServer = pszServer;
```

```
ttsServerConfig.m_tvTimeout.tv_sec = 15;
ttsServerConfig.m_tvTimeout.tv_usec = 0;
//
// Setup our configuration
//
ttsConfig.m_eEngineModel = TTSENGINEMODEL_CLIENTSERVER;
ttsConfig.m_eEngineBehavior = TTSENGINE_SYNCHRONOUS;
ttsConfig.push_back(ttsServerConfig);
//
// Create the engine
//
result = ttsCreateEngine(&pEngine, ttsConfig);
//
// Success?
//
if (result == TTS_OK && this->m_pEngine) {
        //
        // AddRef() the engine
        //
        pEngine->AddRef();
        //
        // application continues
        //
        . . . .
        //
        // Release the engine
        //
        pEngine->Release();
}
```

5.3.4 Creating an Engine for the Server-Lite and Desktop Editions

Invoking the `ttsCreateEngine()` function will result in a `CTTSEngine` pointer being returned to the application. If the call is successful, the application should `AddRef()` the returned `CTTSEngine` pointer. When the application no longer needs the `CTTSEngine` pointer, it should call its `Release()` member function.

The format of the `ttsCreateEngine()` function is:

```
TTS_RESULT ttsCreateEngine(CTTSEngine **ppEngine,
                           const TTSConfig &ttsConfig);
```

Two different types of TTS engines can by created from the Server-Lite and Desktop SDK: a standalone engine, and a desktop engine. The standalone engine runs within the process space of the application that called `ttsCreateEngine()`. It is specified by setting the `m_eEngineModel` member of the `TTSConfig` structure to `TTSENGINEMODEL_STANDALONE`, and linking against the `TTSSDKLibrarySA.lib` library. The desktop engine runs in a separate process. The process that calls `ttsCreateEngine()` communicates with the new process over a local network connection. It is specified by setting the `m_eEngineModel` member to `TTSENGINEMODEL_DESKTOP` and linking against the `TTSSDKLibaryCS.lib` library.

The decision to use the desktop vs. standalone configuration depends on how many instances of the engine are created by the application. The standalone version is limited to a single engine instance per process. An attempt to call `ttsCreateEngine()` for a second standalone engine will fail, unless the first engine has been released. The standalone engine will perform slightly better, however, because it does not incur the overhead of inter-process communication. If the application needs to have more than one instance of the engine active at a time, you must use the desktop model.

Below is sample code for creating a desktop engine. To create a standalone engine, replace `TTSENGINEMODEL_DESKTOP` with `TTSENGINEMODEL_STANDALONE`.

```
CTTSEngine *pEngine = 0;
TTSConfig ttsConfig;
//
// Setup our configuration
//
ttsConfig.m_eEngineModel = TTSENGINEMODEL_DESKTOP;
ttsConfig.m_eEngineBehavior = TTSENGINE_SYNCHRONOUS;
//
// Create the engine
//
result = ttsCreateEngine(&pEngine, ttsConfig);
//
// Success?
//
if (result == TTS_OK && this->m_pEngine) {
        //
        // AddRef() the engine
        //
        pEngine->AddRef();
        //
        // application continues
        //
        . . . .
        //
        // Release the engine
        //
        pEngine->Release();
}
```

5.4 Receiving Engine Notifications And Messages

5.4.1 CTTSSink Notifications

An application should implement a `CTTSSink`-derived class and send a pointer to it to the `CTTSEngine::SetSink()` function. A `CTTSSink` object receives TTS notifications such as audio, bookmarks, word information, and phoneme information. When a `CTTSEngine::Speak()` function is called, the `CTTSEngine` object will send `CTTSNotification` objects to the application through its `CTTSSink` pointer.

```
class   CSimpleEngine : public CTTSSink
{
public:
        CSimpleEngine(void)
        :
        CTTSSink()
        {
        }
        virtual TTS_RESULT onNotification(CTTSNotification *pNotification)
        {
                switch (pNotification->Notification()) {
                        ……
                }
                return TTS_OK;
        }
protected:
        virtual ~CSimpleEngine(void)
        {
        }
};
```

An application tells the SDK which notifications it is interested in receiving by using the `CTTSEngine::SetNotifications()` function.

TTSNOTIFY_* definition	***Description***
TTSNOTIFY_STARTED	A Speak has started
TTSNOTIFY_NOTIFYCHANGED	A notification value has changed
TTSNOTIFY_VOICECHANGED	The voice has changed
TTSNOTIFY_AUDIOFORMATCHANGED	The audio format has changed
TTSNOTIFY_FINISHED	A Speak has finished
TTSNOTIFY_AUDIO	Audio data notification
TTSNOTIFY_WORD	Word notification
TTSNOTIFY_PHONEME	Phoneme notification
TTSNOTIFY_VISEME	Viseme notification
TTSNOTIFY_BOOKMARK	Bookmark notification
TTSNOTIFY_VOLUME	Volume value change
TTSNOTIFY_RATE	Speaking rate change
TTSNOTIFY_PITCH	Pitch change

When performing a `CTTSEngine::SetNotifications()` function call, the application specifies which notifications it is interested in modifying and whether the notification should be turned on or off. For example, the following code tells the engine that the application wishes to turn on audio and end-of-speech notifications, and to turn off phoneme notifications:

```
//
// This turns audio and finished notifications on and turns
// phoneme notifications off.
//
TTS_RESULT result = pEngine->SetNotifications(
               TTSNOTIFY_AUDIO | TTSNOTIFY_FINISHED | TTSNOTIFY_PHONEME,
               TTSNOTIFY_AUDIO | TTSNOTIFY_FINISHED);
```

Turning off phoneme and viseme notifications can make a dramatic performance improvement because the events are not generated by the server and then processed by the client. If your application does not require phonemes or visemes, you can disable them with the following code:

```
//
// This turns audio notifications on and turns
// phoneme and viseme notifications off.
//
TTS_RESULT result = pEngine->SetNotifications(
               TTSNOTIFY_AUDIO | TTSNOTIFY_PHONEME | TTSNOTIFY_VISEME,
               TTSNOTIFY_AUDIO);
```

5.4.2 CTTSNotification Object

The `CTTSNotification` object contains all the information for any type of notification. To determine the type of the notification, the application should call the `CTTSNotification::Notification()` function which returns a `TTSNOTIFY_*` code.

```
TTS_RESULT
CmySink::onNotification(CTTSNotification *pNotification)
{
        switch (pNotification->Notification()) {
        case TTSNOTIFY_AUDIO: {
                long lLength;
                if (pNotification->AudioData(&pAudioData, &lLength) ==
                        TTS_OK) {
                        . . play the audio . .
                }
                break;
        }
        case TTSNOTIFY_FINISHED:
                . . signal that it is finished with a speak call. .
                break;
        case TTSNOTIFY_WORD: {
                TTSWordNotification wordNotification;
                if (pNotification->Word(&wordNotification) == TTS_OK) {
                        cout << "WORD: ";
                        cout << wordNotification.m_lCharacterOffset;
                }
                break;
        }
        case TTSNOTIFY_BOOKMARK: {
                PCUTF8String szBookmark = 0;
                if (pNotification->Bookmark(&szBookmark) == TTS_OK &&
                        szBookmark) {
                        cout << "BOOKMARK: " << szBookmark << "\n";
                }
                break;
        }
        case TTSNOTIFY_PHONEME: {
                TTSPhonemeNotification phoneme;
                if (pNotification->Phoneme(&phoneme) == TTS_OK) {
                        vector<PCUTF8String>::const_iterator iPhonemes
                                = phoneme.m_vPhonemeIDs.begin();
                        while (iPhonemes != phoneme.m_vPhonemeIDs.end()) {
                                cout << *iPhonemes++;
                        }
                        cout << " " << phoneme.m_lDuration;
                        cout << " " << phoneme.m_lStressLevel;
                }
                break;
        }
        return TTS_OK;
}
```

5.4.3 TTSWordNotification structure

The `TTSWordNotification` structure contains all the information for a word notification.

TTSWordNotification members	***Description***
m_lCharacterOffset	Offset in bytes of the word from the beginning of text input.
m_lWordLength	The length of the word in bytes.
m_lSentenceLength	If this is the first word of a sentence, this is the length of the sentence in bytes.
m_lWordFlags	A bit-field of flags indicating what the word is: TTSWORDFLAG_SENTENCE TTSWORDFLAG_PARAGRAPH TTSWORDFLAG_NOUNPHRASE
m_ucPartOfSpeech	The part of speech of the word

5.4.4 TTSPhonemeNotification structure

The `TTSPhonemeNotification` structure contains all the information for a phoneme notification.

TTSPhonemeNotification members	***Description***
m_vPhonemeIDs	This is a vector containing one or more DARPA phoneme strings.
m_lDuration	The duration of the phoneme string in milliseconds.
m_lStressLevel	The stress level of the phoneme string. A value between 1 and 3.

5.4.5 CTTSErrorInfoObject

An application may *optionally* implement a `CTTSErrorInfoObject`-derived class and send a pointer to it to the `CTTSEngine::SetErrorInfoObject()` function. A `CTTSErrorInfoObject` receives error, information, and trace messages from the SDK classes and can be useful for debugging and logging information.

```
class   CSimpleEngine : public CTTSErrorInfoObject
{
public:
        CSimpleEngine(void)
        :
        CTTSErrorInfoObject()
        {
        }
        virtual TTS_RESULT    onErrorMessage(TTS_RESULT error,
                                        const char *pszErrorMessage)
        {
                try {
                        cout << pszErrorMessage << "\n";
                        cout << CTTSResult::GetErrorString(error) << "\n";
                }
                catch(...) {
                }
                return TTS_OK;
        }
        virtual TTS_RESULT    onInformationMessage(
                                        const char *pszInformationMessage)
        {
                try {
                        cout << pszInformationMessage << "\n";
                }
                catch(...) {
                }
                return TTS_OK;
        }
        virtual TTS_RESULT    onTraceMessage(int nTraceLevel,
                                        const char *pszTraceMessage)
        {
                try {
                        cout << pszTraceMessage << "\n";
                }
                catch(...) {
                }
                return TTS_OK;
        }
protected:
        virtual ~CSimpleEngine(void)
        {
        }
};
```

5.5 Initializing And Shutting Down The Engine

After creating a `CTTSEngine` object, the application must call the `CTTSEngine::Initialize()` function. If the `CTTSEngine::Initialize()` call returns a successful status code, then the application must call `CTTSEngine::Shutdown()` when it is finished with the `CTTSEngine` object.

These calls must be symmetrical, for every `CTTSEngine::Initialize()` there should be a `CTTSEngine::Shutdown()`.

```
//
// Initialize the engine
//
TTS_RESULT result = pEngine->Initialize();
if (SUCCEEDED(result)) {
        . . . speak etc.

        //
        // we are done, so shut down the engine
        //
        pEngine->Shutdown();
}
```

5.6 Setting The Voice

5.6.1 TTSVoice Structure

An application uses a `TTSVoice` structure to enumerate, set or query the current voice the engine is using.

TTSVoice member	***Description***
m_nGender	Indicates the gender. The possible values: TTSGENDER_DEFAULT – use the default gender TTSGENDER_MALE – male voice TTSGENDER_FEMALE – female voice
m_nAge	Indicates the TTSAge enumeration. The possible values are: TTSAGE_CHILD – child voice TTSAGE_TEEN – teenage voice TTSAGE_ADULT – adult voice TTSAGE_SENIOR – senior voice
m_szName	The name of the voice. If this is “”, then the default voice will be used.
m_szLocale	This is the language locale for the voice. If this is “”, then the default locale will be used. The locale names used in the TTSVoice structure are described in the UNIX setlocale(3C) documentation. The 1.4 engine supports US English, whose locale name is “en_us” and US Spanish with locale name “es_us”.
m**_nSampleRate**	This the sample rate of the voice, either 8000Hz or 16000Hz. Both are 16 bit PCM.

5.6.2 Enumerating The Voices

An application can find out how many voices are available by calling the `CTTSEngine::NumVoices` function and can then enumerate their characteristics using the `CTTSEngine::EnumVoice` function.

```
//
// How many voices are there?
//
int nVoices = 0;
TTS_RESULT result = pEngine->NumVoices(&nVoices);
if (SUCCEEDED(result) && nVoices) {
        for (int i = 0; i < nVoices; i++) {
                TTSVoice voice;
                result = pEngine->EnumVoice(i, &voice);
                if (SUCCEEDED(result)) {
                        cout << voice.m_szName;
                }
```

```
        }
}
```

5.6.3 Setting The Voice

An application can set the voice using the `CTTSEngine::SetVoice` function. The application should be aware that the returned voice may not be exactly what the application desired. The SDK will select the closest supported voice for the supplied `TTSVoice` structure. The application may check the returned `TTSVoice` structure to determine the characteristics of the actual voice selected by the SDK.

```
//
// Set the male voice
//
TTSVoice voice, voiceReturned;
voice.m_nGender = TTSGENDER_MALE;
TTS_RESULT result = pEngine->SetVoice(&voice, &voiceReturned);
if (SUCCEEDED(result)) {
        //
        // Did it really change it?
        //
        if (voiceReturned.m_nGender == TTSGENDER_MALE) {
                // Yes!
        }
}
```

5.7 Setting The Audio Format

5.7.1 TTSAudioFormat Structure

An application uses a `TTSAudioFormat` structure to enumerate, set or query the current audio format the engine is using.

TTSAudioFormat member	***Description***
m_nType	Indicates the type of audio format. Possible values are: TTSAUDIOTYPE_DEFAULT – the default format. For the current server, the default format is TTSAUDIOTYPE_LINEAR. TTSAUDIOTYPE_MULAW - this is CCITT g.711 mulaw TTSAUDIOTYPE_LINEAR – this is linear PCM TTSAUDIOTYPE_ALAW - this is CCITT g.711 alaw
m_nSampleRate	This is the sample rate. Set it to 0 to use the default sample rate in samples per second for the selected audio type. This will be either 8000 or 16000.
m_nBits	This is the number of bits per sample. Set it to 0 to use the default number of bits. For the current server, this will either be 8 or 16 bits.
m_nReserved	This is reserved field, please do not use it.

5.7.2 Enumerating The Audio Formats

An application can find out how many audio formats there are by calling the `CTTSEngine::NumAudioFormats` function and can then enumerate their characteristics using the `CTTSEngine::EnumAudioFormat` function.

```
//
// How many audio formats are there?
//
int nFormats = 0;
TTS_RESULT result = pEngine->NumAudioFormats(&nFormats);
if (SUCCEEDED(result) && nFormats) {
        for (int i = 0; i < nFormats; i++) {
                TTSAudioFormat format;
                result = pEngine->EnumAudioFormat(i, &format);
                if (SUCCEEDED(result)) {
                        cout << format.m_nType;
                }
        }
}
```

5.7.3 Setting the Audio Format

An application can set the audio format using the `CTTSEngine::SetAudioFormat` function. The application should be aware that the returned audio format may not be exactly what the application desired. The SDK will select the closest supported audio format for the TTSAudioFormat structure supplied by the application. The application may check the returned `TTSAudioFormat` structure to determine the characteristics of the actual audio format selected by the SDK.

```
//
// Set to linear PCM
//
TTSAudioFormat format, formatReturned;
format.m_nType = TTSAUDIOTYPE_LINEAR;
TTS_RESULT result = pEngine->SetAudioFormat(&format, &formatReturned);
if (SUCCEEDED(result)) {
        //
        // Did it really change it?
        //
        if (formatReturned.m_nType == TTSAUDIOTYPE_LINEAR) {
                // Yes!
        }
}
```

5.8 Speaking Text

An application speaks text using the CTTSEngine::Speak() function. If the application has created a TTSENGINE_SYNCHRONOUS engine then the CTTSEngine::Speak() call will not return until all the audio and other notifications have been sent to the CTTSSink object. If the application has created a TTSENGINE_ASYNCHRONOUS engine then the CTTSEngine::Speak() call will return immediately. Applications using the asynchronous model optionally get a TTSSPEAKHANDLE when making a CTTSEngine::Speak call which can be used to associate Notifications with a particular asynchronous Speak call.

An application can queue multiple speak requests by calling the `CTTSEngine::Speak()` function multiple times.

5.8.1 Speaking Text

The SDK requires UTF-8 character input for internationalization. The SDK supports the Microsoft SAPI 4, Microsoft SAPI 5.1, the SSML, and JSML text markup languages. The application uses the `CTTSEngine::SpeakFlags` argument of the `CTTSEngine::Speak()` call to tell the SDK what kind of text the application is sending.

CTTSEngine::Speak Flags	***Description***
sf_default	Default text string with no tags or XML information.
sf_sapixml	SAPI, SSML, or JSML formatted text.

```
//
// Speak some text
//
string szText = "hello!";
TTS_RESULT result = pEngine->Speak((PUTF8String) szText.c_str(),
                                   CTTSEngine::sf_default);
if (FAILED(result)) {
        cout << CTTSResult::GetErrorString(result);
}
```

5.8.2 Speaking CTTSTextFragment objects

The SDK also supports speaking `CTTSTextFragment` objects. The application implements a `CTTSTextFragment` derived object and passes a list of `CTTSTextFragment` objects to the SDK through the text fragment version of the `CTTSEngine::Speak(const list<CTTSTextFragment *> &lstFragments)` call. When passed a list of fragments, the SDK will interpret the `CTTSTextFragment` object and call its functions to retrieve information pertaining to the fragment.

A `CTTSTextFragment` object describes a fragment of text. Each fragment has an action which determines how to interpret the data within the fragment.

CTTSTextFragment::Actions	***Description***	
a_speak	The fragment contains text to speak. The member functions are interpreted as follows:	
	`Text()`	The text to speak.
	`TextLength()`	The length of the text.
	`TextOffset()`	The offset of the text from the beginning of the original text buffer.
	`PartOfSpeech()`	The part of speech for the text.
	`EmphasisAdjustment()`	Is the text emphasized or not?
	`RateAdjustment()`	Speech rate adjustment for the text above or below normal rate.
	`VolumeAdjustment()`	Speech volume adjustment for the text above or below normal rate.
	`PitchAdjustment()`	Speech pitch adjustment for the text above or below the normal pitch.
	`Context()`	Context flags for interpreting the text.
a_silence	The fragment contains silence. The member functions are interpreted as follows:	
	`Silence()`	Milliseconds of silence to insert.

a_pronounce	The fragment contains text with an associated phonemic pronunciation. The member functions are interpreted as for a_speak with the addition of:	
	`PhonemeIDs()`	DARPA phonemes that indicate the pronunciation to use for the given text.
a_bookmark	The fragment contains a bookmark. The member functions are interpreted as follows:	
	`Text()`	The ID of the bookmark.
	`TextLength()`	The length of the bookmark.
	`TextOffset()`	The offset in bytes from the beginning of the original text buffer.
a_spell	The fragment contains text to spell. The member functions are interpreted as for a_speak.	
a_section	The fragment contains a section of text. Not currently implemented.	

5.9 Setting Volume and Rate

An application can change the default speaking volume and rate with the `CTTSEngine::SetVolume()` and `CTTSEngine::SetRate()` functions. These functions may be called from a separate thread or from within the same thread.

Volume is expressed as a percentage of the normal speaking volume of a voice. The default value is 100, but can range from 0 to 500. *Note: Setting the volume above 200 may result in distortion of the voice.*

Speaking rate is expressed on a logarithmic scale from –18 to 18, representing 1/8 the normal rate to 8 times the normal rate. The default value of 0 indicates the normal speaking rate of the voice.

```
// Rate and volume adjustment
//
pEngine->SetRate(-10);        // Approximately 1/3 normal rate
pEngine->SetVolume(25);       // ¼ normal speaking volume
pEngine->Speak((PCUTF8String)"A slow whisper", 14, CTTSEngine::sf_default);
pEngine->SetRate(10);  // Approximately 3 times normal rate
pEngine->SetVolume(200);      // twice normal speaking volume
pEngine->Speak((PCUTF8String)"A quick shout!", 14, CTTSEngine::sf_default);
```

5.10 Stopping The Engine

An application may call the `CTTSEngine::Stop()` function to stop the engine from speaking text. All queued speaking requests will be stopped. This function may be called from a separate thread or from within the same thread.

```
//
// Stop speaking
//
        TTS_RESULT result = pEngine->Stop();
        . . .
```

5.11 Retrieving Phonetic Transcriptions

You may wish to find the phonetic transcription of a word or phrase to use to modify or create a new dictionary item. If you're not a linguist, it may be easiest to retrieve the phonetic transcription of a word or phrase from the TTS engine.

```
TTSTranscription myTranscription;
myTranscription.m_utfWord = "frog";
TTS_RESULT res = pEngine->Transcribe(myTranscription);
if (SUCCEEDED(res)) {
        cout << myTranscription.m_utfTranscription;
}
```

5.12 Custom Dictionaries

An application can specify pronunciations to be used while speaking text through the use of custom dictionaries. A dictionary is a list of entries consisting of a word, and one or more phonemic pronunciations for that word.

The format of a dictionary entry is as follows:

```
Word {phone1-1 phone1-2 phone1-3 …}(/part-of-speech) {phone2-1 …}
```

The "`Word`" is the textual representation of the word whose pronunciation is being specified. It is followed by one or more transcriptions. A transcription consists of an open brace (`{`) followed by a space-separated list of DARPA phoneme names for English words, SAMPA phoneme names for Spanish words, or SAMPA phoneme names for German words. The last phoneme is followed by a closing brace (`}`). This may be optionally followed by a slash (`/`) and a part of speech designation, (`noun, verb, modifier, function, or interjection`). If a part of speech is present, that pronunciation will only be used for words whose part of speech matches.

Here is a small, sample dictionary:

```
// Sample Dictionary
//
tomato {t ow - m ey 1 t ow}
lead {l iy d} {l eh d}/noun
```

Dictionaries are managed through the following set of member functions in the `CTTSEngine` class.

5.12.1 Creating a Dictionary

An application creates a dictionary with the `CTTSEngine::CreateDictionary()` function. The application supplies a name by which the dictionary can be referenced, a weight, used to specify the order in which dictionaries are sorted, (the higher the weight, the earlier the dictionary is searched), and the contents of the dictionary.

```
// Creating a dictionary - Be sure to surround the phonemes with curly braces
//
PCUTF8String pszMyDictionary = "\n\
tomato {t ow - m ey 1 t ow}\n\
lead {l iy d} {l eh d}/noun\n\
";

PCUTF8String pszMyDictName = "MyDictionary";

TTS_RESULT result =
    pEngine->CreateDictionary(pszMyDictName, 0, pszMyDictionary,
                              (PCUTF8String)"att_darpabet_english");
```

5.12.2 Updating a Dictionary

Once a dictionary has been created by an application, it can add and replace transcriptions using the `CTTSEngine::UpdateDictionary()` function.

```
// Updating a dictionary
//
PCUTF8String pszMyDictionaryUpdate = "\n\
tomato {t ow - m aa 1 t ow}\n\
potato {p ow - t ey 1 t ow}\n\
";

TTS_RESULT result = pEngine->UpdateDictionary(pszMyDictName,
                                              szMyDictionaryUpdate);
```

5.12.3 Clearing and Deleting a Dictionary

An application can clear all entries from a dictionary with the `CTTSEngine::ClearDictionary()` function. This leaves the dictionary in place with its current weight, but removes all entries from the dictionary. This call is useful if the application plans to completely replace the contents of a dictionary without losing its position in the dictionary search list. If an application is completely finished using a dictionary, it can be completely removed with the `CTTSEngine::DeleteDictionary()` function.

```
// Clearing and deleting a dictionary
//
pEngine->ClearDictionary(pszMyDictName);
pEngine->UpdateDictionary(pszMyDictName, pszMyDictionaryUpdate);
…
pEngine->DeleteDictionary(psz8MyDictName);
```

5.12.4 Changing Search Order

An application can change the order in which its custom dictionaries are searched by changing the weight of a dictionary. Increasing the weight will move the dictionary closer to the beginning of the list. Decreasing the weight will move it closer to the end

of the list. Dictionary weights are set when the dictionary is created, and can be modified later with the `CTTSEngine::ChangeDictionaryWeight()` function.

```
// Change search order
//
pEngine->CreateDictionary((PCUTF8String)”Dict1”, 5, 0);
pEngine->CreateDictionary((PCUTF8String)”Dict2”, 5, 1);

// Current search order is dict2 (weight 1) followed by dict1 (weight 0)
//

pEngine->ChangeDictionaryWeight((PCUTF8String)”Dict1”, 2);
// New search order is dict1 (weight 2) followed by dict2 (weight 1)
//
```

5.13 Optional Operating System Specific Classes

There are some classes defined that an application may use if desired.

5.13.1 CTTSWin32AudioPlayer

A Win32 audio player class is provided and used by the TTSDesktopPlayer application. It plays audio using the Microsoft WAVE API.

5.13.2 CTTSWin32AudioWriter

A Win32 RIFF WAVE file writer class is provided and used by the TTSDesktopFile application. It will write .WAV files for an application.

5.13.3 CTTSUnixAudioPlayer

A LINUX audio player class is provided and used by the TTSClientPlayer application.[5]

5.13.4 CTTSUnixAudioWriter

A UNIX audio writer class is provided and used by the TTSClientFile class. It writes audio out to raw files.

5.14 Changes from AT&T Natural Voices 1.0 Release

5.14.1 New Engine Models

The `TTSConfig` structure contains some new engine models to support specific types of AT&T Natural Voices server engines and the new desktop engine.

5.14.2 Library names have changed

New libraries have been created that separate support for client/server SDK and the standalone SDK.

[5] Release 1.1 does not include a Solaris audio player class.

6 Java Speech API Implementation

Release 1.4 allows developers to access the AT&T Natural Voices TTS engine from a Java application. You may also include JSML control tags in your input text. This chapter describes the features of the Java wrapper and explains how to access the AT&T Natural Voices TTS engine from a Java application.

6.1 Requirements

- JSDK 1.3.1 or higher is needed to compile the examples
- Insure that the JSDK/bin directory is included in your path

6.2 Installation

- Edit the "CLASSPATH" environment variable to include attjsapi.jar, jsapi.jar, and the directory which holds the examples SpeakArg.java and SpeakArgToFile.java. Finally, make sure the current directory is included as well.
- Copy "attjsapidt.dll" to your WINNT directory for Windows NT/2000/XP or to your Windows directory for Windows 98. Alternatively, you may provide the following argument to java when running an AT&T JSAPI Java application: `-Djava.library.path=<path to attjsapidt.dll>`
- Copy the "speech.properties" file to your home directory or to the jre/lib directory relative to your Java installation.

6.3 Compiling the Examples

To compile the SpeakArg examples, do the following in the "demo" directory:

```
javac SpeakArg.java SpeakArgToFile.java
```

If the examples don't compile cleanly, double check your CLASSPATH environment variable with "echo %CLASSPATH%". Make sure that both attjsapidt.jar and jsapi.jar are visible to running programs.

6.4 Running the Examples

To run the first example, do:

```
java SpeakArg "Welcome to AT&T Natural Voices."
```

After a brief initialization, you should here a voice speak the quoted text.

To run the second example, do:

```
java SpeakArgToFile ".\welcome.wav" "Welcom to AT&T Natural Voices."
```

After running, you should have a WAV file called "welcome.wav" in the current directory.

6.5 Using AT&T Natural Voices JSAPI Implementation

First, familiarize yourself with JSAPI by browsing the "synthesis" portion of the JSAPI Specification.

AT&T's implementation is a partial implementation due to architecture differences between the JSAPI specification and the design of AT&T's Natural Voices TTS Synthesizer.

The examples provided with AT&T's implementation of JSAPI provide a good base by which to start building JSAPI applications.

6.6 Available Synthesizer Voices

The following sythesizer modes and voices are supported by this implementation of JSAPI:

Name	Gender	Age	Style	Locale
Mike	Male	30	business	en_us
Mike16	Male	30	business	en_us
Crystal	Female	30	business	en_us
Crystal16	Female	30	business	en_us
Claire	Female	30	business	en_us
Claire16	Female	30	business	en_us
Rich	Male	30	business	en_us
Rich16	Male	30	business	en_us
Rosa	Female	30	business	es_us
Rosa16	Female	30	business	es_us
Klara	Female	30	business	de_de
Klara16	Female	30	business	de_de
Reiner	Male	30	business	de_de
Reiner16	Male	30	business	de_de
Audrey	Female	30	business	en_uk
Audrey16	Female	30	business	en_uk
Charles	Male	30	business	en_uk
Charles16	Male	30	business	en_uk

Alain	Male	30	business	fr_fr
Alain16	Male	30	business	fr_fr

Note that the JSAPI specification assumes that the engine mode determines the locale of the voice, however, the AT&T Natural Voices engine allows a single engine mode to support many simultaneous voices with different locales. The best strategy is to use the voice name to choose the voice and language you'd like to use.

6.7 Available Synthesizer Modes

The following synthesizer modes are supported in Release 1.4.

Engine Name	Mode Name	Locale
Desktop	audible	US
Desktop	file	US
ClientServer	audible	US
ClientServer	file	US
Serverlite	audible	US
Serverlite	file	US

6.8 Differences Between the JSAPI Specification and the AT&T Implementation

6.8.1 Audio

The native Java "javax.sound" classes were used to implement live audio streams from the AT&T Natural Voices TTS engine. Because the Java audio classes have not been finalized as of this writing, several of the audio classes in the JSAPI specification were not implemented:

1) javax.speech.AudioManager
2) javax.speech.AudioListener
3) javax.speech.AudioAdapter
4) javax.speech.AudioEvent

This means that there is essentially no listener interface provided for audio events.

Support for pcm, μlaw, alaw, and pcm live audio streams and files are provided.

6.8.2 Vocabulary

The javax.speech.VocabManager is not supported.

6.8.3 Queuing

The queuing mechanisms provided for in the JSAPI specification were partially implemented. In the AT&T TTS Engine, there is no true notion of word "queuing", in that there is no method by which a programmer can "peek" ahead to find out what is coming along in a queue.

That being said, queuing plays an important role in JSAPI, particularly when it comes to events. In AT&T JSAPI, the following table of queue events are used and implemented:

Event ID	Description
QUEUE_EMPTIED	This event is triggered whenever the TTS Engine has finished processing some speakable text. I.e. there are no more words to process.
QUEUE_UPDATED	This event is triggered whenever the TTS Engine has finished processing a word. These events always happen *before* the word is spoken.
TOP_OF_QUEUE	This event is immediately triggered whenever a "speak" function is called. Since the AT&T Natural Voices architecture doesn't use a queueing scheme, this provides the closest representation of the JSAPI specification.

Finally, due to reasons previously discussed the "SynthesizerQueueItem" is not implemented.

6.8.4 XML

AT&T Natural Voices 1.4 supports JSML and SSML, while version 1.2.1 supports only SSML. You may pass SSML to any of the "speak" methods that take some form of JSML as an argument, as no syntax checking is performed.

6.8.5 Voices

Several attributes of JSAPI voices were not implemented. The following describes the differences:

- AGE_MIDDLE_ADULT, AGE_YOUNGER_ADULT, and AGE_NEUTRAL all map to an "Adult" voice, all others map exactly.
- GENDER_DON'T_CARE and GENDER_NEUTRAL both map to "Don't care" gender.

6.8.6 Synthesizer

The following are not implemented:

- isRunning
- enumerateQueue
- phoneme
- cancel
- cancelAll

6.8.7 Engine

The following are not implemented:

- isRunning
- pause
- resume

6.8.8 Permissions

Speech permission is not implemented.

7 SAPI 4 Text Markup

The AT&T Natural Voices TTS engine does a great job of synthesizing most text without special instructions but there may be special circumstances where you may wish to provide hints to fine-tune the pronunciation of certain words or phrases. The AT&T Natural Voices TTS engine allows users to mark up the text to be spoken to include special control tags that change the way the text is pronounced. Four different speech platforms and markup languages are supported: Microsoft SAPI 4.0, Microsoft SAPI 5.1, the Speech Synthesis Markup Language (SSML) component of Voice XML, and JSML, the Java Speech Markup Language.

This chapter describes the SAPI 4 control tags that are supported by the AT&T Natural Voices TTS engine.

Not all tags are supported in every language. The description of each control tag lists the languages that support that tag.

The following table specifies which tags are supported and in which languages:

SAPI 4.0 Tag	US English	UK English	German	Spanish	French
Chr	No	No	No	No	No
\Com=*string*\	Yes	Yes	Yes	Yes	Yes
\Ctx=*string*\					
Address	Yes	Yes	Yes	Yes	Yes
C	No	No	No	No	No
Document	Yes	Yes	Yes	Yes	Yes
E-Mail	Yes	Yes	Yes	Yes	Yes
Numbers	Yes	Yes	Yes	Yes	Yes
Spreadsheet	No	No	No	No	No
Unknown	Yes	Yes	Yes	Yes	Yes
Normal	Yes	Yes	Yes	Yes	Yes
\Dem\	No	No	No	No	No
\Emp\	No	No	No	No	No
\Eng;GUID:command\	No	No	No	No	No
\Mrk=*number*\	Yes	Yes	Yes	Yes	Yes
\Pau=*msecs*\	Yes	Yes	Yes	Yes	Yes
\Pit=*Number*\	No	No	No	No	No
\Pra=*value*\	No	No	No	No	No
\Prn=*text=pronunciation*\	Yes	Yes	Yes	Yes	Yes
\Pro=*number*\	No	No	No	No	No
\Prt=*string*\	No	No	No	No	No
\Rms=*number*\	Yes	Yes	No	No	No
\RmW=*number*\	Yes	No	No	No	No
\RPit=*value*\	No	No	No	No	No
\RPn=*value*\	No	No	No	No	No
\RSpd=*rate*\	Yes	Yes	Yes	Yes	Yes
\Rst\	Yes	Yes	Yes	Yes	Yes
\Spd=*rate*\	Yes	Yes	Yes	Yes	Yes
\Vce=*characteristic=value*\	Yes	Yes	Yes	Yes	Yes
\Vol=*level*\	Yes	Yes	Yes	Yes	Yes

NOTE: The SAPI client and the AT&T Natural Voices TTS SDK both treat control tags as case-insensitive.

7.1 Com

The **COM** tag tells the TTS engine to treat the text as a comment which is not synthesized.

Syntax: **\Com=***string*****

Languages: US English, UK English, Spanish, German, French

Example: Comments can be inserted \Com=this isn't spoken\ anywhere in the text.

is pronounced "Comments can be inserted anywhere in the text"

7.2 Ctx

Context tags provide hints to the TTS engine about how text should be pronounced. The TTS engine supports a number of different contexts that can be used to fine tune the pronunciation of words.

7.2.1 Address Context

The **Address** context tells the TTS engine to treat the text as an address.

Syntax: **\Ctx="Address"** ***text***

Languages: US English, UK English, Spanish, German, French

Example: \Ctx="Address"\ 123 Main St. New York, NY 10017

is pronounced "one twenty three main street, New York, New York one zero zero one seven"

7.2.2 Document Context

The **Document** context tells the TTS engine to treat the text as a document.

Syntax: **\Ctx="Document"** ***text***

Languages: US English, UK English, Spanish, German, French

Example: \Ctx="Document"\ text

This is the default mode of the engine and the text is interpreted as a document.

7.2.3 Email Context

The **Email** context tells the TTS engine to treat the text as an email address.

Syntax: **\Ctx="Email"** ***text***

Languages: US English, UK English, Spanish, German, French

Example: \Ctx="Email"\ help@naturalvoices.att.com

is pronounced "help at natural voices dot A T T dot com"

7.2.4 Numbers Context

The **Numbers** context tells the TTS engine to treat the text as a number.

Syntax: **\Ctx="Numbers"** ***text***

Languages: US English, UK English, Spanish, German, French

Example: \Ctx="Numbers"\ 10,243

is pronounced "ten thousand two hundred forty three"

Example: \Ctx="Numbers"\ 3.14159

is pronounced "three point one four one five nine"

7.2.5 Unknown Context

The **Unknown** context tells the TTS engine to do its best with the input without additional contextual clues.

Syntax: \Ctx="Unknown"\ *text*

Languages: US English, UK English, Spanish, German, French

Example: \Ctx="Unknown"\ this is random text

is pronounced "this is random text"

7.2.6 Normal Context

The **Normal** context tells the TTS engine to do its best with the input without additional contextual clues.

Syntax: \Ctx="Normal"\ *text*

Languages: US English, UK English, Spanish, German, French

Example: \Ctx="Unknown"\ this is random text

is pronounced "this is random text"

7.3 Mrk

The application may insert a user bookmark in the text. The TTS Engine will optionally inform the application via a notification when that bookmark is reached in audio stream.

Any number of bookmarks may be inserted anywhere in the text.

Syntax: \Mrk="*number*"

where *number* is the number of the bookmark. The TTS engine will generate a bookmark notification with the specified *number* when that spot is reached in the audio stream.

Languages: US English, UK English, Spanish, German, French

Example: `The quick \Mrk="1"\ fox jumped over the lazy \Mrk="2"\ dog.`

The TTS engine will provide a notification just before speaking the word "fox" and another notification just before speaking the word "dog".

7.4 Pau

The **Pau** tag inserts silence of the specified duration (in milliseconds) into the audio stream at the specified location.

Syntax: **`\Pau="`*`number`*`"\`**

Languages: `US English, UK English, Spanish, German, French`

Example: `\Pau=500\`

inserts 500 milliseconds of silence into the audio stream.

7.5 Prn

The **Prn** tag allows the user to change the pronunciation of a word by explicitly specifying the pronunciations using the IPA phonetic alphabet. See Appendix A for more details about the IPA phonetic alphabet The transcriptions specified by the Prn tag persist across engine instances until they are removed or changed with a matching \Prn\ tag.

Syntax: **`\Prn=orthography=pronunciation\`**

sets the phonetic transcription for *orthography* to *pronunciation* for all SAPI4 applications.

\Prn=orthography

restores the phonetic transcription of *orthography* to the system default.

Languages: `US English, UK English, Spanish, German, French`

Example: `\Prn=MB=m eh g ax b ay t s\` 80 MB

changes the pronunciation of "MB" to "megabytes" in the system dictionary so text is pronounced "80 megabytes"

7.6 RmS

The **RmS** tag tells the engine to spell all words and enumerate all digits.

Syntax: **`\RmS=1\`**

Turns on spell mode where all words are spelled and numbers are enumerated.

Syntax: **`\RmS=0\`**

Turns off spell mode (default).

Languages: `US English`

Example: `hello 123 \RmS=1\ hello 123 \RmS=0\ hello`

Is pronounced “hello one hundred twenty three H E L L 0 1 2 3 hello”

7.7 RmW

The **RmW** tag tells the engine to leave a brief pause between words.

Syntax: **`\RmW=1\`**

turns on pause between words mode.

Syntax: **`\RmS=0\`**

Turns off pause between words mode (default).

Languages: `US English`

Example: `hello world \RmW=1\ hello world \RmW=0\ hello world`

Is pronounced “hello world hello <*pause*> world <*pause*> hello world”

7.8 RSpd

The **RSpd** tag sets the relative speaking rate of the voice..

Syntax: **`\RSpd=`*`value`*`\`**

Sets the relative speed to *value* where 100 is the default speed for the voice.

Languages: `US English, UK English, Spanish, German, French`

Example:
```
\RSpd=100\This is speed 100, the default rate
\RSpd=50\This is half speed \RSpd=100\
\RSpd=200\This is double the default speed \RSpd=100\
\RSpd=300\This is triple the default speed \RSpd=100\
```

7.9 Rst

The **Rst** tag resets the engine to the default settings for the current mode

Syntax: **`\Rst\`**

Languages: `US English, UK English, Spanish, German, French`

Example:
```
\Rst\ this is the default rate
\RSpd=50\This is half speed
\Rst\ back to the default rate
```

7.10 Spd

The **Spd** tag sets the speaking rate of the voice in words per minute.

Syntax: **`\Spd=`*`wpm`*`\`**

sets the speaking rate to *wpn* words per minute.

Languages: `US English, UK English, Spanish, German, French`

Example: `\Spd=300\This is 300 words per minute`

```
\Spd=150\This is 150 words per minute
```

7.11 Vce

The **Vce** tag allows the application to change the voice of the TTS speaker from the input text. You can use this feature to change voices, e.g. you might use different voices to speak different sections of an email message or carry on a conversation between two different voices. This release includes both a US English male voice called “Mike” and a US English female voice called “Crystal”.

The default voice for a server is specified when the server process is started (see the Server Command Line Options for details).

You choose a voice by specifying a set of required and optional voice attributes where the attributes are:

- Gender: `male, female, or neutral`
- Age: `baby, toddler, child, adolescent, adult, or senior`
- Language: use `409` for US English, `809` for UK English, `40a` for Spanish, `407` for German, or `40c` for French
- Speaker: `mike, crystal, rich, Claire, Rosa, Reiner, Klara, mike16,crystal16, rich16, claire16, rosa16, reiner16, klara16, Audrey, audrey16, charles, charles16, alain, alain16` or other voices names
- You may specify several attributes for the required and optional fields.

Syntax: **`\Vce=characteristic=value[[,characteristic=value]]`**

Languages: `US English, UK English, Spanish, German, French`

Example: `\Vce=speaker=mike> Hi, I’m Mike`

is pronounced, “Hi, I’m Mike” using Mike’s voice.

Example: `\Vce=speaker=rosa\ Hola, me llamo Rosa`

is pronounced, “Hola, me llamo Rosa” using Rosa’s Spanish voice.

Example: `\Vce=speaker=reiner\ Ich bin Reiner`

is pronounced, “Ich bin Reiner” using Reiner’s German voice.

Example: `\Vce=speaker=crystal\ I’m Crystal \`

is pronounced, “I’m Crystal” using Crystal’s voice.

Example: `This is Mike \Vce=speaker=crystal\ This is Crystal \Rst\ This is Mike again.`

Assume for this example that the default voice is Mike. This string is pronounced in Mike’s voice “This is Mike”, then in Crystal’s voice, “This is Crystal”, then in Mike’s voice, “This is Mike again”.

You can include any number of `<voice>` tags in the input text to change voices and languages.

Example:

```
\Vce=speaker=crystal\ I’m Crystal
\Vce=speaker=rosa\ me llamo Rosa
```

```
\Vce=speaker=mike\ I'm Mike
```

switches from Crystal speaking English, to Rosa speaking Spanish, to Mike speaking English.

Note that switching voices forces the server to load the data files for the each voice that is used which may result in noticeable delays. Voice switches will happen instantaneously once the voice data is in place.

7.12 Vol

The **Vol** tag allows the application to change the volume of the TTS voice. Note that this does not change the volume of the output device, but it does raise or lower the volume of the text spoken within the context of the tag.

Syntax: \VOL=*volume*

where *volume* is a value from 0 (silence) to 65535 (maximum volume).

Languages: US English, UK English, Spanish, German, French

Example: This is the default volume

```
\Vol=32768\ this is the middle volume
\Pau=1000\ \Vol=1\ this is nearly silent
\Vol=65536\ this is max volume
```

8 Microsoft SAPI 5.1 Control Tags

The AT&T Natural Voices TTS engine does a great job of synthesizing most text without special instructions but there may be special circumstances where you may wish to provide hints to fine-tune the pronunciation of certain words or phrases. The AT&T Natural Voices TTS engine allows users to mark up the text to be spoken to include special control tags that change the way the text is pronounced. Four different speech platforms and markup languages are supported: Microsoft SAPI 4.0, Microsoft SAPI 5.1, the Speech Synthesis Markup Language (SSML) component of Voice XML, and JSML, the Java Speech Markup Language.

The AT&T Natural Voices TTS engine supports a subset of the SAPI 5.1 control tags and adds a few extras. Not all tags are supported in all languages. The following table specifies which tags are supported and in which languages:

SAPI Tag	US English	UK English	German	Spanish	French
ATT_Div	Yes	No	No	No	No
ATT_Ignore_Case	Yes	Yes	Yes	Yes	Yes
Bookmark	Yes	Yes	Yes	Yes	Yes
Context					
Address	Yes	Yes	Yes	Yes	Yes
Address_postal	Yes	Yes	Yes	Yes	Yes
ATT_literal	Yes	No	No	No	No
ATT_math	Yes	No	No	No	No
ATT_measurement	Yes	No	No	No	No
Currency	Yes	Yes	Yes	Yes	Yes
Date	Yes	Yes	Yes	Yes	Yes
Date_ymd	No	No	No	No	No
Date_ym	No	No	No	No	No
Date_my	No	No	No	No	No
Date_dm	No	No	No	No	No
Date_md	Yes	Yes	Yes	Yes	Yes
Date_mdy	Yes	No	No	No	No
Date_year	Yes	No	No	No	No
Number_cardinal	Yes	No	Yes	No	No
Number_digit	No	No	No	No	No
Number_fraction	Yes	No	No	No	No
Number_decimal	Yes	Yes	Yes	Yes	Yes
Phone_number	Yes	No	No	No	No
Time	Yes	No	No	No	No
Web	Yes	No	No	No	No
Web_url	Yes	No	No	No	No
E-mail	Yes	No	No	No	No
E-mail_address	Yes	No	No	No	No
Emph	No	No	No	No	No
Lang	No	No	No	No	No
Partofsp	No	No	No	No	No
Pitch	No	No	No	No	No
Pron	Yes	Yes	Yes	Yes	Yes

Rate	Yes	Yes	Yes	Yes	Yes
Silence	Yes	Yes	Yes	Yes	Yes
Spell	Yes	Yes	No	No	No
Voice	Yes	Yes	Yes	Yes	Yes
Volume	Yes	Yes	Yes	Yes	Yes

NOTE: The SAPI client and the AT&T Natural Voices TTS SDK both treat control tags as case-insensitive.

8.1 ATT_Div

The **ATT_DIV** tag tells the TTS engine to change the prosody to reflect the end of a sentence or paragraph, regardless of the surrounding punctuation.

Syntax: **<ATT_DIV Type="paragraph"/>**

<ATT_DIV Type="sentence"/>

Languages: US English

Example: The quick <ATT_DIV Type="sentence"/> fox jumped over <ATT_DIV Type="paragraph"/> the lazy dog.

The TTS engine changes the prosody as if the sentence ended after the word *"quick"* and changes the prosody again as if the paragraph ended after speaking the word "*over*".

8.2 ATT_Ignore_case

The **ATT_Ignore_case** tag tells the TTS engine to ignore the capitalization of words within the specified context. This mode was most useful prior to Release 1.4 to override the default behavior which is to spell capitalized words. This tag is provided for backward compatibility.

Syntax: **<ATT_IGNORE_CASE>** *text* **</ATT_IGNORE_CASE>**

Languages: US English, UK English, Spanish, German, French

Example: <ATT_ignore_case> THIS CONTRACT </ATT_ignore_case>

is pronounced "this contract"

8.3 Bookmark

The application may insert a user bookmark in the text. The TTS Engine will optionally inform the application via a notification when that bookmark is reached in audio stream.

Any number of bookmarks may be inserted anywhere in the text.

Syntax: **<BOOKMARK Mark=***"label"***/>** *text*

where *label* is an text label. The TTS engine will generate a bookmark notification with the specified *label* when that spot is reached in the audio stream.

Languages: US English, UK English, Spanish, German, French

Example: The quick <BOOKMARK Mark="mark 1"/> fox jumped over the lazy <BOOKMARK Mark="mark 2"/> dog.

The TTS engine will provide a notification just before speaking the word "fox" and another notification just before speaking the word "dog".

8.4 Context

Context tags provide hints to the TTS engine about how text should be pronounced. The TTS engine supports a number of different contexts that can be used to fine tune the pronunciation of words.

Syntax: <CONTEXT ID="context_id [, context_id]+"/> *text* ***</context>***

Note: The SAPI 5.1 specification allows a context to specify only one context_id while the AT&T TTS engine allows many simultaneous contexts, all of which apply to the scoped context.

Languages: US English, UK English, Spanish, German, French

Example: 3 1/2"

is pronounced "three one slash two double quote"

Example: <Context Id="ATT_measurement"> 3 1/2" </Context>

is pronounced "three one slash two inches"

Example: <Context Id="ATT_measurement, number_fraction"> 3 1/2" </Context>

is pronounced "three and one half inches"

8.4.1 Address Context

The **Address** context tells the TTS engine to treat the text as an address.

Syntax: <CONTEXT Id="Address"> *text* </CONTEXT>

Languages: US English, UK English, Spanish, German, French

Example:
```
<Context Id="Address">
    123 Main St.
    New York, NY 10017
    </Context>
```

is pronounced "one twenty three main street, New York, New York one zero zero one seven"

8.4.2 Address_postal Context

The **Address_Postal** context tells the TTS engine to treat the text as a postal address.

Syntax: <CONTEXT Id="Address_postal"> *text* </CONTEXT>

Languages: US English, UK English, Spanish, German, French

Example:
```
<Context Id="Address_Postal">
    123 Main St.
```

```
New York, NY 10017
</Context>
```

is pronounced “one twenty three main street, New York, New York one zero zero one seven”

Note: The Address and Address_Postal contexts are equivalent.

8.4.3 ATT_Literal Context

The **ATT_Literal** context instructs the TTS engine to pass the string through literally without applying additional context processing such as abbreviations, addresses, dates, math symbols, measurements, names, numbers, times, or phone numbers.

Syntax: <CONTEXT Id=”ATT_Literal”> *text* </CONTEXT>

Languages: US English

Example: `<Context Id=”ATT_Literal”> misc. </Context> misc.`

is pronounced “misk miscellaneous”

8.4.4 ATT_Math Context

The **ATT_Math** context tells the TTS engine to treat the text as a mathematical expression.

Syntax: <CONTEXT Id=”ATT_Math”> *text* </CONTEXT>

Languages: US English

Example: `<Context Id=”ATT_Math”> 3+4=7 </Context> 3+5=8`

is pronounced “three plus four equals seven three plus sign five equal sign eight”

8.4.5 ATT_Measurement Context

The **ATT_Measurement** context tells the TTS engine to treat the text as a measurement, e.g. single quotes are pronounced “feet” and double quotes are pronounced “inches”.

Syntax: <CONTEXT Id=”ATT_Measurement”> *text* </CONTEXT >

Languages: US English

Example: `<Context Id=”ATT_Measurement”> 5’ 3” </Context> 7’ 9”`

is pronounced “five feet three inches seven single quote nine double quote”

8.4.6 Currency Context

The **Currency** context tells the TTS engine to treat the text as currency and expand $ and decimal numbers appropriately. The Currency Context works only with US currency and not other currencies.

Syntax: <CONTEXT Id=”Currency”> *text* </CONTEXT>

Languages: US English, UK English, Spanish, German, French

Example: `<Context Id=”Currency”> $25.32 </Context>`

is pronounced “twenty five dollars and thirty two cents”

8.4.7 Date_MD Context

The **Date_MD** context tells the TTS engine to treat the text as a month and day.

Syntax: **<CONTEXT Id="Date_MD"> *text* </CONTEXT>**

Languages: US English, UK English, Spanish, German, French

Example: <Context Id="Date_MD"> Dec 25 </Context>

is pronounced "December twenty fifth"

8.4.8 Date_MDY Context

The **Date_MDY** context tells the TTS engine to treat the text as a month, day, and year.

Syntax: **<CONTEXT Id="Date_MDY"> *text* </CONTEXT>**

Languages: US English

Example: <Context Id="Date_MDY"> Dec 25, 2001 </Context>

is pronounced "December twenty fifth two thousand one"

8.4.9 Date_Year Context

The **Date_Year** context tells the TTS engine to treat the text as a year.

Syntax: **<CONTEXT Id="Date_Year"> *text* </CONTEXT>**

Languages: US English

Example: <Context Id="Date_Year"> 1999 </Context> 1999

is pronounced "nineteen ninety nine nineteen hundred ninety nine"

8.4.10 Number_Cardinal Context

The **Number_Cardinal** context tells the TTS engine to treat the text as a cardinal number.

Syntax: **<CONTEXT Id="Number_Cardinal"> *text* </CONTEXT>**

Languages: US English, Spanish

Example: <Context Id="Number_Cardinal"> 10,243 </Context>

is pronounced "ten thousand two hundred forty three"

8.4.11 Number_Decimal Context

The **Number_Decimal** context tells the TTS engine to treat the text as a decimal number.

Syntax: **<CONTEXT Id="Number_Decimal"> *text* </CONTEXT>**

Languages: US English, UK English, Spanish, German, French

Example: <Context Id="Number_Decimal"> 3.14159 </Context>

is pronounced "three point one four one five nine"

8.4.12 Number_Fraction Context

The **Number_Fraction** context tells the TTS engine to treat the text as a fraction.

Syntax: **<CONTEXT Id="Number_Fraction"> *text* </CONTEXT>**

Languages: US English

Example: <Context Id="Number_Fraction"> 5 3/4 </Context>
is pronounced "five and three fourths"

Note: Fractions may not have spaces between the numerator, the slash, and the denominator.

Example: <Context Id="Number_Fraction"> 11 /16 11/16 </Context>
is pronounced "eleven slash 16 eleven sixteenths"

8.4.13 Phone_Number Context

The **Phone_Number** context tells the TTS engine to treat the text as a telephone number.

Syntax: **<CONTEXT Id="Phone_Number"> *text* </CONTEXT>**

Languages: US English

Example: <Context Id="Phone_Number"> (212)555-1212 </Context>
is pronounced "two one two five five five one two one two"

8.4.14 Time Context

The **Time** context tells the TTS engine to treat the text as a time.

Syntax: **<CONTEXT Id="Time"> *text* </CONTEXT>**

Languages: US English

Example: <Context Id="Time"> 12:34 PM </Context>
is pronounced "twelve thirty four P M"

8.4.15 Web Context

The **Web** context tells the TTS engine to treat the text as a URL.

Syntax: **<CONTEXT Id="Web"> *text* </CONTEXT>**

Languages: US English, UK English, Spanish, German, French

Example: <Context Id="Web"> http://www.naturalvoices.att.com </Context>
is pronounced "H T T P W W W natural voices dot A T T dot com"

8.4.16 Web_url Context

The **Web** context tells the TTS engine to treat the text as a URL.

Syntax: **<CONTEXT Id="Web_url"> *text* </CONTEXT>**

Languages: US English, UK English, Spanish, German, French

Example: <Context Id="Web_url"> http://www.naturalvoices.att.com </Context>
is pronounced "H T T P W W W natural voices dot A T T dot com"

8.4.17 Email Context

The **Email** context tells the TTS engine to treat the text as an email address.

Syntax: **<CONTEXT Id="Email">** ***text*** **</CONTEXT>**

Languages: US English, UK English, Spanish, German, French

Example: <Context Id="Email"> help@naturalvoices.att.com </Context>

is pronounced "help at natural voices dot A T T dot com"

8.4.18 Email_address Context

The **Email_address** context tells the TTS engine to treat the text as an email address.

Languages: US English, UK English, Spanish, German, French

Example: <Context Id="Email"> help@naturalvoices.att.com </Context>

is pronounced "help at natural voices dot A T T dot com"

8.5 Pron

The **PRON** tag allows the user to specify pronunciations explicitly in the input text. Including the orthography, i.e. the word or words represented by the phonemes, are optional, but recommended, because some modules in the front end depend on orthography. The pronunciations must be represented using the SAPI phoneme set when using the SAPI 5.1 client or the DARPA phoneme set when using the AT&T Natural Voices TTS Client SDK. See Appendix A for more details about DARPA, SAPI, and SAMPA phoneme sets for English, UK English, German, Spanish, and French.

Syntax: **<PRON sym=" *phoneme*$^{+}$ "/>**

Syntax: **<PRON sym=" *phoneme*$^{+}$ "> *orthography* </PRON>**

Languages: US English, UK English, Spanish, German, French

Example: <Pron sym="b ow t 1"/>

is pronounced "boat"

Example: <Pron sym="m eh 1 zh er"/> measure </Pron>

is pronounced "measure" when spoken by an English voice.

Example: <Pron sym="m w i 1"/>

is pronounced "muy" when spoken by a Spanish voice.

Example: <Pron sym="m al 1 n"/>

is pronounced "mein" when spoken by a German voice.

8.6 Rate

The **RATE** tag changes the rate at which the text is spoken. You can specify either the absolute rate or a relative change in the current speaking rate. The rate can be

used using a scale of integer values from –18 to 18 where the rate is distributed on a logarithmic scale[6] where 0 is default rate for the voice.

The following table shows frequently used rates:

Rate	Effect
-18	1/8x default rate
-10	1/3 x default rate
-6	½ x default rate
0	default rate
6	2x default rate
10	3x default rate
18	8x default rate

Syntax: **<RATE ABSSPEED=" *absoluteRate* "> *content* </RATE>**

where *absoluteRate* is an integer between –18 and 18 sets the speaking rate to the specified value.

Syntax: **<RATE SPEED=" *relativeRate* "> *content* </RATE>**

where *relativeRate* is an integer between –18 and 18 increments the rate by the specified value. Using a relativeRate < 0 decreases the speed.

Languages: `US English, UK English, Spanish, German, French`

Example: This is the default speed
```
<rate speed="-6">
    this is half speed -6
    <rate speed="12">  this is double speed 6 </rate>
    back to half speed -6
</rate>
back to the default rate
```

Example: This is the default speed
```
<rate speed="-6">
    this is half speed -6
    <rate absspeed="6">  this is double speed 6 </rate>
    back to half speed -6
</rate>
back to the default rate
```

8.7 Silence

The **SILENCE** tag inserts silence of the specified duration (in milliseconds) into the audio stream at the specified location.

Syntax: **<SILENCE msec="*integer*"/>**

[6] The logarithmic scale is used to provide SAPI compliance. We think it's confusing too…

Languages: `US English, UK English, Spanish, German, French`

Example: `<Silence msec="500"/>`

inserts 500 milliseconds of silence into the audio stream.

8.8 Spell

The **Spell** tag spells all of the words within the context. The **Spell** tag is useful in a number of different situations including forcing an acronym to be spelled rather than spoken.

Syntax: **<SPELL>** ***text*** **</SPELL>**

Languages: `US English`

Example: `Entering spell mode <spell> now </spell>`

is pronounced "entering spell mode N-O-W".

8.9 Voice

The **Voice** tag allows the application to change the voice of the TTS speaker from the input text. You can use this feature to change voices, e.g. you might use different voices to speak different sections of an email message or carry on a conversation between two different voices. This release includes both a US English male voice called "Mike" and a US English female voice called "Crystal".

The default voice for a server is specified when the server process is started (see the Server Command Line Options for details).

You choose a voice by specifying a set of required and optional voice attributes where the attributes are:

- Gender: `male` or `female`
- Age: `child`, `teen`, `adult`, or `senior`
- Language: use `409` for US English, or `40a` for Spanish, or `407` for German
- Name: `mike, crystal, rich, Claire, Rosa, Reiner, Klara, mike16,crystal16, rich16, claire16, rosa16, reiner16, klara16` or other voices names
- You may specify several attributes for the required and optional fields.

Syntax: **<VOICE**

[Required="*attributes*"]

[Optional="*attributes*"]>

text

</VOICE>

Where *attributes* is specified as

`attribute`^{1}`=value`^{1} [; `attribute`^{n}`=value`^{n}]*

Languages: `US English, UK English, Spanish, German, French`

Example: `<Voice Required="Name=mike"> Hi, I'm Mike </Voice>`

is pronounced, "Hi, I'm Mike" using Mike's voice.

Example: `<Voice Required=”Name=rosa”> Hola, me llamo Rosa </Voice>`

is pronounced, “Hola, me llamo Rosa” using Rosa’s Spanish voice.

Example: `<Voice Required=”Name=reiner”> Ich bin Reiner </Voice>`

is pronounced, “Ich bin Reiner” using Reiner’s German voice.

Example: `<Voice Required=”Name=crystal”> I’m Crystal </Voice>`

is pronounced, “I’m Crystal” using Crystal’s voice.

Example: `This is Mike <Voice Required=”Name=crystal”> This is Crystal </voice> This is Mike again.`

Assume for this example that the default voice is Mike. This string is pronounced in Mike’s voice “This is Mike”, then in Crystal’s voice, “This is Crystal”, then in Mike’s voice, “This is Mike again”.

You can include any number of `<voice>` tags in the input text to change voices and languages.

Example:

```
<Voice Required=”Name=crystal”> I’m Crystal </Voice>
<Voice Required=”Name=rosa”> me llamo Rosa </Voice>
<Voice Required=”Name=mike”> I’m Mike </Voice>
```

Switches from Crystal speaking English, to Rosa speaking Spanish, to Mike speaking English.

Note that switching voices forces the server to load the data files for the each voice that is used which may result in noticeable delays. Voice switches will happen instantaneously once the voice data is in place.

8.10 Volume

The **Volume** tag allows the application to change the volume of the TTS voice. Note that this does not change the volume of the output device, but it does raise or lower the volume of the text spoken within the context of the tag.

Syntax: **`<VOLUME Level=” volume “> text </VOLUME>`**

where *volume* is a value from 0-200. A value of 100 is the voice’s default volume, a value of 0 changes the volume to 0 and a value of 200 doubles the volume. The volume changes linearly.

Languages: `US English, UK English, Spanish, German, French`

Example: This is the default volume

```
<volume level="0"> Now I'm whispering </volume>
<volume level="50"> a little louder </volume>
<volume level="100"> back to normal </volume>
<volume level="150"> a little louder </volume>
<volume level="200"> very loud </volume>
<volume level="100"> back to default volume </volume>
```

9 SSML Control Tags

The AT&T Natural Voices TTS engine does a great job of synthesizing most text without special instructions but there may be special circumstances where you may wish to provide hints to fine-tune the pronunciation of certain words or phrases. The AT&T Natural Voices TTS engine allows users to mark up the text to be spoken to include special control tags that change the way the text is pronounced. Four different speech platforms and markup languages are supported: Microsoft SAPI 4.0, Microsoft SAPI 5.1, the Speech Synthesis Markup Language (SSML) component of Voice XML, and JSML, the Java Speech Markup Language.

The AT&T Natural Voices TTS engine supports a subset of the SSML control tags and adds a few extras. Not all tags are supported in all languages. The following table specifies which tags are supported and in which languages:

XML Tag	US English	UK English	German	Spanish	French
<ATT_Ignore_Case>	Yes	Yes	Yes	Yes	Yes
<speak> text </speak>	Yes	Yes	Yes	Yes	Yes
<speak xml:lang="en_us"> text </speak>	Yes	No	No	No	No
<speak xml:lang="es_us"> text </speak>	No	No	No	Yes	No
<speak xml:lang="de_de"> text </speak>	No	No	Yes	No	No
<paragraph> text </paragraph>	Yes	Yes	Yes	Yes	Yes
text	Yes	Yes	Yes	Yes	Yes
<sentence> text </sentence>	Yes	Yes	Yes	Yes	Yes
<s> text </s>	Yes	Yes	Yes	Yes	Yes
<say-as type="ATT_Literal"> text </say-as>	Yes	No	No	No	No
<say-as type="ATT_Math"> text </say-as>	Yes	No	No	No	No

<say-as type="ATT_Measurement"> text </say-as>	Yes	No	No	No	No
<say-as type="acronym"> text </say-as>	Yes	No	No	No	No
<say-as type="number"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="number:ordinal"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="number:digits"> text </say-as>	No	No	No	No	No
<say-as type="date"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="date:dmy"> text </say-as>	No	No	Yes	No	No
<say-as type="date:mdy"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="date:ymd"> text </say-as>	No	No	No	No	No
<say-as type="date:ym"> text </say-as>	No	No	No	No	No
<say-as type="date:my"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="date:md"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="date:y"> text </say-as>	Yes	No	No	No	No
<say-as type="date:m"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="date:d"> text </say-as>	No	No	No	No	No
<say-as type="time"> text </say-as>	Yes	No	No	No	No
<say-as type="time:hms"> text </say-as>	Yes	No	No	No	No
<say-as type="time:hm"> text </say-as>	Yes	No	No	No	No
<say-as type="time:h"> text </say-as>	Yes	No	No	No	No
<say-as type="duration"> text </say-as>	No	No	No	No	No
<say-as type="duration:hms"> text </say-as>	No	No	No	No	No
<say-as type="duration:hm"> text </say-as>	No	No	No	No	No
<say-as type="duration:h"> text </say-as>	No	No	No	No	No
<say-as type="duration:m"> text </say-as>	No	No	No	No	No
<say-as type="duration:s"> text </say-as>	No	No	No	No	No
<say-as type="currency"> text	Yes	Yes	Yes	Yes	Yes

</say-as>					
<say-as type="telephone"> text </say-as>	Yes	No	No	No	No
<say-as type="name"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="net"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="net:email"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="net:url"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as type="address"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<say-as sub="sub"> text </say-as>	Yes	Yes	Yes	Yes	Yes
<phoneme alphabet="att_darpabet_english" ph="…"> orthography </phoneme>	Yes	No	No	No	No
<phoneme alphabet="att_sampa_german" ph="…"> orthography </phoneme>	No	No	Yes	No	No
<phoneme alphabet="att_sampa_german" ph="…"> orthography </phoneme>	No	No	Yes	No	No
<phoneme alphabet="att_sampa_spanish" ph="…"> orthography </phoneme>	No	No	No	Yes	No
<phoneme alphabet="att_sampa_french" ph="…"> orthography </phoneme>	No	No	No	No	Yes
<!ENTITY … !>	No	No	No	No	No
<voice					
Gender="male, female, neutral"	Yes	Yes	Yes	Yes	Yes
age="integer"	Yes	Yes	Yes	Yes	Yes
category="child, teenager, adult, elder"	Yes	Yes	Yes	Yes	Yes
variant	No	No	No	No	No
Name="mike, crystal, rosa, rich,claire,reiner,klara,charles, audrey,alain"	Yes	Yes	Yes	Yes	Yes
<emphasis> text </emphasis>	No	No	No	No	No
<break/>	Yes	Yes	Yes	Yes	Yes
<break size="none, small, medium, large" \>	Yes	Yes	Yes	Yes	
<break time="*integer*"\>	Yes	Yes	Yes	Yes	Yes

<prosody					
pitch="high, medium, low, default"	No	No	No	No	No
contour="…"	No	No	No	No	No
range="…"	No	No	No	No	No
rate="fast, medium, slow, default"	Yes	Yes	Yes	Yes	Yes
duration="…"	No	No	No	No	No
volume="silent, soft, medium, loud, default"	Yes	Yes	Yes	Yes	Yes
Audio	No	No	No	No	No
Mark	Yes	Yes	Yes	Yes	Yes

NOTE: AT&T Natural Voices TTS SDK treats control tags as case-insensitive.

9.1 ATT_Ignore_Case

The **ATT_Ignore_case** tag tells the TTS engine to ignore the capitalization of words within the specified context. This mode is useful to override the default behavior which is to spell all capitalized words.

Syntax: <ATT_IGNORE_CASE> *text* **</ATT_IGNORE_CASE>**

Languages: US English, UK English, Spanish, German, French

Example: <ATT_ignore_case> THIS CONTRACT </ATT_ignore_case>

is pronounced "this contract"

9.2 Break

The **Break** tag instructs the TTS engine to insert a pause in the synthesized text in one of three ways.

Syntax: <BREAK/>

Languages: US English, UK English, Spanish, German, French

Example: Time for a pause <Break/> Okay, keep going.

Inserts a brief break after the word "pause".

Syntax: <BREAK Size="none | small | medium | large"/>

Example: No time for a pause <Break size="none"/> Keep going.

Inserts no break after the word "pause".

Example: Time for a pause <Break size="medium"/> Okay, keep going.

Inserts a brief silence, the equivalent of the silence following a sentence, after the word "pause".

Example: Time for a pause <Break size="large"/> Keep going.

Inserts only the default break after the word "pause".

Example: `Time for a pause <Break size="medium"/> Okay, keep going.`

Inserts the equivalent of a paragraph break of silence after the word "pause".

Syntax: **`<BREAK time="`** *`duration`* **`"/>`**

Example: `Break for 100 milliseconds <Break time="100ms"/> Okay, keep going.`

Inserts 100 milliseconds of silence after the word "milliseconds".

Example: `Break for 3 seconds <Break time="3s"/> Okay, keep going.`

Inserts 3 seconds of silence after the word "seconds".

9.3 Mark

The application may insert a user bookmark in the text. The TTS Engine will optionally inform the application via a notification when that bookmark is reached in audio stream.

Any number of bookmarks may be inserted anywhere in the text.

Syntax: **`<Mark Name="`***`label`***`"/>`** *text*

Languages: `US English, UK English, Spanish, German, French`

where *`label`* is an text label. The TTS engine will generate a bookmark notification with the specified *`label`* when that spot is reached in the audio stream.

Example: `The quick <Mark Name="mark 1"/> fox jumped over the lazy <Mark Name="mark 2"/> dog.`

The TTS engine will provide a notification just before speaking the word "fox" and another notification just before speaking the word "dog".

9.4 Paragraph

The **PARAGRAPH** tag tells the TTS engine to change the prosody to reflect the end of a paragraph, regardless of the surrounding punctuation.

Syntax: **`<PARAGRAPH>`** *`text`* **`</PARAGRAPH>`**

`<P>` *text* **`</P>`**

Languages: `US English, UK English,Spanish, German, French`

Example: **`<Paragraph>`** `This example has only one sentence in the paragraph` **`</Paragraph>`**

Example: **`<`**`P`**`>`** `The paragraph tag can be abbreviated as just the letter P.` **`<`**`/P`**`>`**

The TTS engine changes the prosody to reflect the paragraph boundaries.

9.5 Phoneme

The **PHONEME** tag allows the user to specify pronunciations explicitly in the input text. Including the orthography, i.e. the word or words represented by the phonemes, are optional, but recommended, because some modules in the front end depend on orthography. The pronunciations must be represented using the DARPA phoneme set when using the AT&T Natural Voices TTS Client SDK. See Appendix A for more details about the DARPA phonemes.

Syntax: **<PHONEME alphabet="att_darpabet_english" ph="** *phoneme*$^{+}$ **"/>**

Languages: US English

Example: `<Phoneme alphabet="att_darpabet_english" ph="b ow t 1"/>`

is pronounced "boat"

Syntax: **<PHONEME alphabet="att_sampa_spanish" ph="** *phoneme*$^{+}$ **">** ***orthography*** **</PHONEME>**

Syntax: **<PHONEME alphabet="att_darpabet_english" ph="** *phoneme*$^{+}$ **"/>**

Languages: UK English

Example: `<Phoneme alphabet="att_darpabet_ukenglish" ph="b ow t 1"/>`

is pronounced "boat"

Languages: Spanish

Example: `<Phoneme alphabet="att_sampa_spanish" ph="p a 1 D r e"> padre </Phoneme>`

is pronounced "padre"

Syntax: **<PHONEME alphabet="att_sampa_german" ph="** *phoneme*$^{+}$ **">** ***orthography*** **</PHONEME>**

Languages: German

Example: `<Phoneme alphabet="att_sampa_german" ph="b o: t 1"> boot </Phoneme>`

is pronounced "boot"

NOTE: The AT&T Natural Voices implementation of SSML supports only the DARPA phonemes and does not support the IPA, Worldbet, or X-Sampa phonetic dictionaries. See Appendix A for a complete description of the DARPA phoneme set.

9.6 Prosody

Release 1.4 supports only the Rate and Volume attributes of the SSML Prosody tag. The Pitch and Contour tags are not supported.

9.6.1 Rate

The **RATE** attribute of the **Prosody** tag changes the rate at which the text is spoken. You can specify either the absolute rate or a relative change in the current speaking rate. The Release 1.4 engine supports a range of up to eight times faster or slower than the default speed.

Syntax: **<PROSODY RATE=”fast | medium | slow | default”>** *content* **</PROSODY>**

Syntax: **<PROSODY RATE=”** *relativeChange* **”>** *text* **</PROSODY>**

changes the speaking rate which is expressed in Words Per Minute (WPM). *relativeChange* is an floating point number that is added to the current rate. Using a *relativeChange* < 0 decreases the speed.

Languages: US English, UK English, Spanish, German, French

Example:
```
This is the default speed
<prosody rate=”slow”> this is speaking slowly
   <prosody rate=”fast”>  this is speaking fast
</prosody>
    back to slow
</prosody>
back to the default rate
```

Example:
```
This is the default speed
<prosody rate="-50%">
    this is 50% slower
    <prosody rate="+100%">  this is 50% faster than
    normal  </prosody>
    back to 50% slower
</prosody>
back to the default rate
```

9.6.2 Volume

The **Volume** attribute of the **Prosody** tag allows the application to change the volume of the TTS voice. Note that this does not change the volume of the output device, but it does raise or lower the volume of the text spoken within the context of the tag.

Syntax: **<PROSODY VOLUME=”** *level* **”>** *text* **</PROSODY>**

where *level* is a value from 0.0 to 200.0. A value of 100 is the voice’s default volume, a value of 0 changes the volume to 0 and a value of 200 doubles the volume. The volume changes linearly.

Syntax: **<PROSODY VOLUME=” silent | soft | medium | loud | default ”>** *text* **</PROSODY>**

Sets the absolute volume to the specified level.

Syntax: **<PROSODY VOLUME=”** *delta* **”>** *text* **</PROSODY>**

where *delta* is a floating point value which is added to the current volume setting.

Languages: US English, UK English, Spanish, German, French

Example:
```
This is the default volume
<prosody volume="silent"> silence </prosody>
<prosody volume="soft"> Now I'm whispering </prosody>
<prosody volume="+20%"> a little louder </prosody>
<prosody volume="medium"> medium volume</prosody>
<prosody volume="+20%"> a little louder </prosody>
<prosody volume="loud"> very loud </prosody>
<prosody volume="default"> back to default volume </
prosody>
```

9.7 Say-As

Say-As tags provide contextual hints to the TTS engine about how text should be pronounced. The TTS engine supports a number of different contexts that can be used to fine-tune the pronunciation of words.

9.7.1 Acronym

The **Acronym** context tells the TTS engine to treat the text as an acronym and to úpronounce the text as the letters in the words.

Syntax: **<SAY-AS Type="Acronym">** ***text*** **</SAY-AS>**

Languages: US English

Example: MADD `<Say-as type="acronym"> MADD </Say-as>`

is pronounced "mad M-A-D-D". This tag is especially useful if your text is mostly upper case and you use the **ATT_Ignore_Case** tag but then encounter an acronym.

9.7.2 Address

The **Address** context tells the TTS engine to treat the text as an address.

Syntax: **<SAY-AS Type="Address">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example:
```
<Say-as Type="Address">
123 Main St.
New York, NY 10017
</Say-as>
```

is pronounced "one twenty three main street, New York, New York one zero zero one seven"

9.7.3 ATT_Literal

The **ATT_Literal** context instructs the TTS engine to pass the string through literally without applying additional context processing such as abbreviations, addresses, dates, math symbols, measurements, names, numbers, times, or phone numbers.

Syntax: **<SAY-AS Type="ATT_Literal">** *text* **</SAY-AS>**

Languages: US English

Example: `<Say-as type="ATT_Literal"> misc. </Say-as> misc.`

is pronounced “misk miscellaneous”

9.7.4 ATT_Math

The **ATT_Math** context tells the TTS engine to treat the text as a mathematical expression.

Syntax: <SAY-AS Type=”ATT_Math”> *text* **</SAY-AS>**

Languages: US English

Example: <Say-as Type=”ATT_Math”> 3+4=7 </Say-as> 3+5=8

is pronounced “three plus four equals seven three plus sign five equal sign eight”

9.7.5 ATT_Measurement

The **ATT_Measurement** context tells the TTS engine to treat the text as a measurement, e.g. single quotes are pronounced “feet” and double quotes are pronounced “inches”.

Syntax: <SAY-AS Type=”ATT_Measurement”> *text* **</SAY-AS >**

Languages: US English

Example: <Say-as Type=”ATT_Measurement”> 5’ 3” </Say-as> 7’ 9”

is pronounced “five feet three inches seven single quote nine double quote”

9.7.6 Currency

The **Currency** context tells the TTS engine to treat the text as currency and expand $ and decimal numbers appropriately. The Currency Context works only with US currency and not other currencies.

Syntax: <SAY-AS Type=”Currency”> *text* **</SAY-AS>**

Languages: US English

Example: <Say-as Type=”Currency”> $25.32 </Say-as>

is pronounced “twenty five dollars and thirty two cents”

9.7.7 Date

The **Date** context tells the TTS engine to treat the text as date. You may also add qualifiers to provide even more information to the TTS engine but in general the extra qualifiers are not needed.

Syntax: <SAY-AS Type=”Date”> *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type=”Date”> Dec 25, 2001 </Say-as>

is pronounced “December twenty fifth two thousand one”

Syntax: <SAY-AS Type=”Date:M”> *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type=”Date:M”> Dec </Say-as>

is pronounced “December”

Syntax: **<SAY-AS Type="Date:MD">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type="Date:MD"> Dec 25</Say-as>
is pronounced "December twenty fifth"

Syntax: **<SAY-AS Type="Date:MDY">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type="Date:MDY"> Dec 25, 2001 </Say-as>
is pronounced "December twenty fifth two thousand one"

Syntax: **<SAY-AS Type="Date:MY">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type="Date:MY"> Dec, 2001 </Say-as>
is pronounced "December two thousand one"

Syntax: **<SAY-AS Type="Date:Y">** *text* **</SAY-AS>**

Languages: US English

Example: <Say-as Type="Date:Y"> 2001 </Say-as>
is pronounced "two thousand one"

9.7.8 Name

AT&T Natural Voices TTS engine does a great job on names without XML tags but you can tell the engine to expect an name with the **SAY-AS** name tag.

Syntax: **<SAY-AS Type="Name">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type="Name"> Mark Beutnagel </Say-as>
is pronounced "Mark Beutnagel"

9.7.9 Net

The **Net** type tells the engine to expect either an email address or a URL.

Syntax: **<SAY-AS Type="Net">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as Type="Net"> help@naturalvoices.att.com </Say-as>
is pronounced "help at natural voices dot ATT dot com"

Example: <Say-as Type="Net"> http://naturalvoices.att.com </Say-as>
is pronounced "H T T P natural voices dot ATT dot com"

Syntax: **<SAY-AS Type="Net:email">** *text* **</SAY-AS>**

Languages: US English, Spanish, German

Example: <Say-as Type="Net:email"> help@naturalvoices.att.com </Say-as>
is pronounced "help at natural voices dot ATT dot com"

Syntax: **<SAY-AS Type="Net:URL">** *text* **</SAY-AS>**

Example: <Say-as Type="Net:URL"> help@naturalvoices.att.com </Say-as>

is pronounced "help at natural voices dot ATT dot com"

9.7.10 Number

The **Number** type tells the engine to expect a number.

Syntax: **<SAY-AS type="Number">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as type="Number"> 10,243 </Say-as>

is pronounced "ten thousand two hundred forty three"

Syntax: **<SAY-AS type="Number:Decimal">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as type="Number:decimal"> 3.14159 </Say-as>

is pronounced "three point one four one five nine"

Syntax: **<SAY-AS type="Number:Fraction">** *text* **</SAY-AS>**

Languages: US English

Example: <Say-as type="Number:Fraction"> 5 3/4 </Say-as>

is pronounced "five and three fourths"

Syntax: **<SAY-AS type="Number:Ordinal">** *text* **</SAY-AS>**

Languages: US English

Example: <Say-as type="Number:ordinal"> VI </Say-as>

is pronounced "sixth"

9.7.11 Sub

The **SUB** tag allows you to substitute spoken text for the written text.

Syntax: **<SAY-AS sub="*spoken*">** *text* **</SAY-AS>**

Languages: US English, UK English, Spanish, German, French

Example: <Say-as sub=" Mothers Against Drunk Driving"> MADD </Say-as>

is pronounced "mothers against drunk driving"

9.7.12 Telephone

The **Telephone** context tells the TTS engine to treat the text as a telephone number.

Syntax: **<SAY-AS type="Telephone">** *text* **</SAY-AS>**

Languages: US English

Example: <Say-as type="telephone"> (212)555-1212 </Say-as>

is pronounced "two one two five five five one two one two"

9.7.13 Time

The **Time** context tells the TTS engine to treat the text as a time.

Syntax: <SAY-AS Type="Time"> *text* </SAY-AS>

Languages: US English

Example: <SAY-AS type="Time"> 12:34 PM </SAY-AS>
is pronounced "twelve thirty four P M"

Syntax: <SAY-AS Type="Time:HMS"> *text* </SAY-AS>

Languages: US English

Example: <SAY-AS type="Time"> 12:34:56 PM </SAY-AS>
is pronounced "twelve thirty four and fifty six seconds P M"

Syntax: <SAY-AS Type="Time:HM"> *text* </SAY-AS>

Languages: US English

Example: <SAY-AS type="Time"> 12:34 PM </SAY-AS>
is pronounced "twelve thirty four P M"

Syntax: <SAY-AS Type="Time:H"> *text* </SAY-AS>

Languages: US English

Example: <SAY-AS type="Time"> 12 PM </SAY-AS>
is pronounced "twelve P M"

9.8 Sentence

The **SENTENCE** tag tells the TTS engine to change the prosody to reflect the end of a sentence, regardless of the surrounding punctuation.

Syntax: <SENTENCE> *text* **</SENTENCE>**

<S> *text* **</S>**

Languages: US English, UK English, Spanish, German, French

Example: <Sentence> This text is a sentence. </Sentence>

Example: <S> The sentence tag can be abbreviated as just the letter S. </S>

The TTS engine changes the prosody to reflect the sentence boundaries.

9.9 Speak

The **SPEAK** tag tells the TTS engine to synthesize the specified text. You need not include this tag to have your text synthesized.

Syntax: <SPEAK> *text* **</SPEAK>**

Languages: US English, UK English, Spanish, German, French

Example: <Speak> This text is synthesized. </Speak>

is pronounced "this text is synthesized."

9.10 Voice

The **Voice** tag allows the application to change the voice of the TTS speaker from the input text. You can use this feature to change voices, e.g. you might use different voices to speak different sections of an email message or carry on a conversation between two different voices. This release includes both a US English male voice called "Mike" and a US English female voice called "Crystal".

The default voice for a server is specified when the server process is started (see the Server Command Line Options for details).

You choose a voice by specifying one or more of the following attributes:

- **Gender**: `male, female,` or `neutral`
- **Age**: an integer value, e.g. 30
- **Category**: `child`, `teen`, `adult`, or `elder`
- **Language**: `"en_us" or "English"` for US English, "`wn_uk`" or `"uk_english"` for UK English, `"ES_US" or "spanish"` for Spanish, "`de_de`" or "`German`" for German, and "`fr_fr`" or "`French`" for French
- **Name**: `mike`, `crystal, rosa, rich, claire, reiner, klara, Audrey, charles, alain,` or other voices you've installed.

You may specify several attributes but in Release 1.4, it's best to specify the speaker by name. You may purchase additional voices which may make more use of the additional attributes.

Syntax: <VOICE

Gender="male | female | neutral"

Age="*integer***"**

Category="child | teen | adult | elder"

Language="en_us" or "English" for US English

"en_uk" or "UKEnglish" for UK English

"es_us" or "Spanish" for Spanish

"de_de" or "German" for German

"fr_fr" or "French" for French

Name="mike | crystal | rich | rosa | reiner | Klara | Audrey | charles | alain"

</VOICE>

Languages: `US English, UK English, Spanish, German, French`

Example: `<Voice Name="mike"> Hi, I'm Mike </Voice>`

is pronounced, "Hi, I'm Mike" using Mike's 8KHz voice.

Example: `<Voice Name="crystal"> I'm Crystal </Voice>`

is pronounced, "I'm Crystal" using Crystal's 8KHz voice.

Example: `<Voice Name="rosa"> Hola, me llamo Rosa </Voice>`

is pronounced, "Hola, me llamo Rosa" using Rosa's 8KHz voice.

Example: `<voice name="mike">`

```
        This is Mike
        <Voice Name=”crystal”> This is Crystal </voice>
        This is Mike again.
</voice>
```

This string is pronounced in Mike’s voice “This is Mike”, then in Crystal’s voice, “This is Crystal”, then in Mike’s voice, “This is Mike again”.

The optional 16KHz voices are accessed by specifying the 16KHz voice name, e.g. mike16, crystal16, claire16, rich16, rosa16, reiner16, or klara16.

Example: `<Voice Name=”mike16”> Hi, I’m 16KHz Mike </Voice>`

is pronounced, “Hi, I’m 16 KHz Mike” using Mike’s 16KHz voice.

Specifying the voice by name is the best way to get the speaker and language that you want.

Note that switching voices forces the server to load the data files for the each voice that is specified which may result in noticeable delays. Voice switches will happen instantaneously once the voice data is in place.

10 JSML Control Tags

The AT&T Natural Voices TTS engine does a great job of synthesizing most text without special instructions but there may be special circumstances where you may wish to provide hints to fine-tune the pronunciation of certain words or phrases. The AT&T Natural Voices TTS engine allows users to mark up the text to be spoken to include special control tags that change the way the text is pronounced. Four different speech platforms and markup languages are supported: Microsoft SAPI 4.0, Microsoft SAPI 5.1, the Speech Synthesis Markup Language (SSML) component of Voice XML, and JSML, the Java Speech Markup Language.

The AT&T Natural Voices TTS engine supports a subset of the JSML control. Not all tags are supported in all languages. The following table specifies which tags are supported and in which languages:

JSML Tag	US English	UK English	German	Spanish	French
<jsml [lang=*language*] [mark="*mark*"> *text* </jsml>	Yes	Yes	Yes	Yes	Yes
<div type="paragraph | para" [mark="*mark*"]> *text*	Yes	Yes	Yes	Yes	Yes
<div type="sentence | sent" [mark="mark"]> *text*	Yes	Yes	Yes	Yes	Yes
<voice [gender="male | female | neutral"] [age="child | teenager | younger_adult | middle_adult | older_adult | adult"] [variant="variant"] [name="speaker"] [mark="mark"] > text </voice>	Yes	Yes	Yes	Yes	Yes
<sayas class="literal" [mark="*mark*"] > *text* </sayas>	Yes	No	No	No	No
<sayas class ="date"	Yes	Yes	Yes	Yes	Yes

[mark="*mark*"] > *text* </sayas>					
<sayas class ="time" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<sayas class ="name" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<sayas class ="phone" [mark="*mark*"] > *text* </sayas>	Yes	No	No	No	No
<sayas class ="net" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<sayas class ="address" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<sayas class ="currency" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<sayas class ="measure" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<sayas class="number" [mark="*mark*"] > *text* </sayas>	Yes	Yes	Yes	Yes	Yes
<phoneme original="*phonemeString*" [mark="*mark*"] > *text* </phoneme>	Yes	Yes	Yes	Yes	Yes
<emphasis level [mark="*mark*"] > *text* </emphasis>	No	No	No	No	No
<break time="*duration*" [mark="*mark*"]/ >	Yes	Yes	Yes	Yes	Yes
<break size="none | small | medium | large" [mark="*mark*"] />	Yes	Yes	Yes	Yes	Yes
<prosody rate="*rate*" [mark="*mark*"] > *text* </ prosody>	Yes	Yes	Yes	Yes	Yes
<prosody volume="*volume*" [mark="*mark*"] > *text* </prosody>	Yes	Yes	Yes	Yes	Yes
<prosody pitch> *text* </prosody>	No	No	No	No	No
< prosody range> *text* </prosody>	No	No	No	No	No
<marker mark="*mark*"> *text* </marker>	Yes	Yes	Yes	Yes	Yes
<engine name="*name*" [mark="*mark*"] > *text* </engine>	Yes	Yes	Yes	Yes	Yes
<engine data [mark="*mark*"] > *text* </engine>	Yes	Yes	Yes	Yes	Yes

NOTE: AT&T Natural Voices TTS SDK treats control tags as case-insensitive.

10.1 jsml

The **jsml** tag tells the TTS that the document contains JSML instructions.

Syntax: **<jsml [lang="***language***"] [mark="***mark***"]>** *text* **</jsml>**

Languages: US English, UK English, Spanish, German, French

Example: <jsml mark="beginning of document"> This document has only one line </jsml>

10.2 div

The **div** tag divides the text into paragraphs and sentences to provide hints to the TTS engine to improve prosody by including appropriate pauses between sentences and paragraphs.

Syntax: **<div type="paragraph" [mark="***mark***"]>** *text* ** **

<div type="para" [mark="*mark***"]>** *text* ** **

<div type="sentence" [mark="*mark***"]>** *text* ** **

<div type="sent" [mark="*mark***"]>** *text* ** **

Languages: US English, UK English, Spanish, German, French

Example: This example has only one sentence in the paragraph

Example: **<**div type="paragraph" mark="paragraph1"**>** The paragraph tag can include an optional mark tag which generates a bookmark event when the paragraph is synthesized. **<**/div**>**

Example: **<**div type="para"**>** Sentences can be embedded inside paragraphs.   The div tag can include an optional MARK tag which generates a bookmark event when the sentence is synthesized. **<**/div**>**

10.3 voice

The **voice** tag allows the application to change the voice of the TTS speaker from the input text. You can use this feature to change voices, e.g. you might use different voices to speak different sections of an email message or carry on a conversation between two different voices. A variety of voices are supported in US English, Spanish, and German.

The default voice for a server is specified when the server process is started (see the Server Command Line Options for details).

You choose a voice by specifying one or more of the following attributes:

- **gender**: male, female, or neutral
- **age**: an integer value, e.g. 30 or child, teenager, younger_adult, middle_adult, older_adult, or adult
- **variant**: this may be specified but it is not used in selecting a voice
- **Language**: "en_us" or "English" for US English or "ES_US" or "spanish" for Spanish, "de_de" or "German" for German
- **Name**: mike, crystal, rosa, rich, claire, reiner, klara, mike16, crystal16, rosa16, rich16, claire16, reiner16, klara16, audrey, audrey16, charles, charles16, alain, alain16 or other voices you've installed.

You may specify several attributes but in Release 1.4, it's best to specify the speaker by name. You may purchase additional voices which may make more use of the additional attributes.

Syntax: **<voice**

[gender="male | female | neutral"]

[age="*integer* **| child | teenager | younger_adult | middle_adult | older_adult | adult"]**

[name="mike | crystal | rich | claire | rosa | reiner | klara | audrey | charles | alain | mike16 | crystal16 | rich16 | claire16 | rosa16 | reiner16 | klara16 | audrey16 | charles16 | alain16"]

[mark="*mark***"]**

</voice>

Languages: US English, UK English,Spanish, German, French

Example: <voice name="mike"> Hi, I'm Mike </voice>

is pronounced, "Hi, I'm Mike" using Mike's 8KHz voice.

Example: <voice name="crystal"> I'm Crystal </voice>

is pronounced, "I'm Crystal" using Crystal's 8KHz voice.

Example: <voice name="rosa"> Hola, me llamo Rosa </voice>

is pronounced, "Hola, me llamo Rosa" using Rosa's 8KHz voice.

Example:
```
<voice name="mike">
    This is Mike
    <voice Name="crystal"> This is Crystal </voice>
    This is Mike again.
</voice>
```

This string is pronounced in Mike's voice "This is Mike", then in Crystal's voice, "This is Crystal", then in Mike's voice, "This is Mike again".

The optional 16KHz voices are accessed by specifying the 16KHz voice name, e.g. mike16, crystal16, claire16, rich16, rosa16, reiner16, or klara16.

Example: <voice name="reiner16"> Ich bin Reiner </voice>

is pronounced, "Ich bin Reiner" using Reiner's 16KHz voice.

Specifying the voice by name is the best way to get the speaker and language that you want.

Note that switching voices forces the server to load the data files for the each voice that is specified which may result in noticeable delays. Voice switches will happen instantaneously once the voice data is in place.

10.4 sayas

The variants of the **sayas** tag provide hints to the synthesizer to help provide the best sounding output.

10.4.1 literal

The **literal** context tells the TTS engine to treat the text as individual characters and spell the words.

Syntax: <sayas class="literal" [mark="*mark*"] > *text* </sayas>

Languages: US English

Example: <sayas class="literal"> abc 123 </sayas>

is pronounced "A B C 1 2 3"

10.4.2 date

The **Date** context tells the TTS engine to treat the text as date. You may also add qualifiers to provide even more information to the TTS engine but in general the extra qualifiers are not needed.

Syntax: <sayas class="date" [mark="*mark*"] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French

Example: <SAYAS class="date"> Dec 25, 2001 </sayas>

is pronounced "December twenty fifth two thousand one"

Syntax: <sayas class="date:mdy" [mark="*mark*"] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French

Example: <sayas class="date"> Dec 25, 2001 </sayas>

is pronounced "December twenty fifth two thousand one"

Syntax: <sayas class="date:m" [mark="*mark*"] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French

Example: <sayas class="date:m"> Dec </sayas>

is pronounced "December"

Syntax: <sayas class="date:md" [mark="*mark*"] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French
Example: <sayas class="date:md"> Dec 25</sayas>

is pronounced "December twenty fifth"

Syntax: <sayas class="date:mdy" [mark="*mark*"] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French
Example: <sayas class="date:mdy"> Dec 25, 2001 </sayas>

is pronounced "December twenty fifth two thousand one"

Syntax: <sayas class="date:my" [mark="*mark*"] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French
Example: <sayas class="date:my"> Dec, 2001 </sayas>

is pronounced “December two thousand one”

Syntax: <sayas class=”date:y” [mark=”*mark*”] > *text* </sayas>

Languages: US English

Example: <sayas class=”date:y”> 2001 </sayas>

is pronounced “two thousand one”

10.4.3 time

The **time** context tells the TTS engine to treat the text as a time.

Syntax: <sayas class=”time” [mark=”*mark*”] > *text* </sayas>

Languages: US English

Example: <SAY-AS type=”time”> 12:34 PM </sayas>

is pronounced “twelve thirty four P M”

Syntax: <sayas class=”time:hms” [mark=”*mark*”] > *text* </sayas>

Languages: US English

Example: <sayas class=”Time”> 12:34:56 PM </sayas>

is pronounced “twelve thirty four and fifty six seconds P M”

Syntax: <sayas class=”time:hm” [mark=”*mark*”] > *text* </sayas>

Languages: US English

Example: <sayas class=”time”> 12:34 PM </sayas>

is pronounced “twelve thirty four P M”

Syntax: <sayas class=”time:h” [mark=”*mark*”] > *text* </sayas>

Languages: US English

Example: <sayas class=”time”> 12 PM </sayas>

is pronounced “twelve P M”

10.4.4 name

AT&T Natural Voices TTS engine does a great job on names without XML tags but you can tell the engine to expect an name with the **sayas name** tag.

Syntax: <sayas class=”name” [mark=”*mark*”] > *text* </sayas>

Languages: US English, UK English, Spanish, German, French

Example: <sayas class=”name”> Mark Beutnagel </sayas>

is pronounced “Mark Beutnagel”

10.4.5 phone

The **phone** context tells the TTS engine to treat the text as a telephone number.

Syntax: <sayas class=”phone” [mark=”*mark*”] > *text* </sayas>

Languages: US English

Example: <sayas class=”phone”> (212)555-1212 </sayas>

is pronounced “two one two five five five one two one two”

10.4.6 net

The **net** type tells the engine to expect either an email address or a URL.

Syntax: **<sayas class="net"** [**mark**="*mark*"] **>** *text* **</sayas>**

Languages: US English, UK English, Spanish, German, French

Example: <sayas class="net"> help@naturalvoices.att.com </sayas>

is pronounced "help at natural voices dot ATT dot com"

Example: <sayas class="net"> http://naturalvoices.att.com </sayas>

is pronounced "H T T P natural voices dot ATT dot com"

Syntax: **<sayas class="net:email"** [**mark**="*mark*"] **>** *text* **</sayas>**

Languages: US English, UK English, Spanish, German, French
Example: <sayas class="net:email"> help@naturalvoices.att.com </sayas>

is pronounced "help at natural voices dot ATT dot com"

Syntax: **<sayas class="net:URL"** [**mark**="*mark*"] **>** *text* **</sayas>**

Example: <sayas class="net:URL"> help@naturalvoices.att.com </sayas>

is pronounced "help at natural voices dot ATT dot com"

10.4.7 address

The **address** context tells the TTS engine to treat the text as an address.

Syntax: **<sayas class="address">** *text* **</sayas>**

Languages: US English, UK English, Spanish, German, French
Example: <sayas class="address">

```
123 Main St.
New York, NY 10017
</sayas>
```

is pronounced "one twenty three main street, New York, New York one zero zero one seven"

10.4.8 currency

The **currency** context tells the TTS engine to treat the text as currency and expand $ and decimal numbers appropriately. The Currency Context works only with US currency and not other currencies.

Syntax: **<sayas class="currency"** [**mark**="*mark*"] **>** *text* **</sayas>**

Languages: US English

Example: <sayas class="currency"> $25.32 </sayas>

is pronounced "twenty five dollars and thirty two cents"

10.4.9 measure

The **measure class** tells the TTS engine to treat the text as a measurement, e.g. single quotes are pronounced “feet” and double quotes are pronounced “inches”.

Syntax: **<sayas class=”measure”** [**mark**=”*mark*”] **>** *text* **</sayas >**

Languages: US English

Example: <sayas class=”measure”> 5’ 3” </sayas> 7’ 9”

is pronounced “five feet three inches seven single quote nine double quote”

10.4.10 number

The **number** context tells the TTS engine to treat the text as a number. You may also add qualifiers to provide even more information to the TTS engine but in general the extra qualifiers are not needed.

Syntax: **<sayas class=”number”** [**mark**=”*mark*”] **>** *text* **</sayas>**

Languages: US English, UK English, Spanish, German, French

Example: <sayas class=”number”> 12,345 </sayas>

is pronounced “twelve thousand three hundred forty five”

10.5 phoneme

The **phoneme** tag allows the user to specify pronunciations explicitly in the input text. Including the orthography, i.e. the word or words represented by the phonemes, are optional, but recommended, because some modules in the front end depend on orthography. The pronunciations must be represented using the IPA phoneme set when using the AT&T Natural Voices TTS Client SDK. See Appendix A for more details about the IPA phonemes.

Syntax: **<phoneme [original=”***orthography***”] *[mark=”mark”]* *phoneme*$^{+}$ *</phoneme>***

Languages: US English, UK English, Spanish, German, French

Example: <phoneme original=”boat”> b ow t </phoneme>

is pronounced “boat”

Example: <phoneme original=”padre”> p a d r e </phoneme>

is pronounced as the Spanish word “padre”

Example: <phoneme original=”deutsch”> d OY tS </phoneme>

is pronounced as the German word “deutsch”

10.6 break

The **break** tag instructs the TTS engine to insert a pause in the synthesized text in one of three ways.

Syntax: **<break [mark=”***mark***”] />**

Languages: US English, UK English, Spanish, German, French

Example: `Time for a pause <break/> Okay, keep going.`

Inserts a brief, medium size break after the word “pause”.

Syntax: **`<break size=”none | small | medium | large” [mark=”`***`mark`***`”] />`**

Example: `No time for a pause <break size=”none”/> Keep going.`

Inserts no break after the word “pause”.

Example: `Time for a pause <break size=”small”/> Okay, keep going.`

Inserts a brief (250 millisecond) silence, the equivalent of the silence following a sentence, after the word “pause”.

Example: `Time for a pause <break size=”medium”/> Okay, keep going.`

Inserts a brief one second silence, the equivalent of the silence following a sentence, after the word “pause”.

Example: `Time for a pause <break size=”large”/> Keep going.`

Inserts a two second break after the word “pause”.

Syntax: **`<BREAK time=”`** ***`duration`*** **`” [MARK=”`***`mark`***`”] />`**

Example: `Break for 100 milliseconds <BREAK time=”100ms”/> Okay, keep going.`

Inserts 100 milliseconds of silence after the word “milliseconds”.

Example: `Break for 3 seconds <BREAK time=”3s”/> Okay, keep going.`

Inserts 3 seconds of silence after the word “seconds”.

10.7 prosody

The **prosody** tag allows the user to control the rate and volume of the synthesized voice.

10.7.1 rate

The **rate** attribute of the **prosody** tag, defined in words per minute, changes the rate at which the text is spoken. You can specify either the absolute rate or a relative change in the current speaking rate. The Release 1.4 engine supports a range of up to eight times faster or slower than the default speed.

Syntax: **`<prosody rate=”`** ***`n`*** **`” [mark=”`***`mark`***`”]>`** ***`text`*** **`</prosody>`**

where *n* is an integer or floating point number sets the speaking rate to *n* words per minute. *N* can take on several different forms:

n – sets the speaking rate to the specified value

+*n* – increases the speaking rate by n words per minute

-*n* – decreases the speaking rate by n words per minute

+n% - increases the speaking rate by n%

-n% - decreases the speaking rate by n%

fast, **medium**, **slow**, or **default** – sets the speaking rate as specified.

Languages: US English, UK English, Spanish, German, French

Example: <prosody rate="150**"> Speak 150 words per minute </prosody>**

sets the speaking rate to *150* words per minute.

Example: <prosody rate="+50**"> Speak faster </prosody>**

increases the speaking rate by 50 words per minute.

Example: <prosody rate="-50**"> Speak more slowly </prosody>**

decreases the speaking rate by 50 words per minute.

Example: <prosody rate="+10%**"> Speak more quickly </prosody>**

increases the speaking rate by 10%.

Example: <prosody rate="-10%**"> Speak more slowly </prosody>**

decreases the speaking rate by 10% words per minute.

Example: <prosody rate="default**"> Spoken at the default rate </prosody>**

Restores the speaking rate to the default words per minute.

10.7.2 volume

The **volume** attribute of the **prosody** tag, defined as a floating point value between 0.0 and 1.0 inclusive, changes the volume at which the text is spoken. You can specify either the absolute volume or a relative change in the current speaking volume.

Syntax: <prosody volume=" *n* **"> *content* </prosody>**

where *n* is an integer or floating point number sets the volume to *n*. *N* can take on several different forms:

n – sets the volume to the specified value

+n – increases the volume by *n*

-n – decreases the volume by *n*

+n% - increases the volume by *n*%

-n% - decreases the volume by *n*%

- **loud**, **medim**, **quiet,** and **default**– sets the volume

Languages: US English, UK English, Spanish, German, French

Example: <prosody volume="0.5**"> This is the middle volume </prosody>**

Example: <prosody volume="+.2**"> Speak a little louder </prosody>**

Example: <prosody volume=”-.3”> Speak a little more softly **</prosody>**

Example: <prosody volume=”+10%”> Speak 10% louder **</prosody>**

Example: <prosody volume=”-10%”> Speak 10% more softly **</prosody>**

Example: <prosody volume=”reset”> Spoken at the default volume **</prosody>**

10.8 marker

The application may insert a user bookmark in the text. The TTS Engine will optionally inform the application via a notification when that bookmark is reached in audio stream.

Any number of bookmarks may be inserted anywhere in the text.

Syntax: <marker mark=”*label*”/> *text*

Languages: US English, UK English, Spanish, German, French

where *label* is an text label. The TTS engine will generate a bookmark notification with the specified *label* when that spot is reached in the audio stream.

Example: The quick <marker mark=”mark 1”/> fox jumped over the lazy <marker mark=”mark 2”/> dog.

The TTS engine will provide a notification just before speaking the word “fox” and another notification just before speaking the word “dog”.

10.9 engine

The **engine** tag allows the input text to take advantage of features that are unique to a particular vendor’s TTS engine.

Syntax: <engine name=”name[,name2] data=”data” mark=”*label*”> *text* ***</engine>***

Languages: US English, UK English, Spanish, German, French

Where name is any of “AT&T NaturalVoices”, “ATT NaturalVoices”, “NaturalVoices”, “AT&T”, or “ATT”.

Example: You are listening to <element name=”ATT” data=”the best”> an inferior </engine> TTS product.

Is pronounced “You are listening ot the best TTS product” if you are using the AT&T Natural Voices engine or “You are listening to an inferior TTS product” with any other product.

11 Custom Dictionaries

The AT&T Natural Voices TTS engine allows you to define custom dictionaries that change the default pronunciation of words. The AT&T Natural Voices TTS engine does a great job pronouncing most words and even most names correctly but there are some words that even we can't get right. Custom dictionaries allow you to change pronunciations of words for your users or application.

11.1 Defining Custom Pronunciations

The first step in building custom dictionaries is to define new pronunciations for words or phrases using the DARPA phonemes for English dictionaries, the SAMPA phonemes for Spanish dictionaries, or the SAMPA phonemes for German. For example, say that you want the TTS engine to say "toe MAH toe" rather than "toe MAY toe". The first step is to identify the set of DARPA phonemes.

Finding just the right set of phonemes to make a word sound just right can be tricky. Microsoft Windows users can use the WinVoiceEdit GUI tool that is included with the SDK. Another way to do this is to check the samples in Appendix A to find words that sound like the word you wish to say, then experiment using the SAPI <PRON> tag or the SSML <PHONEME> tag to experiment until you find the pronunciation you want. E.g.

<pron sym=" t ow m ey 1 t ow" /> is pronounced "toe MAY toe"

and

<pron sym=" t ow m aa 1 t ow" /> is pronounced "toe MAH toe"

Adding stress marks can also help the TTS engine pronounce the word just the right way. You can specify that phonemes should be unstressed (0), have primary (1) or secondary stress (2), e.g.

<pron sym=”f ax 0 n aa 1 m aa 0 n aa n 2”> phenomenon </pron>

Finding the proper set of phonemes to say a word in just the right way can be challenging. A colleague with expertise in linguistics is a great resource for this problem. The TTS engine does not expose the system lexicon to applications. An upcoming release provides tools to make this procedure much simpler.

11.2 Using the Win32 WinDictEdit Tool

Microsoft Windows users can take advantage of the WinDictEdit graphical editor to create files with custom pronunciations. You’ll find WinDictEdit from the start menu

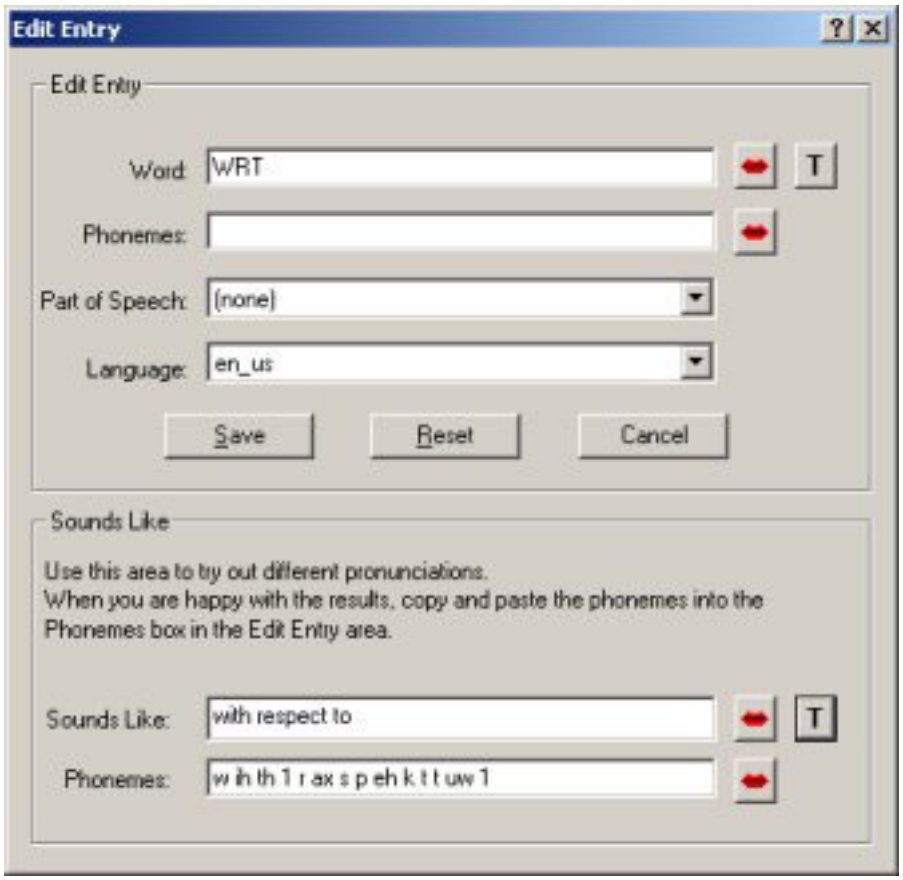
```
Start->Programs->AT&T Natural Voices 1.4->Desktop->Dictionary Editor

Start->Programs->AT&T Natural Voices 1.4->ServerLite->Dictionary Editor

Start->Programs->AT&T Natural Voices 1.4->ClientServer->Dictionary Editor
```

The Dictionary Editor can be used to change the pronunciation of words or allow you to easily add a new abbreviation. We’ll describe two different examples to explain how to use the Dictionary Editor to do both.

11.2.1 Defining Replacements

In this example, we'll demonstrate how to add a new transcription for the word "WRT" which we want to synthesize as "with respect to". This same technique works for other words as well such as "NE" for "north east", "MPH" for "miles per hour", etc.

First, start the Dictionary Editor and either open an existing dictionary file or create a new file. Next, create a new word from the Edit menu. This will pop up a dialog as shown below.



The top portion of the dialog box describes the word that is being added or changed in the dictionary. The bottom portion of the dialog can be used to experiment with different pronunciations. The red "mouth/lips" button allows you to hear the how the word or list of phonemes will be pronounced. The 'T' button displays the phonemes for the word in the "word" or "sounds like" text box.

Adding replacements is very simple:

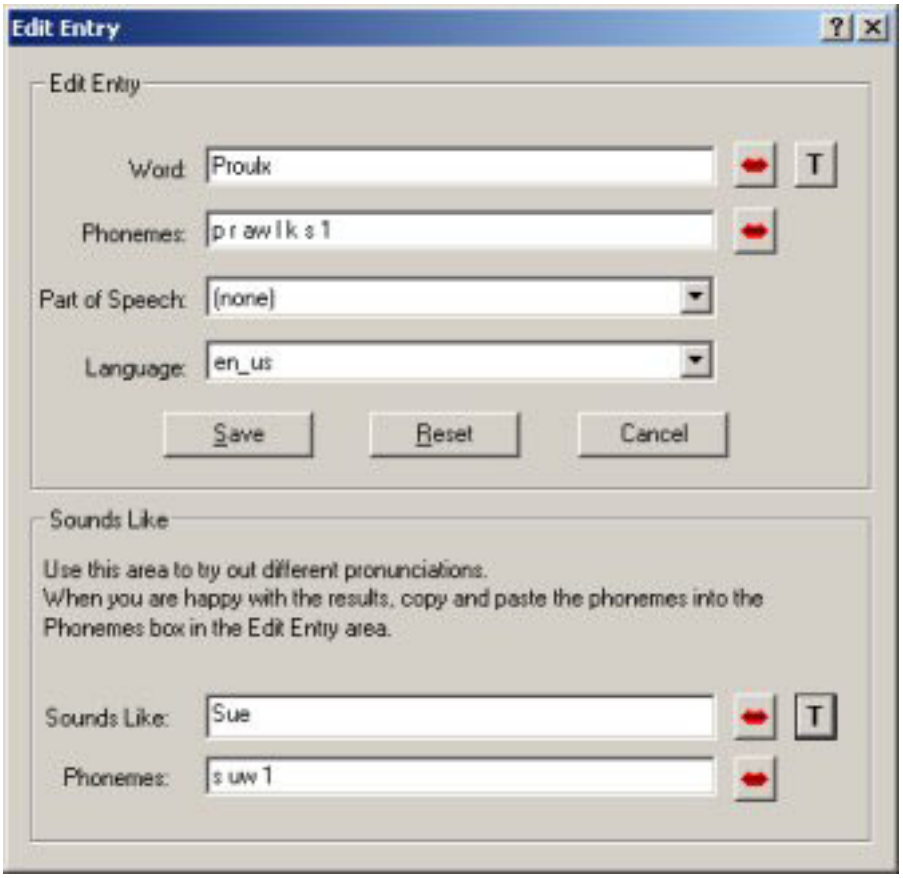
1. Type in the word to be expanded in the "word" text box in the top section of the dialog, "WRT" in our example.
2. Type in the replacement text in the bottom "Sounds Like" text box in the bottom section of the dialog, e.g. "with respect to".
3. click the "T" button next to the "Sounds Like" text box to retrieve the list of phonemes for the text in the "Sounds Like" text box.

4. Copy the phonemes from the “Phonemes” text box in the “Sounds Like” section at the bottom of the dialog and paste the phonemes into the phonemes text box in the “Edit Entry” section at the top of the dialog.
5. Click the red “mouth/lips” button next to the phonemes in the top part of the dialog to hear how the word “WRT” will be pronounced.
6. Click the “save” button and you’ve defined a new pronunciation.

Next step is to tell the TTS engine about your dictionary file, but first we’ll explore changing the pronunciation of a word.

11.2.2 Changing the default pronunciation of a word

In this example, we’ll change the pronuncation of the name “Proulx” from the default which sounds like “prowlx” to sound like “Pru” which rhymes with “Sue”. This time, we’re going to use the WinDictEdit tool to replace a few of the phonemes in the default pronunication of the word.



Here’s the procedure:

1. Type in the word to be expanded in the “word” text box in the top section of the dialog, “Proulx” in our example

2. Click the red “mouth/lips” button next to the word in the top part of the dialog to hear how the word “Proulx” will be pronounced.
3. Click the “T” button next to the word “Proulx” in the top part of the dialog to see the phonemes for the word “Proulx” which are “p r aw l k s 1”. Note that the first part of the pronunciation is correct so we can use the “p r” phonemes but need to replace the “aw l k s 1” with something that rhymes with “Sue”.
4. Next, we need to identify the phonemes that make up the “ue” in “Sue” so type “Sue” in the “Sounds like” text box at the bottom of the dialog.
5. Click the “T” button next to the “Sounds Like” text box to retrieve the list of phonemes “s uw 1” for the text “Sue” in the “Sounds Like” text box. The “uw” phoneme provides the “ue” sound in “Sue”. You can verify this by deleting the ‘s’ phoneme from the phonemes text box in the “Sounds Like” section and then click the “lips/mouth” button to hear the “uw” phoneme.
6. We need to replace the “aw l k s” phonemes in the “Phonemes” text box in the “Word” pane of the dialog with the “uw” from the “Sounds Like” pane so copy that phoneme up to the top, making the list of phonemes in the word pane “p r uw 1”.
7. Click the red “mouth/lips” button next to the phonemes in the top part of the dialog to hear how the word “Proulx” will be pronounced. Sure enough, “Proulx” now rhymes with “Sue”.
8. Click the “save” button and you’ve defined a new pronunciation for the name “Proulx”.

11.3 Adding Custom Pronunciations to Your Application

Once you have defined your custom pronunciations, you’ll need to tell your application about them. Microsoft SAPI 5.1 allows for the creation of user and application dictionaries. The intent of this feature is that the application dictionary includes pronunciations that apply to all users while the user dictionary is unique to an individual user but you may use them any way you wish. The AT&T Natural Voices TTS SDK allows an application to define any number of custom dictionaries, and to control the order in which they are searched.

When searching for a transcription, the TTS Server first searches custom dictionaries, in the order specified by the client application, and then its own internal dictionaries to find some pronunciation for the word. The search stops as soon as a pronunciation is found.

You’ll find code samples for adding custom dictionaries to your application in Chapter 5 which describes the SDK.

Alternatively, you can update the `tts.cfg` file which the engine uses during engine initialization. You’ll find tts.cfg in the data subdirectory, e.g.

```
C:\program files\attnaturalvoices\tts1.4\desktop\data\tts.cfg
```

The `tts.cfg` file describes all of the voices and languages that are available to the TTS engine. Browse through the file to find the “language” section for the language for which you’re defining the dictionary, e.g.

```
Language                en_us
LanguageLocale          en_us
LanguageDictionary      en_us\en_us.dict att_cecilbet_english
LanguageTextAnalysis    en_us\fe_en_us.dll
```

is the section for the US English language. To use a dictionary file `en_us\mydict.dict`, add a new line to `tts.cfg` as follows:

```
Language                en_us
LanguageLocale          en_us
LanguageDictionary      en_us\en_us.dict att_cecilbet_english
LanguageTextAnalysis    en_us\fe_en_us.dll
UserDictionary          en_us\mydict.dict
```

The TTS engine will use the transcriptions in mydict.dict.

12 Performance Guidelines

This chapter provides some heuristics for measuring TTS Server performance and offers suggestions on sizing the load for your application's needs. First, we discuss the memory requirements of the TTS engine processes then we describe some relevant measurements and heuristics for optimizing those measurements.

12.1 TTS Memory Requirements

Table 12.1.1 lists the approximate memory requirements for the AT&T Natural Voices TTS engine processes using the Crystal US English female voice. Note that the first channel loads large data files into shared memory and all of this data is shared across all subsequent channels. Loading these data structures into memory on the first Speak() request is the cause of the initial delay.

Table 12.1.1. Memory requirements

Edition	First Channel	Subsequent Channels
Server	150 MB	10 MB
Server-Lite	94 MB	10 MB
Desktop	94 MB	10 MB

Note that the server need not have physical memory for each channel since the operating system uses paging to optimize the physical memory.

12.2 Definitions

Performance measurements concentrate on three criteria:

Time-to-first-audio (TTFA): The elapsed time from when the client first sends text to the server to when the client receives the first audio buffer from the server, i.e.,

TTFA = time first audio received - time first text sent

A large TTFA will be interpreted by an end user as a negative experience. Note that by default, an audio buffer contains 500 milliseconds of audio so there will be a minimum delay until the first 500 milliseconds of audio is produced by the audio. TTFA also includes network delays when using the Server Edition.

Audio-buffer-underflow (ABU): This measures the rate at which audio is sent from the TTS server to the client. If at any time the client has played all received buffers and is waiting longer than 500 milliseconds for another buffer from the server to arrive, then an underflow condition has arisen. This can result in the user "hearing" prolonged silence in the midst of a Speak; it implies that the TTS server is not running in at least real time or the network is experiencing delays. An ABU condition exists if the following is true:

$$t_0 + (\text{TotalBytesReceived}/8K) < t_i$$

where

- t_0 is the time the first buffer arrived
- t_i is the time the i^{th} buffer arrived
- TotalBytesReceived is the total number of bytes received before t_i
- There are 8K bytes of audio per second
- The expression *TotalBytesReceived/8K* represents the total time it takes for the audio object to play the audio it has received.

Real Time Ratio (xRT): The xRT is defined as

This ratio gives a rough indication of the real time processing of the audio stream. For example, if the TTS server takes 0.50 seconds to synthesize a string that takes 2.0 seconds to speak, then the xRT is 0.25, i.e. the server synthesized the text in 25% of the time needed to listen to the audio. A ratio greater than one is an indication of a problem since the server takes longer to create the audio than to play it back which implies that the server cannot keep up.

A TTFA, ABU, or xRT performance problem condition implies any one of a number of system problems: network delays, CPU overload, memory overload, process scheduling delays, etc. Therefore, system resources need to be monitored (e.g., via *sar*, *perfmon,Windows Performance Tool*) while the tests are conducted. All three measures can be calculated from the trace output of the clients and servers.

12.3 Performance Results

The performance results presented here are collected from a reference task, reading email messages, that is a typical use of TTS technology. Your results may vary but our intent is to provide data points that you can use to size your application's needs. Note that there are many factors that will affect your performance such as other applications that are running on the server, network latencies, and other events, both predictable and unpredictable.

The reference task that we've constructed for performance testing emulates an email reader which sends the email message to the TTS Server to be processed. An email

message includes three headers and a body. Headers tend to be relatively short, e.g. fewer than 10 words, and message bodies may be short, medium or long.

The client application starts *N* simultaneous channels of TTS, each doing a Speak() call with three message headers and one message body. The client application makes a Speak() request and then synchronously waits for the results to be returned from the server. Once the audio is returned, the client application immediately sends another request – the client application does *not* play back the audio or sleep for the duration of the audio. Note that this test is designed to determine the maximum server throughput to provide a worst-case scenario. An alternative is for the client application to sleep for the duration of the spoken message to emulate one potential implementation however this alternative approach would deliver higher, and potentially misleading, channel capacity.

The client application is run for several minutes before collecting performance data to allow all of the resources required by the client application and the TTS server to be loaded to better emulate a real service. In all of the results described below, the client application does not reside on the same machine as the server so all results include network latencies.

Our test results show that a Server Edition running on a 500MHz Intel Pentium-III processor with 512 MB of memory can support around 32 simultaneous channels of TTS with reasonable performance. The Server Edition running on a 1 GHz Intel Pentium-4 with 1 GB of memory supports about 48 simultaneous channels of TTS. In both cases, the TTS server computer is not running any other applications.

The Server-Lite Edition supports approximately half as many channels as the Server Edition, so the Server-Lite Edition supports approximately 16 channels on a 500 MHz 500MHz Intel Pentium-III processor with 512 MB of memory and approximately 24 channels on a 1 GHz Intel Pentium-4 with 1 GB of memory. Predicting channel capacity with the Server-Lite Edition is much more challenging because both the TTS engine and application are running on the same computer and are competing for system resources such as memory and CPU time.

The Desktop Edition is throttled to support only one channel at a time.

12.4 Recommendations

The performance results shown above are intended as guidelines to help you size your application needs. Keep in mind that the Reference Task used for these benchmarks continually sends new Speak requests to the server rather than pausing to play back the audio so your performance results are likely to be better. You'll need to experiment with your application to see how the TTS Server works with your application but you may want to keep the following heuristics in mind.

- The first call to SetVoice(), typically made indirectly by the Speak() call, requires all editions of the TTS engine to load large data files into memory. The TTFA on the first speak call is likely to be much longer than subsequent calls. You may want to call Speak() with an empty string to force the data to be loaded before the first Speak() request for a "real" user.
- You can improve TTFA by using shorter sentences or include punctuation to break sentences into phrases.
- TTFA does not improve dramatically as the CPU speed increases.

- TTFA decreases as the number of channels increases.
- Faster CPUs support more channels than slower CPUs with similar TTFA and ABU/Speak ratios. Add faster CPUs to add more channels.
- Allow at least 512MB of memory per CPU, 1GB if you are using multiple voices.
- Users are most likely to notice slow TTFA behavior. Tune your application or provide hardware to keep an average TTFA of < 2 seconds.
- Adding memory beyond 512 MB per CPU is likely to improve TTFA and channel capacity when running more than 32 channels per CPU.
- A 500 MHz PIII Intel with 512 MB of memory will support around 32 simultaneous channels with the Server Edition or around 16 channels with the Server-Lite Edition.
- A 1 GHz PIII Intel with 512 MB of memory running the Server Edition will support around 48 simultaneous channels with TTFA < 2 seconds for many applications.
- The performance you experience will depend on a number of factors including the type of text to be synthesized and network delays.

Appendix A: Phonetic Alphabets

The AT&T Natural Voices TTS package allows you to specify the phonetic pronunciation of a word both in the context of control tags and dictionaries. Phonemes may also be returned by notifications. Unfortunately, there's no single choice of phonetic alphabets so we offer IPA, DARPA, SAMPA, and SAPI phonetic alphabets. The phonetic alphabet you use depends upon the language and the interface you're using with the TTS engine:

	SAPI 4	SAPI 5	SSML	JSML
US English	IPA/English	DARPA	DARPA or IPA/English	IPA/English
Spanish	IPA/Spanish	SAMPA/Spanish	SAMPA/Spanish or IPA/Spanish	IPA/Spanish
German	IPA/German	SAMPA/German	SAMPA/German or IPA/German	IPA/German
UK English	IPA/UK English	SAMPA/UK English	SAMPA/UK English or IPA/UK English	IPA/UK English

A.1. IPA Phonetic Alphabets

The IPA phonetic alphabet supports all languages but we've broken the complete set into subsets for US English, Spanish, and German for your convenience. The following tables describe the subsets of the IPA phonetic alphabet that are suppported in the Natural Voices product.

A.1.1 US English SAPI 4/IPA Phonetic Alphabet

SAPI 4/DARPA Symbol	IPA Symbol	Example	Transcription
aa	ɑ	Bob	b aa b 1
ae	æ	bat	b ae t 1
ah	ʌ	but	b ah t 1
ao	ɔ	bought	b ao t 1
aw	aʊ	down	d aw n 1
ax	ə	about	ax 0 b aw t 1
ay	aɪ	bite	b ay t 1
b	b	bet	b eh t 1
ch	ʧ	church	ch er ch 1
d	d	dig	D ih g
dh	ð	that	dh ae t 1
dx	ɾ	butter	b ah 1 dx er 0
eh	ɛ	bet	b eh t 1
em	m	Chatham	ch ae 1 dx em 0
en	n̩	satin	s ae 1 q en 0
er	ɚ	bird	b er d 1
ey	ɛɪ	bait	b ey t 1
f	f	fog	f ao g 1
g	g	got	g aa t 1
h	h	hot	h aa t 1
hh	h	hot	hh aa t 1
ih	ɪ	bit	b ih t 1
iy	i:	beat	b iy t 1
jh	ʤ	jump	jh ah m p 1
k	k	cat	k ae t 1
l	l	lot	l aa t 1
m	m	Mom	m aa m 1
n	n	nod	n aa d 1
ng	ŋ	sing	s ih ng 1
ow	ɔʊ	boat	b ow t 1

oy	ɔɪ	boy	b oy 1
p	p	pot	p aa t 1
q	t	button	b ah 1 q en 0
r	ɹ	rat	r ae t 1
s	s	sit	s ih t 1
sh	ʃ	shut	sh ah t 1
t	t	top	t aa p 1
th	θ	thick	th ih k 1
uh	ʊ	book	b uh k 1
uw	u:	boot	b uw t 1
v	v	vat	v ae t 1
w	w	won	w ah n 1
y	j	you	y uw 1
z	z	zoo	z uw 1
zh	ʒ	measure	m eh 1 zh er 0
0		Unstressed	
1		Primary stress	
2		Secondary stress	
- (hyphen)		Syllable boundary	
&		Word boundary	
!		Sentence terminator	
, (comma)		Sentence terminator	
. (period)		Sentence terminator	
?		Sentence terminator	
_ (underscore)		Silence	
pau		Silence	

A.1.2. Spanish SAPI 4/IPA Phonetic Alphabet

SAPI 4/SAMPA Phoneme Symbol	IPA Symbol	Example	Transcription
p	p	padre	p a 1 d r e
b	b	vino	b i 1 n o
t	t	tomo	t o 1 m o
d	d	donde	d o n 1 d e
k	k	casa	k a 1 s a
g	g	gata	g a 1 t a
tS	ʧ	mucho	m u 1 tS o
jj	ʤ	hielo	jj e 1 l o
f	f	fácil	f a 1 s i l
B	β	cabra	k a 1 B r a
th	θ	cinco	th i N 1 k o
T	θ	cinco	T i N 1 k o
s	s	sala	s a 1 l a
x	x	mujer	m u x e r 1
G	ɣ	luego	l w e 1 G o
m	m	mismo	m i 1 s m o
n	n	nunca	n u n 1 k a
J	ɲ	año	a 1 J o
l	l	lejos	l e 1 x o s
ll		caballo	k a b a 1 ll o
L	ʎ	caballo	k a b a 1 L o
r	ɾ	puro	p u 1 r o
rr	r	torre	t o 1 rr e
j	j	rei	rr e 1 j
j	j	pie	p j e 1
w	w	deuda	d e 1 w d a
w	w	muy	m w i 1
i	i	pico	p i 1 k o

e	e	pero	p e 1 r o
a	a	valle	b a 1 L e
o	o	toro	t o 1 r o
u		duro	d u 1 r o
pau		silence	
1		Primary stress	

A.1.3. German SAPI 4/IPA Phonetic Alphabet

SAPI 4/SAMPA Phoneme Symbol	IPA Symbol	Example	Transcription
p	p	pein	p aI 1 n
b	b	bein	b aI 1 n
t	t	teich	t aI 1 C
d	d	Deich	d aI 1 C
k	k	kunst	k U 1 n s t
g	g	gunst	g U 1 n s t
pf	pf	Pfahl	p f a: 1 l
ts	ʦ	Zahl	ts a: 1 l
tS	tʃ	deutsch	d OY 1 tS
ch		deutsch	d OY 1 ch
dZ	dʒ	Dschungel	dZ U 1 N l
jh		Dschungel	jh U 1 ng l
f	f	fast	f a 1 s t
v	v	was	v a 1 s
s	s	Tasse	t a 1 s @
z	z	Hase	h a: 1 z @
S	ʃ	waschen	v a 1 S n
sh		waschen	v a 1 sh n
Z	ʒ	Genie	Z e 1 n i:
zh		Genie	zh e 1 n iy
C	ç	sicher	s I 1 C 6
x		sicher	s ih 1 x ax
j	j	Jahr	j a: 1 6
y		Jahr	y a : 1 ax
x	x	Buch	b u: 1 x
h	h	Hand	h a 1 n t
m	m	mein	m aI 1 n
n	n	nein	n aI 1 n

N	ŋ	Ding	d I 1 N
ng		Ding	d ih 1 ng
l	l	Leim	l al 1 m
R	ʀ	Reim	R al 1 m
r	R	Reim	r al 1 m
I		Sitz	z I 1 ts
ih		Sitz	z ih 1 ts
E		Gesetz	g @ 1 z E ts
eh		Gesetz	g ax 1 z eh ts
a	a	Satz	z a 1 ts
aa		Satz	z aa 1 ts
O	ɔ	Trotz	t r O 1 ts
oh	ɔ	Trotz	t r oh 1 ts
U	ʊ	Schutz	S U 1 ts
uh	ʊ	Schutz	sh uh 1 ts
Y	Y	hübsch	h Y 1 p S
uy		hübsch	h uy 1 p sh
9	œ	plötzlich	p l 9 1 ts l I C
oe	œ	plötzlich	p l oe 1 ts l ih ch
i:	i:	Lied	l i: 1 t
iy	i:	Lied	l iy 1 t
e:	e:	Beet	b e: 1 t
ey	e:	Beet	b ey 1 t
E:	ɛ:	spät	S p E: 1 t
eh :	ɛ:	spät	sh p eh : 1 t
a:	a:	Tat	t a: 1 t
aa :		Tat	t aa : 1 t
o:	o:	rot	r o: 1 t
ow		rot	r ow 1 t
u:	u:	Blut	b l u: 1 t
uw		Blut	b l uw 1 t
y:	y:	süß	z y: 1 s
ue		süß	z uy 1 s

2:	ø:	blöd	b l 2: 1 t
eu	ø:	blöd	b l eu 1 t
aI	aɪ	Eis	aI 1 s
ay		Eis	ay 1 s
aU	aʊ	Haus	h au 1 s
aw		Haus	h aw 1 s
OY	ɔɪ	Kreuz	k r OY 1 ts
oy		Kreuz	k r oy 1 ts
@	ə	bitte	b I 1 t @
ax		bitte	b ih 1 t ax
?	ʔ	Verein	f E 6 1 ? aI n
^		Verein	f eh ax 1 ^ ay n
pau		silence	
1		Primary stress	

A.1.4 French SAPI 4/IPA Phonetic Alphabet

SAPI 4/SAMPA Symbol	IPA Symbol	Example	Transcription
p	p	**p**ont	p o~ 1
b	b	**b**on	b o~ 1
t	t	**t**errain	t E 0 R e~ 1
d	d	**d**ix	d i s 1
k	k	**c**ol	k O l 1
g	g	**g**arçon	g a R 0 s o~ 1
dZ	ʤ	**j**azz (non-native)	a z 1
f	f	**f**ort	f O R 1
v	v	**v**ache	v a S 1
s	s	**s**el	s E l 1
z	z	**z**éro	z e 0 R o 1
S	ʃ	**ch**eval	S 9 0 v a l 1
Z	ʒ	**j**eu	Z 2 1
m	m	**m**ort	m O R 1
n	n	**n**om	n o~ 1
J	ɲ	a**gn**eau	a 0 J o 1
N	ŋ	campi**ng**	k a~ 0 p i N 1
l	l	**l**ivre	l i 1 v R @ 0
R	ʁ	**r**ouge	R u Z 1
r	ɹ	**r**ide (english)	R i d 1
h	h	**h**ome (english)	o m 1
j	j	p**i**erre	p j E R 1
w	w	**ou**i	w i 1
H	ɥ	p**u**its	p H i 1
i	i	t**y**pe	t i p 1
e	e	**é**cran	e 0 k R a~ 1
E	ɛ	f**e**r	f E R 1
a	a	p**a**tte	p a t 1
o	o	**eau**	o 1

O	ɔ	s**o**rt	s O R 1
u	u	r**oue**	R u 1
y	y	m**û**r	m y R 1
2	ø	d**eux**	d 2 1
9	œ	p**eu**r	p 9 R 1
@	ə	tabl**e**	t a 1 b l @ 0
{	æ	b**a**t (english)	b a 1
aU	aʊ	h**ou**se (english)	
a~	ɑ̃	s**ang**	s a~ 1
e~	ɛ̃	f**in**	f e~ 1
o~	ɔ̃	l**ong**	l o~ 1
0		Unstressed	
1		Primary stress	
2		Secondary stress	
- (hyphen)		Syllable boundary	
&		Word boundary	
!		Sentence terminator	
, (comma)		Sentence terminator	
. (period)		Sentence terminator	
?		Sentence terminator	
_ (underscore)		Silence	
pau		Silence	

A.1.5 UK English SAPI 4/IPA Phonetic Alphabet

SAPI 4/DARPA Symbol	IPA Symbol	Example	Transcription
p	p	**p**oint	p OI n t 1
b	b	**b**ig	b I g 1
t	t	**t**eam	t i: m 1
d	d	**d**are	d e @ 1
k	k	**c**ase	k eI s 1
g	g	**g**ood	g U d 1
dZ	dʒ	**g**in**g**er	dZ I n 1 dZ @ 0
tS	tʃ	**ch**eck	tS e k 1
f	f	**f**ool	f u: l 1
v	v	**v**est	v e s t 1
D	ð	**th**is	D I s 1
T	θ	**th**ick	T I k 1
s	s	**s**ell	s e l 1
z	z	**z**eal	z i: l 1
S	ʃ	**sh**oot	S u: t 1
Z	ʒ	mea**s**ure	m e 1 Z @ 0
h	h	**h**ouse	h aU s 1
m	m	**m**ain	m eI n 1
n	n	**n**ame	n eI m 1
N	ŋ	si**ng**	s I N 1
l	l	**l**ife	l aI f 1
@l	ʌ	bott**le**	b Q 1 t @l 0
r	ɹ	**r**ight	r aI t 1
j	j	**y**es	j e s 1
w	w	**w**ood	w U d 1
i:	i:	b**ea**t	b i: t 1
I	ɪ	b**i**t	b I t 1
eI	eɪ	b**ai**t	b eI t 1
e	ɛ	b**e**t	b e t 1
A:	ɑ:	f**a**ther	f A: 1 D @ 0

{	æ	b**a**t	b { t 1
@U	əʊ	b**oa**t	b @U t 1
O:	ɔ:	b**ough**t	b O: t 1
Q	ɒ	b**o**ss	b Q s 1
u:	u:	b**oo**t	b u: t 1
U	ʊ	b**oo**k	b U k 1
V	ʌ	b**u**t	b V t 1
3:	ɜ:	b**ir**d	b 3: d 1
aU	aʊ	b**ou**t	b aU t 1
OI	ɔɪ	b**oy**	b OI 1
aI	aɪ	b**i**te	b aI t 1
@	ə	scall**o**p	s k { 1 l @ p 0
I	ɪ	b**e**lieve	b I 0 l i: v 1
0		Unstressed	
1		Primary stress	
2		Secondary stress	
- (hyphen)		Syllable boundary	
&		Word boundary	
!		Sentence terminator	
, (comma)		Sentence terminator	
. (period)		Sentence terminator	
?		Sentence terminator	
_ (underscore)		Silence	
pau		Silence	

A.2.1 The SAPI 5 and DARPA US English Phonetic Alphabets

The SAPI 5.1 client accepts only the SAPI Phonetic alphabet. The Client SDK and the engine support only the DARPA phonetic alphabet. The SAPI 5.1 client maps the SAPI phonemes to the corresponding DARPA phonemes automatically.

Phoneme Symbol	Example	Transcription	SAPI and/or DARPA
aa	Bob	b aa b 1	DARPA, SAPI
ae	bat	b ae t 1	DARPA, SAPI
ah	but	b ah t 1	DARPA, SAPI
ao	bought	b ao t 1	DARPA, SAPI
aw	down	d aw n 1	DARPA, SAPI
ax	about	ax 0 b aw t 1	DARPA, SAPI
ay	bite	b ay t 1	DARPA, SAPI
b	bet	b eh t 1	DARPA, SAPI
ch	church	ch er ch 1	DARPA, SAPI
d	dig	D ih g	DARPA, SAPI
dh	that	dh ae t 1	DARPA, SAPI
dx	butter	b ah 1 dx er 0	DARPA only
eh	bet	b eh t 1	DARPA, SAPI
em	Chatham	ch ae 1 dx em 0	DARPA only
en	satin	s ae 1 q en 0	DARPA only
er	bird	b er d 1	DARPA, SAPI
ey	bait	b ey t 1	DARPA, SAPI
f	fog	f ao g 1	DARPA, SAPI
g	got	g aa t 1	DARPA, SAPI
h	hot	h aa t 1	SAPI only
hh	hot	hh aa t 1	DARPA only
ih	bit	b ih t 1	DARPA, SAPI
iy	beat	b iy t 1	DARPA, SAPI
jh	jump	jh ah m p 1	DARPA, SAPI
k	cat	k ae t 1	DARPA, SAPI
l	lot	l aa t 1	DARPA, SAPI
m	Mom	m aa m 1	DARPA, SAPI

n	nod	n aa d 1	DARPA, SAPI
ng	sing	s ih ng 1	DARPA, SAPI
ow	boat	b ow t 1	DARPA, SAPI
oy	boy	b oy 1	DARPA, SAPI
p	pot	p aa t 1	DARPA, SAPI
q	button	b ah 1 q en 0	DARPA only
r	rat	r ae t 1	DARPA, SAPI
s	sit	s ih t 1	DARPA, SAPI
sh	shut	sh ah t 1	DARPA, SAPI
t	top	t aa p 1	DARPA, SAPI
th	thick	th ih k 1	DARPA, SAPI
uh	book	b uh k 1	DARPA, SAPI
uw	boot	b uw t 1	DARPA, SAPI
v	vat	v ae t 1	DARPA, SAPI
w	won	w ah n 1	DARPA, SAPI
y	you	y uw 1	DARPA, SAPI
z	zoo	z uw 1	DARPA, SAPI
zh	measure	m eh 1 zh er 0	DARPA, SAPI
0	Unstressed		DARPA only
1	Primary stress		DARPA, SAPI
2	Secondary stress		DARPA, SAPI
- (hyphen)	Syllable boundary		SAPI only
&	Word boundary		DARPA, SAPI
!	Sentence terminator		SAPI only
, (comma)	Sentence terminator		SAPI only
. (period)	Sentence terminator		SAPI only
?	Sentence terminator		SAPI only
_ (underscore)	Silence		SAPI only
pau	Silence		AT&T only

A.2.2. SAPI 5 SAMPA Spanish Phonetic Alphabet

	Phoneme Symbol	Example	Transcription	SAPI and/or SAMPA
Plosives				
	p	padre	p a 1 D r e	SAPI, SAMPA
	b	vino	b i 1 n o	SAPI, SAMPA
	t	tomo	t o 1 m o	SAPI, SAMPA
	d	donde	d o n 1 D e	SAPI, SAMPA
	k	casa	k a 1 s a	SAPI, SAMPA
	g	gata	g a 1 t a	SAPI, SAMPA
affricates				
	tS	mucho	m u 1 tS o	SAMPA
	ch	mucho	m u 1 ch o	SAPI
	jj	hielo	jj e 1 l o	SAPI, SAMPA
fricatives				
	f	fácil	f a 1 s i l	SAPI, SAMPA
	B	cabra	k a 1 B r a	SAPI, SAMPA
	th	cinco	th i N 1 k o	SAPI
	T	cinco	T i N 1 k o	SAMPA
	D	nada	n a 1 D a	SAPI, SAMPA
	s	sala	s a 1 l a	SAPI, SAMPA
	x	mujer	m u x e r 1	SAPI, SAMPA
	G	luego	l w e 1 G o	SAPI, SAMPA
	z	mismo	m i 1 z m o	SAMPA
nasals				
	m	mismo	m i 1 z m o	SAPI, SAMPA
	n	nunca	n u N 1 k a	SAPI, SAMPA
	N	tango	t a n 1 g o	SAPI, SAMPA
	nj	año	a 1 nj o	SAPI
	J	año	a 1 J o	SAMPA
liquids				
	l	lejos	l e 1 x o s	SAPI, SAMPA
	ll	caballo	k a b a 1 ll o	SAPI

	L	caballo	k a b a 1 L o	SAMPA
	r	puro	p u 1 r o	SAPI, SAMPA
	rr	torre	t o 1 rr e	SAPI, SAMPA
semivowels				
	j	rei	rr e 1 j	SAPI, SAMPA
	j	pie	p j e 1	SAPI, SAMPA
	w	deuda	d e 1 w D a	SAPI, SAMPA
	w	muy	m w i 1	SAPI, SAMPA
vowels				
	i	pico	p i 1 k o	SAPI, SAMPA
	e	pero	p e 1 r o	SAPI, SAMPA
	a	valle	b a 1 L e	SAPI, SAMPA
	o	toro	t o 1 r o	SAPI, SAMPA
	u	duro	d u 1 r o	SAPI, SAMPA
Other				
	pau	silence		AT&T only
	1	Primary stress		AT&T only

A.2.3. SAPI 5 SAMPA German Phonetic Alphabet

	Phoneme Symbol	Example	Transcription	SAPI and/or SAMPA
Plosives				
	p	pein	p al 1 n	SAPI, SAMPA
	b	bein	b al 1 n	SAPI, SAMPA
	t	teich	t al 1 C	SAPI, SAMPA
	d	Deich	d al 1 C	SAPI, SAMPA
	k	kunst	k U 1 n s t	SAPI, SAMPA
	g	gunst	g U 1 n s t	SAPI, SAMPA
affricates				
	pf	Pfahl	p f a: 1 l	SAPI,SAMPA
	ts	Zahl	ts a: 1 l	SAPI,SAMPA
	tS	deutsch	d OY 1 tS	SAMPA
	ch	deutsch	d OY 1 ch	SAPI
	dZ	Dschungel	dZ U 1 N l	SAMPA
	jh	Dschungel	jh U 1 ng l	SAPI
fricatives				
	f	fast	f a 1 s t	SAPI, SAMPA
	v	was	v a 1 s	SAPI, SAMPA
	s	Tasse	t a 1 s @	SAPI, SAMPA
	z	Hase	h a: 1 z @	SAPI, SAMPA
	S	waschen	v a 1 S n	SAMPA
	sh	waschen	v a 1 sh n	SAPI
	Z	Genie	Z e 1 n i:	SAMPA
	zh	Genie	zh e 1 n iy	SAPI
	C	sicher	s I 1 C 6	SAMPA
	x	sicher	s ih 1 x ax	SAPI
	j	Jahr	j a: 1 6	SAMPA
	y	Jahr	y a : 1 ax	SAPI
	x	Buch	b u: 1 x	SAPI, SAMPA
	h	Hand	h a 1 n t	SAPI, SAMPA

nasals				
	m	mein	m aI 1 n	SAPI, SAMPA
	n	nein	n aI 1 n	SAPI, SAMPA
	N	Ding	d I 1 N	SAMPA
	ng	Ding	d ih 1 ng	SAPI
	l	Leim	l aI 1 m	SAPI, SAMPA
	R	Reim	R aI 1 m	SAMPA
	r	Reim	r aI 1 m	SAPI
checked vowels				
	I	Sitz	z I 1 ts	SAMPA
	ih	Sitz	z ih 1 ts	SAPI
	E	Gesetz	g @ 1 z E ts	SAMPA
	eh	Gesetz	g ax 1 z eh ts	SAPI
	a	Satz	z a 1 ts	SAMPA
	aa	Satz	z aa 1 ts	SAPI
	O	Trotz	t r O 1 ts	SAMPA
	oh	Trotz	t r oh 1 ts	SAPI
	U	Schutz	S U 1 ts	SAMPA
	uh	Schutz	sh uh 1 ts	SAPI
	Y	hübsch	h Y 1 p S	SAMPA
	uy	hübsch	h uy 1 p sh	SAPI
	9	plötzlich	p l 9 1 ts l I C	SAMPA
	oe	plötzlich	p l oe 1 ts l ih ch	SAPI
free vowels				
	i:	Lied	l i: 1 t	SAMPA
	iy	Lied	l iy 1 t	SAPI
	e:	Beet	b e: 1 t	SAMPA
	ey	Beet	b ey 1 t	SAPI
	E:	spät	S p E: 1 t	SAMPA
	eh :	spät	sh p eh : 1 t	SAPI
	a:	Tat	t a: 1 t	SAMPA
	aa :	Tat	t aa : 1 t	SAPI

	o:	rot	r o: 1 t	SAMPA
	ow	rot	r ow 1 t	SAPI
	u:	Blut	b l u: 1 t	SAMPA
	uw	Blut	b l uw 1 t	SAPI
	y:	süß	z y: 1 s	SAMPA
	ue	süß	z uy 1 s	SAPI
	2:	blöd	b l 2: 1 t	SAMPA
	eu	blöd	b l eu 1 t	SAPI
	aI	Eis	aI 1 s	SAMPA
	ay	Eis	ay 1 s	SAPI
	aU	Haus	h au 1 s	SAMPA
	aw	Haus	h aw 1 s	SAPI
	OY	Kreuz	k r OY 1 ts	SAMPA
	oy	Kreuz	k r oy 1 ts	SAPI
	@	bitte	b I 1 t @	SAMPA
	ax	bitte	b ih 1 t ax	SAPI
	?	Verein	f E 6 1 ? aI n	SAMPA
	^	Verein	f eh ax 1 ^ ay n	SAPI
Other				
	pau	silence		AT&T only
	1	Primary stress		AT&T only

A.2.4. SAPI 5 SAMPA French Phonetic Alphabet

Phoneme Symbol	Example	Transcription	SAPI and/or SAMPA
p	**p**ont	p o~ 1	SAPI, SAMPA
b	**b**on	b o~ 1	SAPI, SAMPA
t	**t**errain	t E 0 R e~ 1	SAPI, SAMPA
d	**d**ix	d i s 1	SAPI, SAMPA
k	**c**ol	k O l 1	SAPI, SAMPA
g	**g**arçon	g a R 0 s o~ 1	SAPI, SAMPA
dZ	**j**azz (non-native)	a z 1	SAMPA
zh	**j**azz (non-native)	a z 1	SAPI, SAMPA
f	**f**ort	f O R 1	SAPI, SAMPA
v	**v**ache	v a S 1	SAPI, SAMPA
s	**s**el	s E l 1	SAPI, SAMPA
z	**z**éro	z e 0 R o 1	SAPI, SAMPA
S	**ch**eval	S 9 0 v a l 1	SAMPA
sh	**ch**eval	sh 9 0 v a l 1	SAPI
Z	**j**eu	Z 2 1	SAMPA
zh	**j**eu	zh 2 1	SAPI
m	**m**ort	m O R 1	SAPI, SAMPA
n	**n**om	n o~ 1	SAPI, SAMPA
J	a**gn**eau	a 0 J o 1	SAMPA
jn	a**gn**eau	a 0 jn o 1	SAPI
N	campi**ng**	k a~ 0 p i N 1	SAMPA
ng	campi**ng**	k a~ 0 p i ng 1	SAPI
l	**l**ivre	l i 1 v R @ 0	SAPI, SAMPA
R	**r**ouge	R u Z 1	SAMPA
r	**r**ide (english)	r i d 1	SAPI, SAMPA
h	**h**ome (english)	o m 1	SAPI, SAMPA
hy	**h**ome (english)	o m 1	SAPI
j	p**i**erre	p j E R 1	SAMPA

w	**ou**i	w i 1	SAPI, SAMPA
H	p**u**its	p H i 1	SAMPA
hy	p**u**its	p hy i 1	SAPI
i	t**y**pe	t i p 1	SAMPA
e	**é**cran	e 0 k R a~ 1	SAMPA
E	f**e**r	f E R 1	SAMPA
eh	f**e**r	f eh R 1	SAPI
a	p**a**tte	p a t 1	SAPI, SAMPA
o	**eau**	o 1	SAMPA
ow	**eau**	ow 1	SAPI
O	s**o**rt	s O R 1	SAMPA
oh	s**o**rt	s oh R 1	SAPI
u	r**oue**	R u 1	SAMPA
uy	r**oue**	R uy 1	SAPI
y	m**û**r	m y R 1	SAPI, SAMPA
2	d**eux**	d 2 1	SAMPA
ue	d**eux**	d eu 1	SAPI
9	p**eu**r	p 9 R 1	SAMPA
oe	p**eu**r	p oe R 1	SAPI
@	tabl**e**	t a 1 b l @ 0	SAMPA
{	b**a**t (english)	b a 1	SAMPA
aU	h**ou**se (english)		SAMPA
aa	h**ou**se (english)		SAPI
a~	s**ang**	s a~ 1	SAPI, SAMPA
e~	f**in**	f e~ 1	SAPI, SAMPA
o~	l**ong**	l o~ 1	SAPI, SAMPA
0	Unstressed		SAMPA only
1	Primary stress		SAMPA, SAPI
2	Secondary stress		SAPI
- (hyphen)	Syllable boundary		SAPI only
&	Word boundary		SAMPA, SAPI
!	Sentence		SAPI only

	terminator		
, (comma)	Sentence terminator		SAPI only
. (period)	Sentence terminator		SAPI only
?	Sentence terminator		SAPI only
_ (underscore)	Silence		SAPI only
pau	Silence		AT&T only

A.2.5. SAPI 5 SAMPA UK English Phonetic Alphabet

SAMPA Symbol	Example	Transcription	SAPI and/or SAMPA
p	**p**oint	p OI n t 1	SAMPA, SAPI
b	**b**ig	b I g 1	SAMPA, SAPI
t	**t**eam	t i: m 1	SAMPA, SAPI
d	**d**are	d e @ 1	SAMPA, SAPI
k	**c**ase	k eI s 1	SAMPA, SAPI
g	**g**ood	g U d 1	SAMPA, SAPI
dZ	**g**in**g**er	dZ I n 1 dZ @ 0	SAMPA, SAPI
tS	**ch**eck	tS e k 1	SAMPA, SAPI
f	**f**ool	f u: l 1	SAMPA, SAPI
v	**v**est	v e s t 1	SAMPA, SAPI
D	**th**is	D I s 1	SAMPA, SAPI
T	**th**ick	T I k 1	SAMPA, SAPI
s	**s**ell	s e l 1	SAMPA, SAPI
z	**z**eal	z i: l 1	SAMPA, SAPI
S	**sh**oot	S u: t 1	SAMPA, SAPI
Z	mea**s**ure	m e 1 Z @ 0	SAMPA, SAPI
h	**h**ouse	h aU s 1	SAMPA, SAPI
m	**m**ain	m eI n 1	SAMPA, SAPI
n	**n**ame	n eI m 1	SAMPA, SAPI
N	si**ng**	s I N 1	SAMPA, SAPI
l	**l**ife	l aI f 1	SAMPA, SAPI
@l	bott**le**	b Q 1 t @l 0	SAMPA, SAPI
r	**r**ight	r aI t 1	SAMPA, SAPI
j	**y**es	j e s 1	SAMPA, SAPI
w	**w**ood	w U d 1	SAMPA, SAPI
i:	b**ea**t	b i: t 1	SAMPA, SAPI
I	b**i**t	b I t 1	SAMPA, SAPI
eI	b**ai**t	b eI t 1	SAMPA, SAPI
e	b**e**t	b e t 1	SAMPA, SAPI

A:	f**a**ther	f A: 1 D @ 0	SAMPA, SAPI
{	b**a**t	b { t 1	SAMPA, SAPI
@U	b**oa**t	b @U t 1	SAMPA, SAPI
O:	b**ough**t	b O: t 1	SAMPA, SAPI
Q	b**o**ss	b Q s 1	SAMPA, SAPI
u:	b**oo**t	b u: t 1	SAMPA, SAPI
U	b**oo**k	b U k 1	SAMPA, SAPI
V	b**u**t	b V t 1	SAMPA, SAPI
3:	b**ir**d	b 3: d 1	SAMPA, SAPI
aU	b**ou**t	b aU t 1	SAMPA, SAPI
OI	b**oy**	b OI 1	SAMPA, SAPI
aI	b**i**te	b aI t 1	SAMPA, SAPI
@	scall**o**p	s k { 1 l @ p 0	SAMPA, SAPI
I	b**e**lieve	b I 0 l i: v 1	SAMPA, SAPI
0	Unstressed		SAMPA, SAPI
1	Primary stress		SAMPA, SAPI
2	Secondary stress		SAMPA, SAPI
- (hyphen)	Syllable boundary		SAPI only
&	Word boundary		SAMPA
!	Sentence terminator		SAPI only
, (comma)	Sentence terminator		SAPI only
. (period)	Sentence terminator		SAPI only
?	Sentence terminator		SAPI only
_ (underscore)	Silence		SAPI only
pau	Silence		AT&T only

Appendix B: SAPI 5.1 Compliance

The AT&T Natural Voices TTS SAPI 5.1 client supports most of the SAPI 5.1 interfaces and features. This appendix describes the compliance in detail.

B.1. Support for SAPI 5.1

The AT&T Natural Voices TTS engine is also implemented as an OLE COM-compliant object on Microsoft Windows NT and Windows 2000 platforms. It works with either 8-bit ASCII text files or 16-bit Unicode text. See Chapter 5 for a list of the supported SAPI control tags and the table below for a list of supported interfaces.

Figure B.1. SAPI interfaces and functions supported by the AT&T Natural Voices TTS engine

Interface	Functions	Required?	Supported?
IUnknown	QueryInterface	Yes	Yes
	AddRef	Yes	Yes
	Release	Yes	Yes
ISpTTSEngine	Speak	Yes	Yes
	GetOutputFormat	Yes	Yes
ISpObjectWithToken	SetObjectToken	Yes	Yes
	GetObjectToken	Yes	Yes
User Lexicon		Yes	Yes
Application Lexicon		Yes	Yes