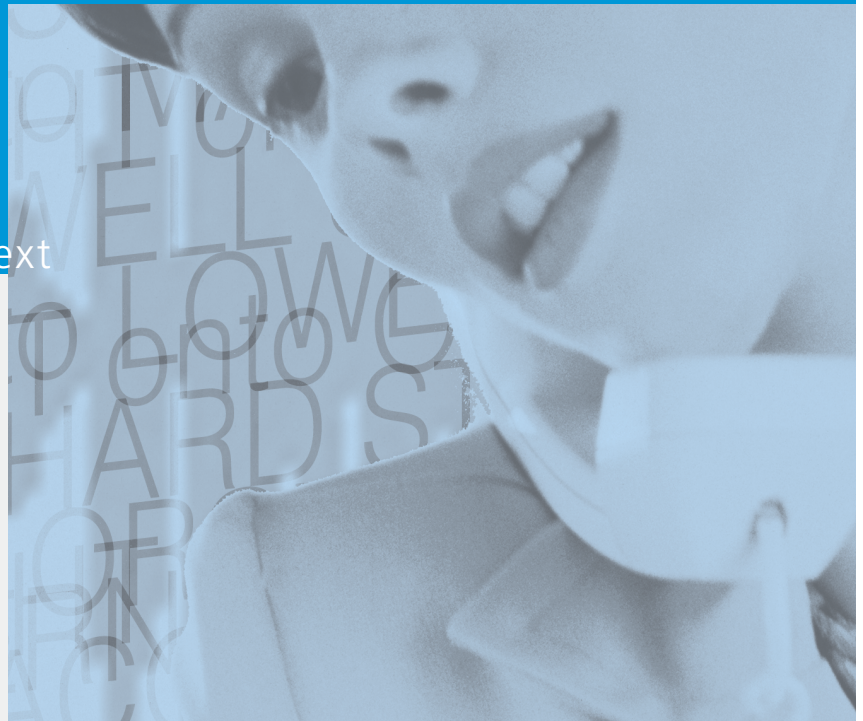


■ *White Paper*

Improving TTS Output  
by Controlling Input Text



## Introduction

Text-to-speech (TTS) or speech synthesis has greatly improved within the last 1-2 years. Synthetic voice quality has become so natural that some systems are almost indistinguishable from human speech. This makes blending TTS and pre-recorded prompts in an automated voice response system a compelling and viable solution. In fact, some developers have completely replaced pre-recorded prompts with synthetic output.

So why does synthetic speech still sometimes sound unnatural? Are there bugs or data problems that can be avoided? Frequently output problems are caused by the input text. Remember, TTS systems don't understand the text they read. They do not have a sense of the underlying meaning or context of what they are reading, so they are unable to make certain decisions about how a text should be read aloud. Because of this, TTS systems rely heavily on correctly spelled, well-punctuated, unambiguous input texts for clues as to how a text should be read.

When designing a TTS application, there are varying degrees of control a developer may have over the input text. If you do have control over the input text, there are some simple guidelines that can be applied to maximize the quality of the output speech. If you have predictably formatted text input (e.g., from database entries), it may be beneficial to implement a custom text pre-processor that reformats the database entries on the fly, in order to elicit the best quality TTS output. If you have no control over the text input (e.g., instant messaging), you must rely on the system's standard pre-processing capability.

The aim of this white paper is to provide the reader some simple guidelines that can be applied to maximize the quality of speech output. Its goals are two-fold:

- Make recommendations for preparing input text in such a way that the speech output is of the highest quality possible.
- Identify some of the causes of sub-optimal TTS output.

Based on its deep experience and long leadership in the TTS market, SpeechWorks Solutions is able to highlight

some of the common issues caused by input texts, and to make suggestions about how to fix or avoid them. There is also a brief case study—identifying and fixing problems with input text to improve the spoken output. ***Specific suggestions are in bold italics.***

The two key elements that should be considered in order to maximize the text to speech output of your speech solution are: context and text formatting.

## Context

Because the majority of people parse a text instantaneously when they read it, they frequently forget that they're actually carrying out a lot of complex tasks—including syntactic analysis (using the structure and order of the words), semantic analysis (using the meaning of the words), and pragmatic analysis (using the real-world context and tone of the text). Consider the following:

- (a) **From 1975-1979, I was studying at the university.**  
*From nineteen seventy five to nineteen seventy-nine, I was studying at the university*
- (b) **Our telephone number is (607) 266-7025.**  
*Our telephone number is six owe seven, two six six, seven owe two five.*

Let's focus just on the dash symbol in those sentences. In the first one, we instinctively know that the numbers surrounding the dash are years, and therefore the dash should be spoken as the word 'to'. In the second sentence, we can see that the sequence of digits makes up a telephone number, and therefore we should not pronounce the dash, rather we should pause, as we would for a comma.

***Test a TTS system's default text pre-processing capabilities thoroughly when designing your application. If your application contains non-standard text formatting, consider a custom text pre-processor\* to make the input text more palatable to the TTS system.***

There are other types of conversion that no TTS system can make reliably without over-generalizing. For example, consider sentence (c) below:

\* A custom text pre-processor is a piece of code that has been designed to deal with the unique text attributes found in a specific application, and format the text in such a way that it will elicit high-quality speech output. For example, an e-mail pre-processor filters out certain lines that the user doesn't want to hear, and formats the message in such a way that the TTS system's default processing can easily handle.

**(c) I read the newspaper from cover to cover every day.**

When you saw this, did you think of the past tense version of read (rhymes with red), or the present tense version (i.e., rhymes with reed)? It's hard to say which is correct. However it becomes obvious to us if we precede that sentence with this one:

**(d) I had absolutely nothing to do last week—it was wonderful.**

Obviously we would pick the past tense version of read in sentence (c) if it were preceded by sentence (d). However:

**(e) It's important that I'm up to date in my job.**

We would pick the present tense version of read in sentence (c) if it were preceded by sentence (e). The correct version only becomes obvious to us if we have knowledge about the tense of the sentence/s that come before it.

Our ability to take into account the knowledge of what's previously been discussed in a paragraph or dialog is not an ability currently shared by TTS systems. Therefore, a TTS system almost always reads a sentence as a single, context-independent item.

As another example, consider the different pronunciations of the word lives in the following two sentences:

**(f) Met a crazy guy skydiving yesterday—lives on the edge.**

**(g) The plight of children in Rio—lives on the edge.**

Words that are spelled the same but have different pronunciations (like lives and read) are called homographs. There is a large variation in the abilities of different TTS systems to deal with them. Homographs are not always full words—abbreviations can behave as homographs too. For example 'Corp' may be pronounced as 'Corporal' and 'Corporation'.

Some spellings can even be full words or abbreviations, for example 'Wed' ('to marry' and 'Wednesday'), 'Sat' ('past tense of sit' and 'Saturday') and 'No' ('opposite of yes' and 'Number')

***Test the TTS system's ability to deal with homographs. If the system doesn't correctly deal with the homograph, re-word the text where possible.***

So far we've discussed how context affects the way in which a text is pre-processed, and how it affects the pronunciations of homographs.

The rule of thumb is to test the system's abilities in advance, and to attempt to avoid the system's limitations—some of which may be system-specific, some of which represent the state-of-the-art. However, equipped with this knowledge, you will be able to avoid many of the causes of poor TTS output in your deployed application.

Looking to the future, there is research into natural language understanding (NLU) and concept-to-speech (CTS) systems, which attempt to incorporate meaning and context into their analyses, but these are unlikely to be commercially deployable for some time. In the meantime, once you are familiar with the contexts that result in sub-optimal speech output, it is fairly easy to write in an unambiguous way that avoids the pitfalls, resulting in a higher quality conversational application.

One important point to make here is that you may be able to take advantage of speech-synthesis mark-up language (SSML) to resolve some of the issues mentioned above. SSML is a standard XML language (cf. <http://www.w3.org/TR/speech-synthesis>) that makes it possible to define how a specific word should be pronounced, which word in a sentence you want to emphasize, and so on, so that it may indeed be possible to synthesize most of the renditions discussed above. In addition to supporting this standard, SpeechWorks Solutions' RealSpeak™ Telecom, has a series of native tags that allow for control over the output speech, if your application cannot take advantage of SSML. SpeechWorks Solutions strongly recommends taking advantage of SSML or system-specific tags. If that is a realistic option in your application, it affords you greater freedom in your text design.

## Text Formatting

Three areas of text formatting frequently interact with the ability of a TTS system to carry out its standard text analysis.

- Punctuation
- Spelling
- Use of case

## Punctuation

Using punctuation to indicate pauses is critical to achieving the best quality TTS output. If well-punctuated text is not entered, then the TTS system will adopt one of two strategies: read the text as entered, or attempt to insert pauses itself. Inserting pauses intelligently into an unpunctuated string of text is one of the liveliest areas of speech synthesis research – and it is still not especially successful. Consider this sentence, taken from a news web site:

- (h) David Elkins arrived in San Francisco Wednesday afternoon to address striking janitors who walked out last Tuesday highlighting the plight of low-salary earners who are living in one of the most expensive cities in the US.**

When TTS systems read this text, there tend to be three main problems. First, the speech is perceived as too fast. The number of pauses in speech plays a major role in the perception of speaking rate, and the lack of punctuation in the sentence means that there are not many pauses in the TTS output (when people are asked to speak faster, one of the main strategies they adopt is to reduce the number of pauses in their speech). Second, the speech is less intelligible than it should be, because people like to hear words grouped together into meaningful phrases, rather than in long unbroken strings. Third, many TTS systems insert pauses into an input text if there is only light punctuation. While this can be beneficial, it does occasionally causes false positives, in which pauses are inserted in inappropriate places.

As an experiment, this text was entered into several commercial TTS systems, and all of the problems described above were observed. The output was significantly improved by liberally inserting punctuation throughout the text, as follows:

- (i) David Elkins arrived in San Francisco Wednesday afternoon, to address striking janitors who walked out last Tuesday, highlighting the plight of low-salary earners, who are living in one of the most expensive cities in the US.**

Punctuation usage is not simply a matter of preference, however—there is a real discrepancy between written and spoken conventions, as illustrated below:

- (j) I know the butcher, the baker and the candlestick maker.**

When reading this sentence aloud, it's better to insert a pause after *baker* in the spoken form of the sentence, even though no comma is required in the written form of the sentence.

Another issue to look for is the conceptual opposite of the one described above. Quite often, we ignore commas, instead of pausing. For example:

- (k) Hello, John.**  
**(l) It's not a good idea, though.**  
**(m) Could you pass me the sugar, please?**

We would rather have all of these sentences read as if there were no comma, but TTS systems often pause, resulting in rather unnatural output. Some other examples to try:

- (n) The building is very, very tall.**  
**(o) Ithaca, NY 14850**

***Insert punctuation liberally, but be sure not to insert punctuation where you don't want pauses—even if normal writing conventions would suggest punctuation.***

In addition to pauses in running text, people frequently pass *visually* formatted text into a TTS system, and are appalled by the output. For the majority of TTS systems, this can be problematic, as they will view input text as a continuous string. For example, consider this shopping list:

- (p) Two dozen eggs  
 4 quarts of Milk  
 Orange Juice  
 Bread (Whole Grain)**

Because there are no periods at the ends of the lines, most TTS systems will read this in one phrase “Two dozen eggs four quarts of milk orange juice bread, whole grain.” This is clearly not the intent of the author. On the one hand, the TTS system doesn’t have the pragmatic context to know it’s dealing with a list; on the other hand it would create all sorts of new problems if the system behavior were generalized to pause at the end of every line. There are HTML parsers that may help prevent this type of problem, but this white paper deals with typical default behavior. The reliable solution in the case of example (r) is to insert punctuation at the end of each list item.

As a second example, consider the use of font formatting in documents like this one, and on the web. **Bold**, *italicized*, underlined and **colored** text usually just looks like plain text by the time it reaches a TTS system. If you want this type of formatting to change the way in which the text is read, you should re-write it for use in a synthesis application.

***The clarity of visually formatted text is not typically preserved in TTS. Either avoid using fonts, bulleted lists, and colors with the intention of affecting the output speech, or implement a parser to automatically reformat the text for synthesis.***

### Spelling

Proper spelling may seem like an obvious point, but TTS systems don’t possess spell-checkers—they read precisely what you enter. In the following sentence *The weahter is expected to be cloudy this morning*, we can tell that the second word is meant to be *weather*, but most TTS systems will read this as something like *wet her* or *wheat her*.

One company rather ambitiously sold its TTS engine as a proofreader to highlight spelling errors in documents. If you misspelled a word, they guaranteed the output would sound bad. In real deployments, a surprisingly large number of reported bugs turn out to be typos.

***If you have control over the input text, be sure to check that it is accurately spelled.***

### Use of Case

Consider some of the stylistic uses of upper case strings in a text:

**Acronyms** – He works for NASA.

**Initialisms** – She studies at MIT.

**Spellings** – It’s spelled S-C-Y-T-H-E.

**Emphasis** – Don’t do it THAT way.

**Emotions** – DROP EVERYTHING AND READ THIS MESSAGE!!!!!!

Most TTS systems have the ability to handle the first three cases in the list. A high quality text-to-speech engine should handle common acronyms and initialisms as would be expected by a speaker.

Uses of case that don’t follow one of the types mentioned above may well affect the system’s ability to process the text accurately. For example, many driving direction applications use upper case strings for all the street names, and normal case for directions: “Start out on ATLANTIC AVENUE.” This convention causes issues for most TTS systems, because they rely on case to make decisions about how words should be pronounced, as discussed below.

### Case Study: Navigation text

In this final section, SpeechWorks Solutions presents a brief case study focused on some common text found in vehicle navigation systems. We identify problems in the TTS output, attribute them to problems with the input text, and then make simple modifications to the text to improve the synthetic speech.

Consider the following text:

- 1) Turn LEFT onto MT VERNON ST. 0.26 miles**
- 2) Turn RIGHT onto MASSACHUSETTS AVE/MASS AVE. 0/76 miles**

At first glance, this looks trivial—it should be read aloud something like this:

*“One. Turn left onto Mount Vernon Street. Zero point two six miles. Two. Turn right onto Massachusetts Avenue. Zero point seven six miles.”*

We entered this text into several commercially available TTS systems, and here is an amalgamation of how the text was synthesized:

*“One. Turn left onto M T Vernon S T. Zero point two six miles two. Turn right onto Massachusetts ave slash mass ave. Zero point seven six miles.”*

Ave was most frequently read as it would be in *Ave Maria*. There are two types of problems to address here—both of which fall into the categories we’ve defined—text formatting and context.

- MT should be “Mount”
- ST should be “Street”
- AVE should be “Avenue”

These text-formatting errors seem to constitute the type of problem that most TTS systems should be able to solve with relative ease. However, look once more at the input text. The mixture of upper case strings in an otherwise normally typed sentence is problematic for most TTS systems.

- 1) Turn LEFT onto MT VERNON ST. 0.26 mile**
- 2) Turn RIGHT onto MASSACHUSETTS AVE/MASS AVE. 0.76 miles**

This is really an anomalous use of upper case—the upper case string doesn’t impart any useful information, and it doesn’t affect how a person would read the text aloud. That is, we wouldn’t emphasize the words unduly, the words aren’t acronyms or initialisms, we wouldn’t spell them out, and we wouldn’t load them with emotion.

The upper case sequences may have been used to highlight the salient information to a person reading directions from a printout, or perhaps it’s just coincidental. When the text was reentered with normalized case, SpeechWorks Solutions correctly expanded all of the abbreviations.

Next, we consider the context errors caused by the input.

- We don’t want to read out MASSACHUSETTS AVE and MASS AVE—one or the other would be better.
- As human readers, we can tell that this is a list of driving directions, so we should pause at the end of every line, even though there is no punctuation telling us to pause.
- If we happened to know the Boston area, we would know that we only expand AVE to Avenue if it is preceded by the string MASSACHUSETTS. If the string MASS precedes AVE, then it’s pronounced as Mass Av in ordinary usage.

All of these factors require real-world knowledge—knowledge that a generic TTS system is highly unlikely to possess.

So how could this be entered into a TTS system unambiguously, so that the TTS system gave us what we wanted? Based on the text formatting and context errors mentioned above, we made three simple modifications to the text:

- (1) We normalized the case of the text**
- (2) We added a period at the end of each line.**
- (3) We removed one of the references to Massachusetts Avenue.**

- 1) Turn left onto Mt. Vernon St. 0.26 miles**
- 2) Turn right onto Massachusetts Ave. 0.76 miles**

The output from RealSpeak was entirely correct. Normal case throughout the text enabled RealSpeak to spot the conversions it failed on previously:

- Mt -> Mount
- Ave -> Avenue
- St -> Street

In addition, by placing a period wherever we wanted a pause, we managed to get extremely good output. These were all simple modifications. We made them manually, but it would be not be difficult to make all of these automatically using a custom text pre-processor for a navigation application.

### Summary

In this paper, SpeechWorks Solutions has highlighted two types of text problems that degrade TTS output—context and text formatting. Some of the issues we’ve covered illustrate the current state of the art of TTS, rather than problems with specific systems; others are caused by anomalous input. We have suggested the following methods to overcome these issues, as they are applicable to your application:

- Thorough evaluation of a system’s text processing capabilities before making a purchase
- Thorough testing of the system’s capabilities, and avoidance of text input that cause poor output for that system
- Liberal and prudent use of punctuation
- Spell-checking
- Usual use of case
- Use of custom text pre-processors
- Use of SSML

- Where possible, use of output control tags that are native to the system, such as those found in RealSpeak

In addition, SpeechWorks Solutions strongly recommends defining a style guide for your application text, based on the topics covered in this white paper. This will enable you to achieve more reliable and consistently pleasing TTS output, as well as reducing the number and severity of output problems in your application.

**ScanSoft®**

SpeechWorks Solutions  
Division  
ScanSoft, Inc.  
9 Centennial Drive  
Peabody, MA 01960  
[www.ScanSoft.com](http://www.ScanSoft.com)

© 2004 ScanSoft, Inc. All rights reserved. ScanSoft and the ScanSoft logo, SpeechWorks, OpenSpeech, DialogModules, RealSpeak, SpeechPAK, and SpeechSecure are registered trademarks or trademarks of ScanSoft, Inc. in the United States and other countries. All other company or product names may be the trademarks of their respective owners.